Technical Report 271

Generating Semantic Descriptions From Drawings of Scenes With Shadows

David L. Waltz

MIT Artificial Intelligence Laboratory

GENERATING SEMANTIC DESCRIPTIONS FROM DRAWINGS OF SCENES WITH SHADOWS

David L. Waltz

November 1972

ARTIFICIAL INTELLIGENCE LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Cambridge Massachusetts 02139

ACKNOWLEDGEMENTS

I offer my thanks to the following people who contributed to this thesis:
to Patrick Winston who supervised, advised, and encouraged me in this project;
to Marvin Minsky, who served as a reader and whose ideas are largely responsible for the existence of this type of work;
to Jerry Sussman, Gene Freuder, Mark Dowson, and Max Clowes, who contributed many ideas to this work both in conversations and otherwise;
to David Huffman and Adolpho Guzman, from whose work I derived my initial motivation;
to Nick Horn, who served as a reader;
to William Henneman, for various and sundry reasons;
to my parents;
and to Bonnie Waltz, for her aid in preparing the figures, and especially for her patience and support (moral and financial) over the years, without which this work would not have been possible.

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.

The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

GENERATING SEMANTIC DESCRIPTIONS FROM DRAWINGS OF SCENES WITH SHADOWS*

Abstract

The research reported here concerns the principles used to automatically generate three-dimensional representations from line drawings of scenes. The computer programs involved look at scenes which consist of polyhedra and which may contain shadows and various kinds of coincidentally aligned scene features. Each generated description includes informationabout edge shape (convex, concave, occluding, shadow, etc.), about the decomposition of the scene into bodies, about the type of illumination for each region (illuminated, projected shadow, or oriented away from the light source), and about the spacial orientation of regions. The methods used are based on the labeling schemes of Huffman and Clowes; this research provides a considerable extension to their work and also gives theoretical explanations to the heuristic scene analysis work of Guzman, Winston, and others.

*This report reproduces a thesis of the same title submitted to the Department of Electrical Engineering, Massachusetts Institute of Technology, in partial fulfillment of the requirements for the degree of Doctor of Philosophy, September 1972.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS 2

ABSTRACT 3

TABLE OF CONTENTS 4

1.0 INTRODUCTION 6
- 1.1 DESCRIPTIONS 11
- 1.2 JUNCTION LABELS 12
- 1.3 JUNCTION LABEL ASSIGNMENT 14
- 1.4 COMBINATION RULES 16
- 1.5 EXPERIMENTAL RESULTS 18
- 1.6 COMPARISON WITH OTHER VISION PROGRAMS 23
- 1.7 HISTORICAL PERSPECTIVE 25
- 1.8 IMPLICATIONS FOR HUMAN PERCEPTION 26

2.0 QUICK SYNOPSIS 29
- 2.1 THE PROBLEM 29
- 2.2 SOLVING THE LABEL ASSIGNMENT PROBLEM 39
- 2.3 BETTER EDGE DESCRIPTION 43
- 2.4 PROGRAMMING CONSEQUENCES 55
- 2.5 HANDLING BAD DATA 62
- 2.6 ACCIDENTAL ALIGNMENT 64
- 2.7 MISSING LINES 67
- 2.8 REGION ORIENTATIONS 69

3.0 TRIHEDRAL JUNCTION LABELS 70
- 3.1 EDGE GEOMETRY 72
- 3.2 A USEFUL HEURISTIC 89
- 3.3 SHADOWS AT TRIHEDRAL VERTICES 93
- 3.4 OTHER NON-DEGENERATE JUNCTIONS 98
- 3.5 A CLASS OF DEGENERACIES 102

4.0 COMPLETING THE REGULAR DATA BASE 107
- 4.1 REGION ILLUMINATION ASSIGNMENTS 108
- 4.2 SUMMARY OF THE DATA BASE 118

5.0 SELECTION RULES 124
- 5.1 REGION BRIGHTNESS 124
- 5.2 SCENE/BACKGROUND BOUNDARY REVISITED 126
- 5.3 EXTENDING THE SUPPORT SURFACE 132
- 5.4 DISCUSSION 145
- 5.5 AN EXAMPLE 149

6.0	THE MAIN LABELING PROGRAM	151
6.1	A SMALL EXAMPLE	151
6.2	DISCUSSION	156
6.3	CONTROL STRUCTURE	164
6.4	PROGRAM PERFORMANCE	173
6.5	PERFORMANCE PROBLEMS	182
7.0	NON-TRIHEDRAL VERTICES & RELATED PROBLEMS	201
7.1	NON-TRIHEDRAL VERTICES	202
7.2	ACCIDENTAL ALIGNMENTS; FIRST TYPE	212
7.3	ACCIDENTAL ALIGNMENT WITHOUT OBSCURING EDGES	217
7.4	ACCIDENTAL ALIGNMENTS; FINAL TYPE	220
7.5	MORE CONTROL STRUCTURE	228
7.6	MISSING EDGES	238
7.7	HEURISTICS	243
8.0	REGION ORIENTATIONS	247
8.1	LINE LABEL ADDITIONS	247
8.2	AN EXAMPLE	254
8.3	GENERAL REGION ORIENTATIONS	257
8.4	GENERAL LINE DIRECTIONS	260
8.5	SUPPORT	276
9.0	HISTORICAL PERSPECTIVE	288
9.1	GUZMAN'S SEE PROGRAM	289
9.2	WORK AFTER GUZMAN; HUFFMAN & CLOWES	296
9.3	AN ACCOUNT OF MY EFFORTS	297
BIBLIOGRAPHY		302
APPENDIX 1		307
APPENDIX 2		321
APPENDIX 3		334
APPENDIX 4		338
APPENDIX 5		348

1.0 INTRODUCTION

How do we ascertain the shapes of unfamiliar objects? Why do we so seldom confuse shadows with real things? How do we "factor out" shadows when looking at scenes? How are we able to see the world as essentially the same 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 1.1 and 1.2? Do we use learning and knowledge to interpret what we see, or do we somehow automatically see the world as stable and independent of lighting? What portions of scenes can we understand from local features alone, and what configurations require the use of global hypotheses?

Various theories have been proposed to explain how people extract three-dimensional information from scenes (Gibson 1950 is an excellent reference). It is well known that we 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,

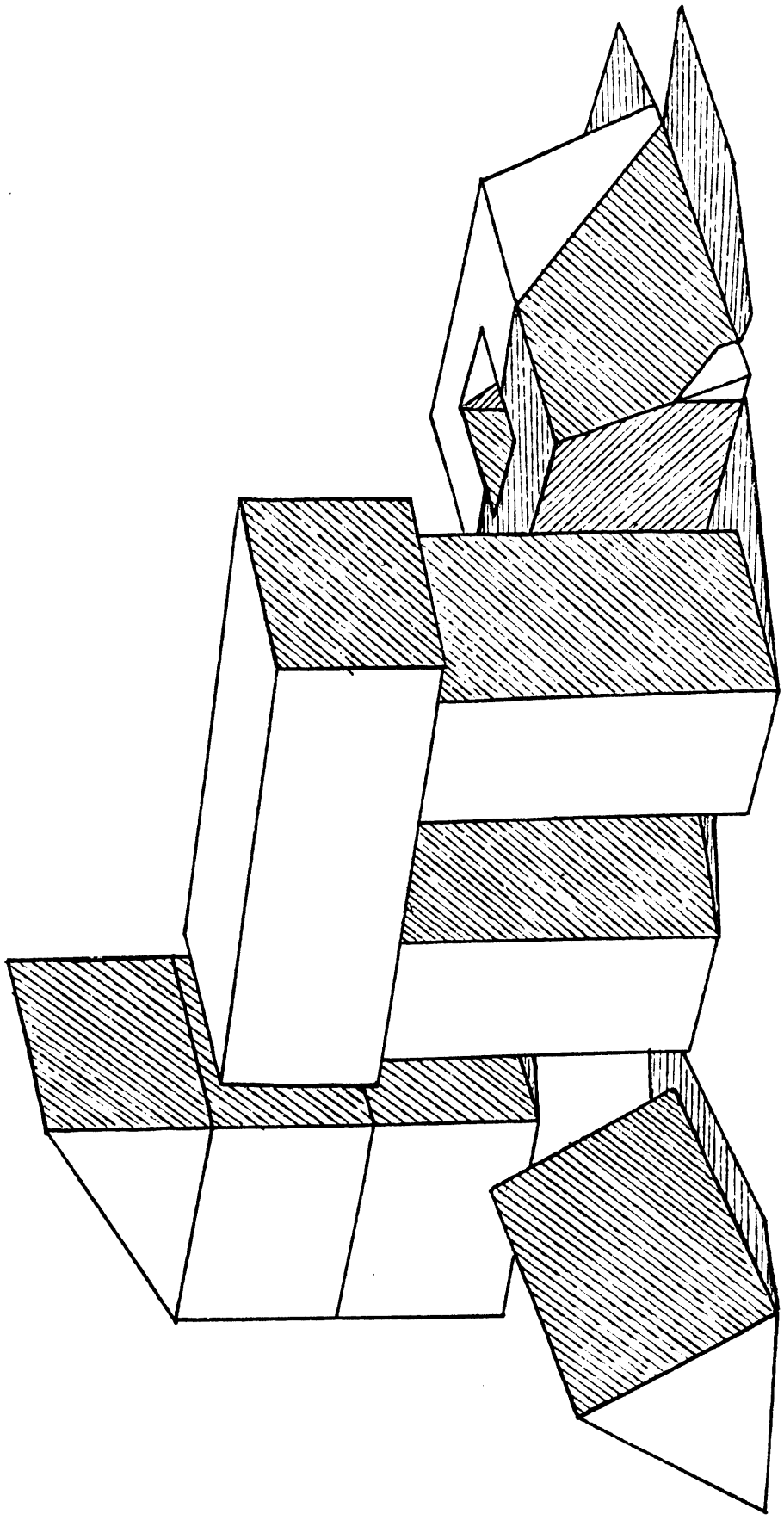


FIGURE 1.1

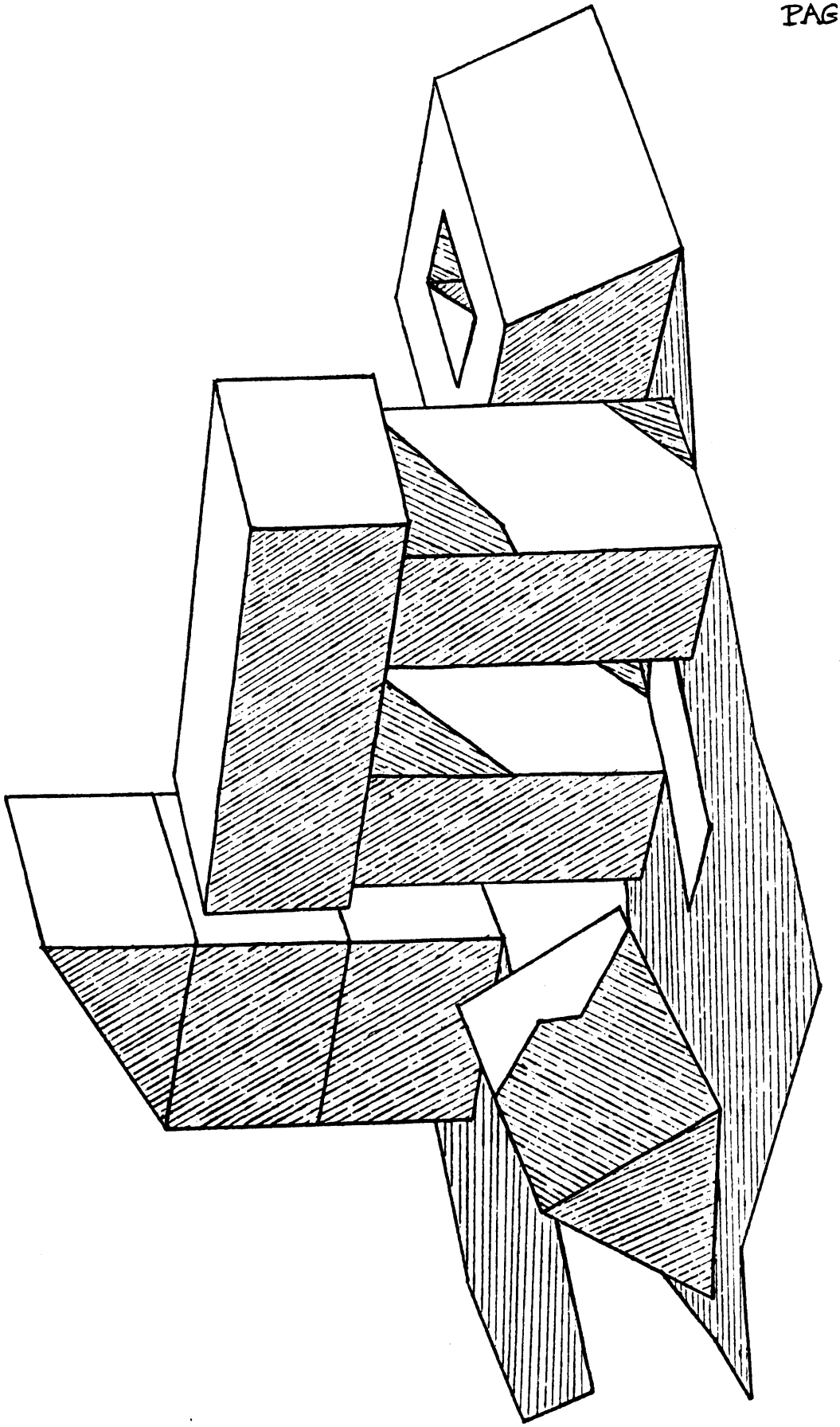


FIGURE 1.2

and use rote memory to assign shape to those 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 shows that we do not need texture or other fine details to ascertain shape, though we may of course use texture gradients and other details to define certain edges.

I undertook this research with the belief that it is possible to discover rules with which a program can obtain a three-dimensional model of a scene, given only a reasonably good line drawing of a scene. Such a program might have applications both in practical situations and in developing better theories of human vision. Introspectively, I do not feel that there is a great difference between seeing "reality" and seeing line drawings.

Moreover, there are considerable difficulties both in processing stereo images (such as the problem of deciding which points on each retina correspond to the same scene point; see Guzman 1968, Lerman 1970) and in building a system incorporating hand-eye coordination which could be used to help explore and disambiguate a scene (Gaschnig 1971). It

seems to me that while the use of range finders, multiple light sources to help eliminate shadows (Shirai 1971), 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 human perception vis-a-vis 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 demonstrated a number of capable vision programs which require only fixed, monocular line drawings for their operation.

In this thesis I describe a working collection of computer programs which reconstruct three-dimensional descriptions from line drawings which are obtained from scenes composed of plane-faced objects under various lighting conditions. In this description the system identifies shadow lines and regions, groups regions which belong to the same object, and notices such relations as contact or lack of contact between the objects, support and in-front-of/behind relations between the objects as well as information about the spacial orientation of various regions, all using the description it has generated.

1.1 DESCRIPTIONS

The overall goal of the system isto provide a precise description of a plausible scene which could give 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 must be capable of describing such objects. The language I have used is an expansion of the labeling system developed by Huffman (Huffman 1971) in the United States and Clowes (Clowes 1971) in Great Britain.

The language employs 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 illumination 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 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 based on this labeling.

1.2 JUNCTION LABELS

Much of the program's power is based on access to lists of possible line label assignments for each type ofjunction 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. Thus each "word" represents a physically possible interpretation for a given junction. 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 made up of 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" need not be 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 can be used to generate the complete dictionary from the original lists. Depending on the amount of computer memory available, it may either be desirable to store the complete lists as compiled knowledge or to generate the lists when they are needed. In my current program the lists are for the most part precompiled.

The composition of the dictionary isinteresting 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 (including region orientation and illumination values).

1.3 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 consistent with a particular junction. The first piece of information isalready 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. Figure 1.3 shows 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).

The program can also use local region brightness and line segment direction to preclude the assignment of certain labels to lines. For example, if it knows that one region is brighter than an adjacent region, then the line which separates the regions can be labeled as a shadowregion 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

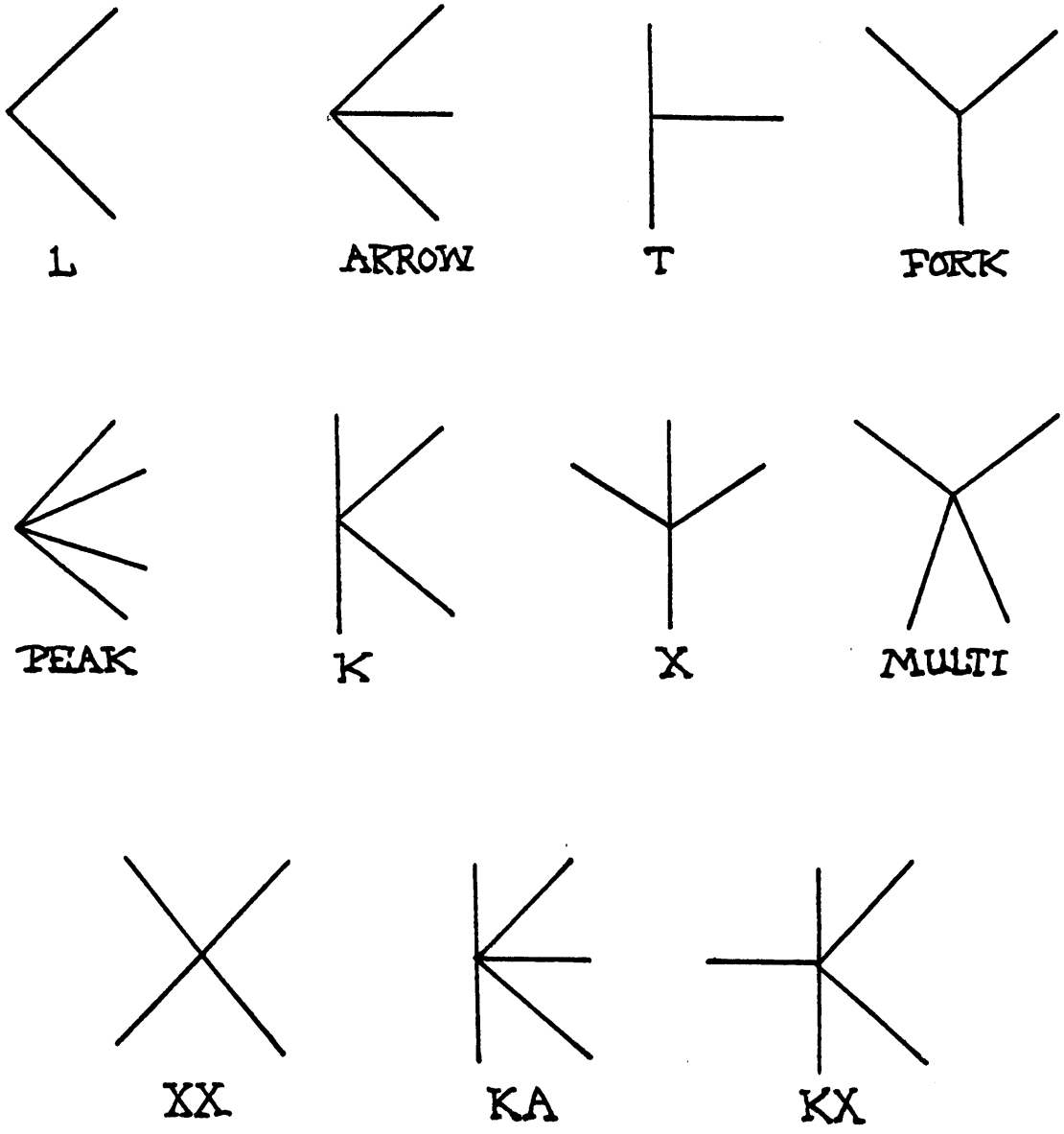


FIGURE 1.3

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.

1.4 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 junction labels "match" if and only if the line segment which joins the junctions gets the same interpretation from both of the junctions at its ends.

Of course, each interpretation (line label) is really a shorthand code for a number of properties of the line and its adjoining regions. If the program 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 it chooses a label to describe a particular junction, it constrains all the junctions which surround the regions touching 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 a program can find which lines represent obscuring edges, it cannot know where to construct hidden features, but if it needs the hidden features to label the lines, itmay not be able to decide which lines represent obscuring edges. As it turns 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.

1.5 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 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 the filtering program computes the possible labels for each line, using the fact that a line label is possible if and only if there is at least one junction label at each end of the line which contains the line label. Thus, the list of possible labels for a line segment 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

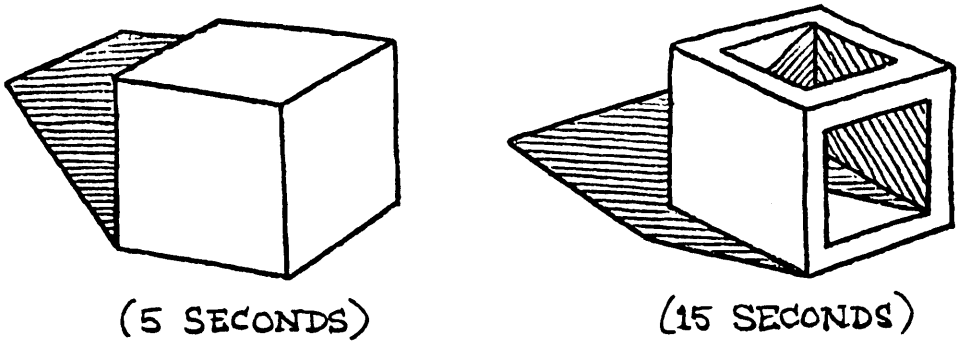
from consideration. The filtering program uses a network iteration scheme to systematically remove all the interpretations which are precluded by the elimination of labels at a particular junction.

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.

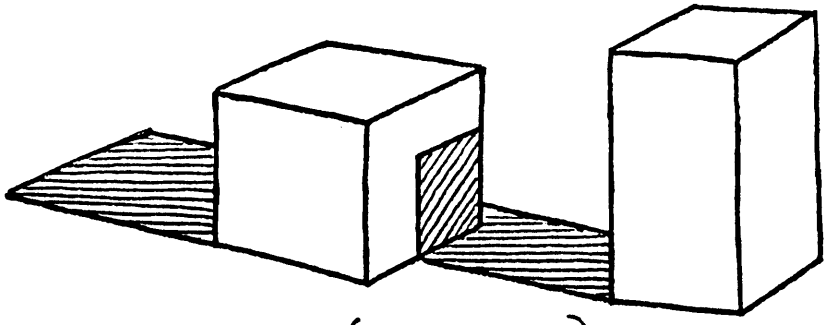
Whether or not more processing is required, the filtering system provides a computationally cheap method for acquiring a great deal of information. For example, in most scenes a large percentage of the line segments are unambiguously labeled, and more complicated processing can be directed to the areas which remain ambiguous. As another example, if I 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, the program may be able to get this information even though some ambiguities remain, since the ambiguity may only involve region illumination type or region orientation.

Figure 1.4 shows some of the scenes which the program is able to handle. The segments which remain ambiguous after its operation are marked with stars, and the approximate amount of time the program requires to label each scene is marked below it. The computer is a PDP-10, and the program is written partially in MICRO-PLANNER (Sussman et al 1971) and partially in compiled LISP.

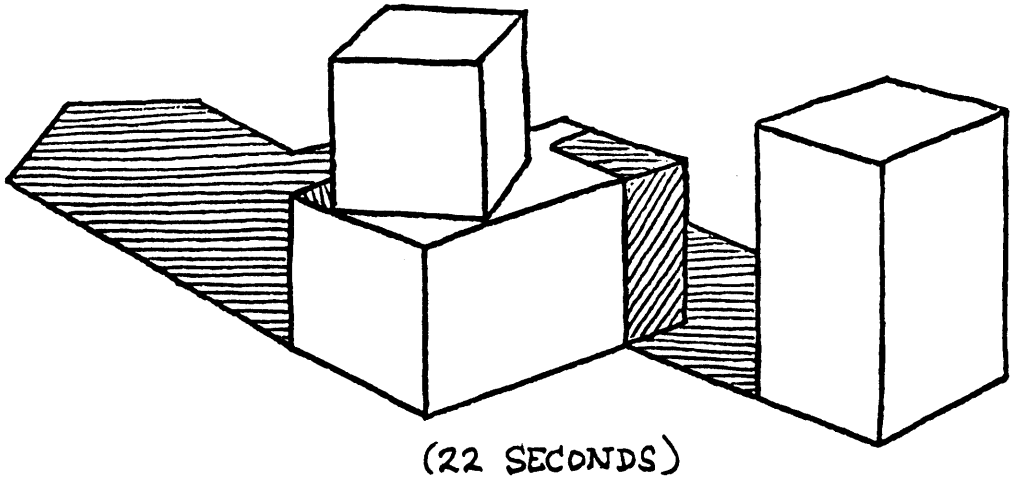


(5 SECONDS)

(15 SECONDS)

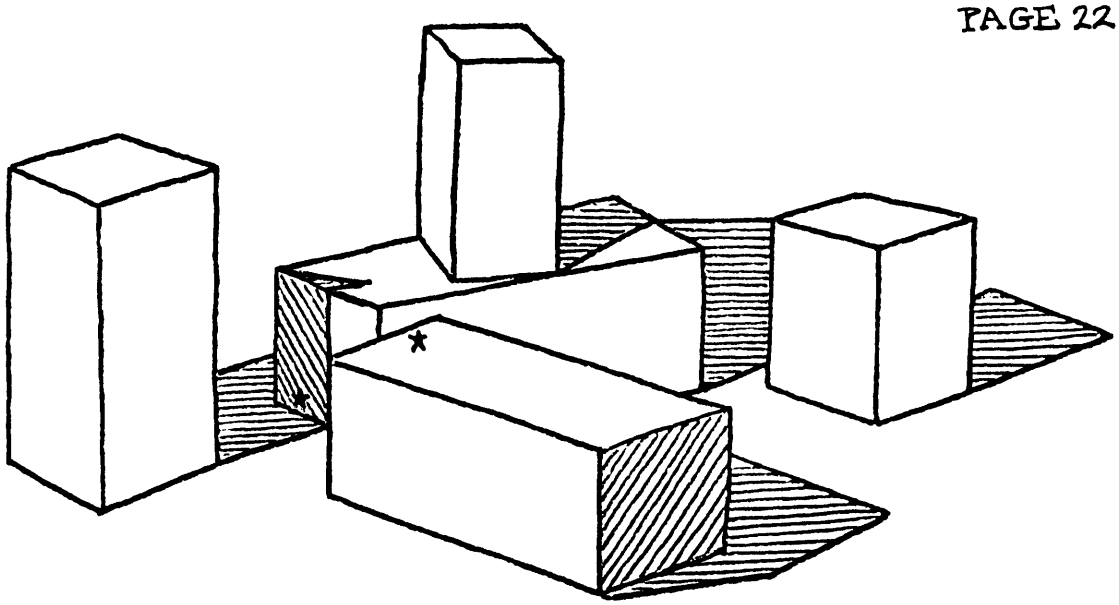


(15 SECONDS)

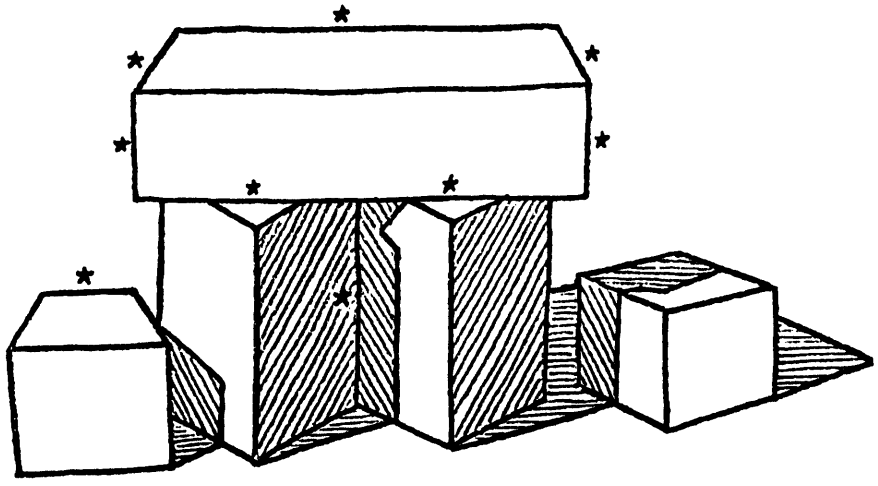


(22 SECONDS)

FIGURE 1.9



(39 SECONDS)



(48 SECONDS)

FIGURE 1.4

1.6 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, some junctions which have missing lines, 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 uses many different types of information to find the possible interpretations 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 preserves

these various possibilities. This tolerance for 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 by the most likely interpretation. This may simplify some problems, but sophisticated programs are needed 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.

1.7 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 discuss in Chapter 9 how my system helps in understanding the work of Guzman (Guzman 1968), Rattner (Rattner 1970), Huffman (Huffman 1971), Clowes (Clowes 1971), and Orban (Orban 1970); and to explain portions of the work of Winston (Winston 1970) and Finin (Finin 1971a, 1971b). For example, I show how

various concepts such as support 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.

1.8 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 abstraction 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. 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 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 so independent of particular objects. Back-up is not necessary in general; the system works for picture fragments as well as for entire scenes; the processing time required is proportional to the number of line segments and not an

exponential function of the number; all these facts 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. (For a quite different approach to the handling of curved objects, see Horn 1970.) I also believe that it will be possible to handle somewhat more general scenes (for instance scenes containing furniture, tools and household articles) by approximating the objects in them by simplified "envelopes" which preserve the gross form of the objects yet which can be described in terms like those I have used. In my estimation such processing cannot be done successfully until the problem of reconstructing the invisible portions of the scene is solved. 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. The ability to label a line drawing in the manner I describe greatly simplifies the specification and hopefully will simplify the solution of these problems. Chapter 8 deals with natural extensions of my program which I believe will

lead toward the eventual solution of these problems.

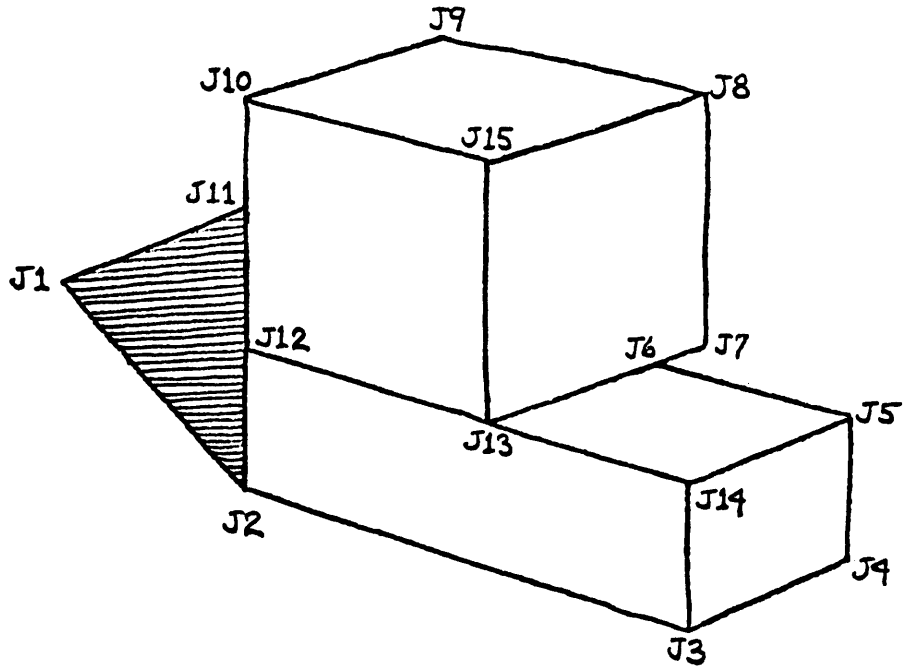
2.0 QUICK SYNOPSIS

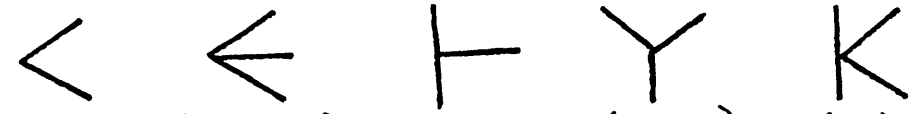
This chapter provides a quick look at some of the technical aspects of my work. All topics covered here are treated either in greater detail or from a different perspective in later chapters. For a hurried reader this chapter provides a map to the rest of the paper, and enough background to understand a later chapter without reading all the intervening ones.

2.1 THE PROBLEM

In what follows I frequently make a distinction 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.

Our first subproblem is to develop a language that allows us to relate these two worlds. I have done this by assigning names called labels to lines in the line drawing, after the manner of Huffman (Huffman 1971) and Clowes (Clowes 1971). Thus, for example, in figure 2.1 line segment J1-J2





(L)	(ARROW)	(T)	(FORK)	(K)
J1	J2	J6	J14	J13
J4	J3	J11	J15	
J7	J5	J12		
J9	J8			
	J10			

FIGURE 2.1

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. Thus, these terms are attached to parts of the drawing, but they designate the kinds of things found in the three-dimensional scene.

When we look at a line drawing of this sort, we usually can easily understand what the line drawing represents. In terms of a labeling scheme 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 it is impossible to 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 such labeling assignments?

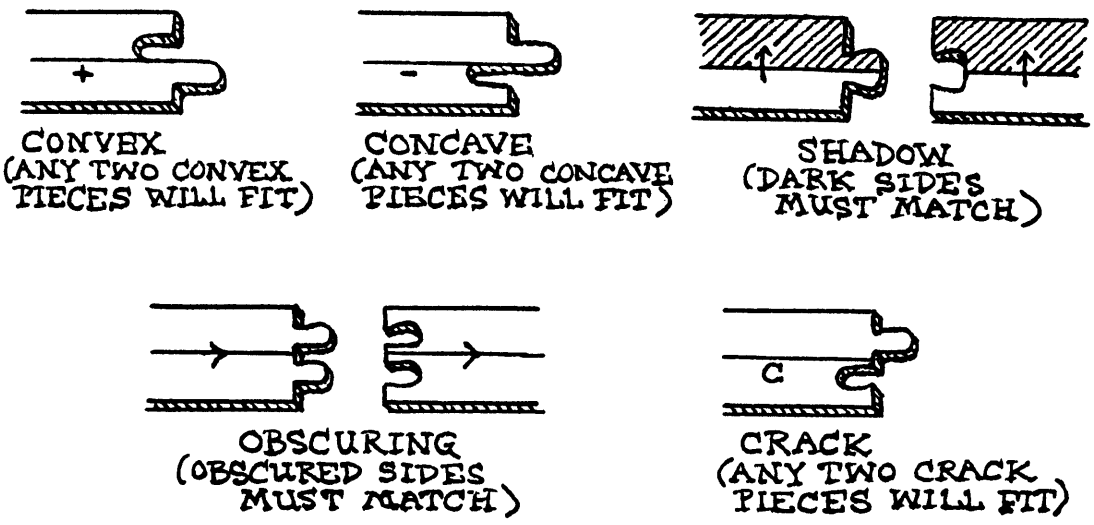
Huffman and Clowes provided a partial answer in their papers. They pointed out that each type of junction can only be labeled in a few ways, and that if we can say with certainty what the label of one particular line is, we can greatly constrain all other lines which intersect that line segment at its 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.

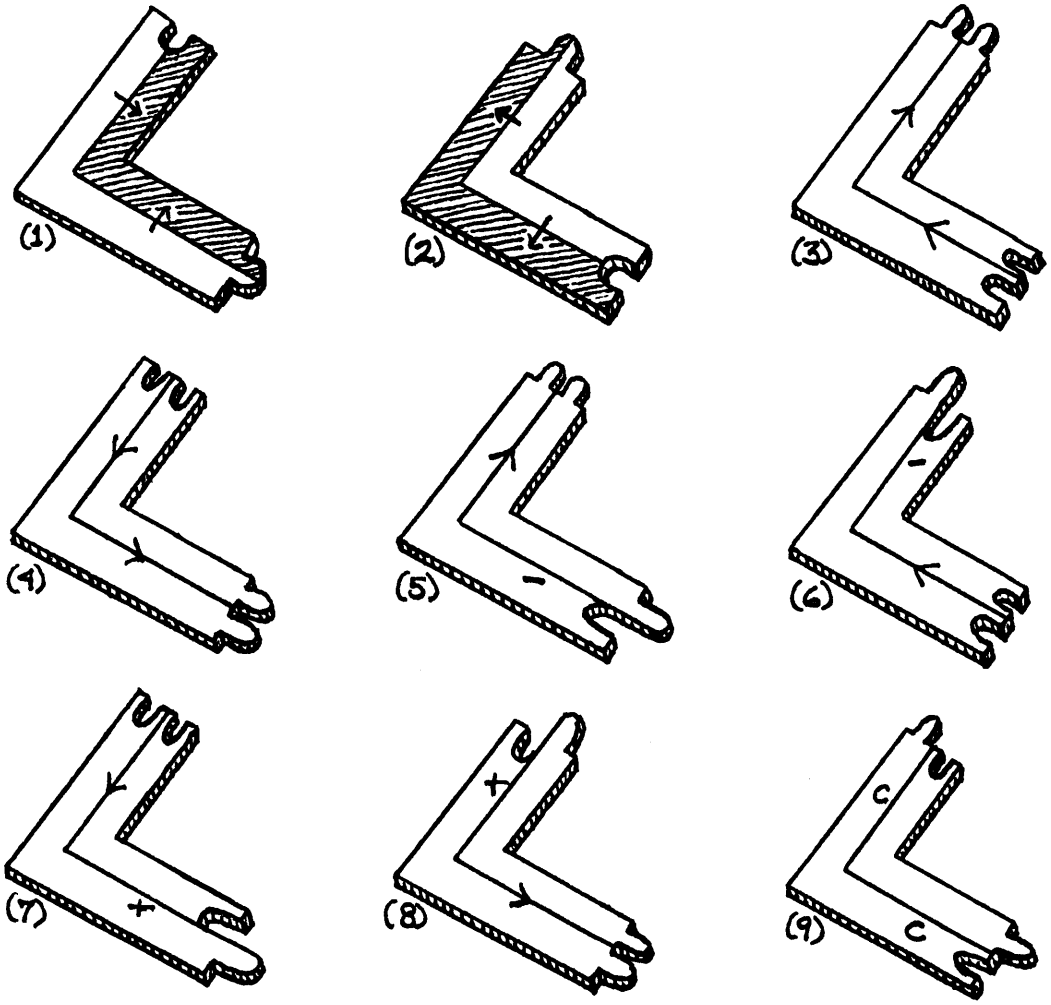
Moreover, shadows are directional, i.e. in order to specify a shadow edge, it must not only be labeled "shadow" but must also be marked to indicate which side of the edge is shadowed and which side is illuminated. Therefore, not only the type of edge but the nature of the regions on each side can be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

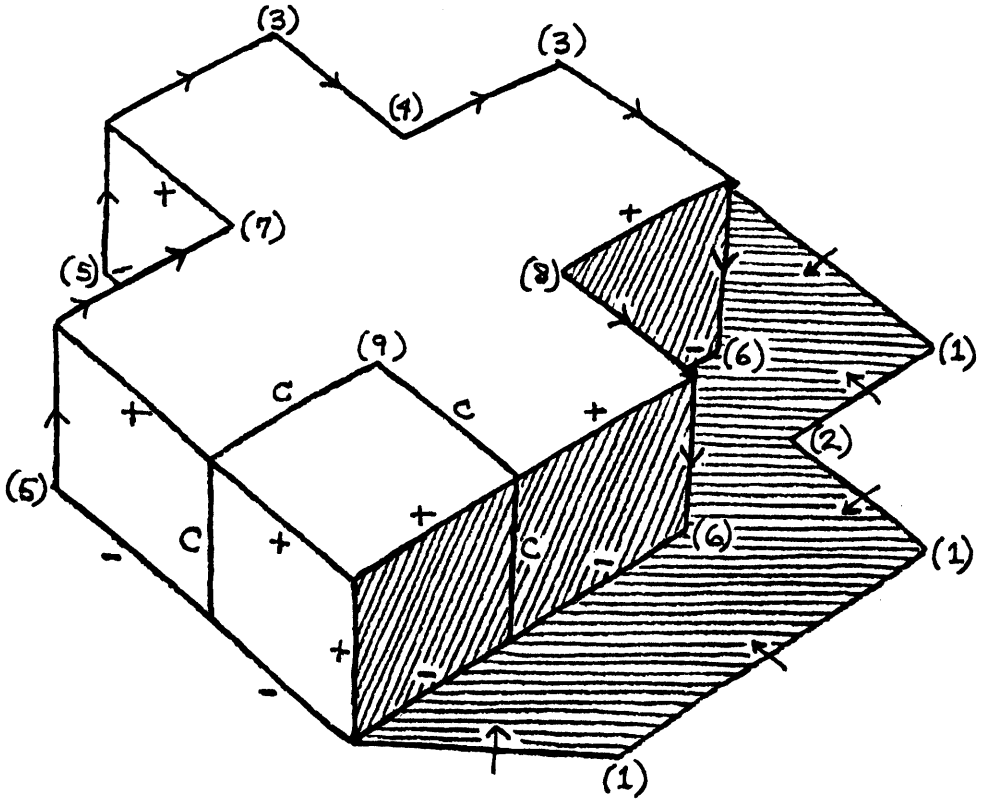
These facts can be illustrated in a jigsaw puzzle analogy, shown in figure 2.2. Given the five different edge types I have discussed so far, there are seven different ways to label any line segment. This implies that if all line labels could be assigned independently there would be $7^2 = 49$ different 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. Table 2.1 summarizes the ways in which junctions can be assigned labelings from this set. In figure 2.3, I show all the possible labelings for each junction type, limiting myself to vertices which are formed by no more than three planes (trihedral vertices) and to junctions of five or fewer lines. In Chapter 3 I explain how to obtain the

FIGURE 2.2



ALL 1 LABELINGS:





(THE NUMBERS REFER BACK TO THE 1, JUNCTION LABELINGS ON THE PRECEEDING PAGE)

FIGURE 2.2

JUNCTION TYPE	NUMBER OF POSSIBILITIES, LABELING BRANCHES INDEPENDENTLY	ACTUAL NUMBER OF LABELINGS	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
XX	2401	5	0.2
K	2401	12	0.5
MULTI	2401	24	1.0
KA	16807	8	0.05
KX	16807	12	0.07

TABLE 2.1

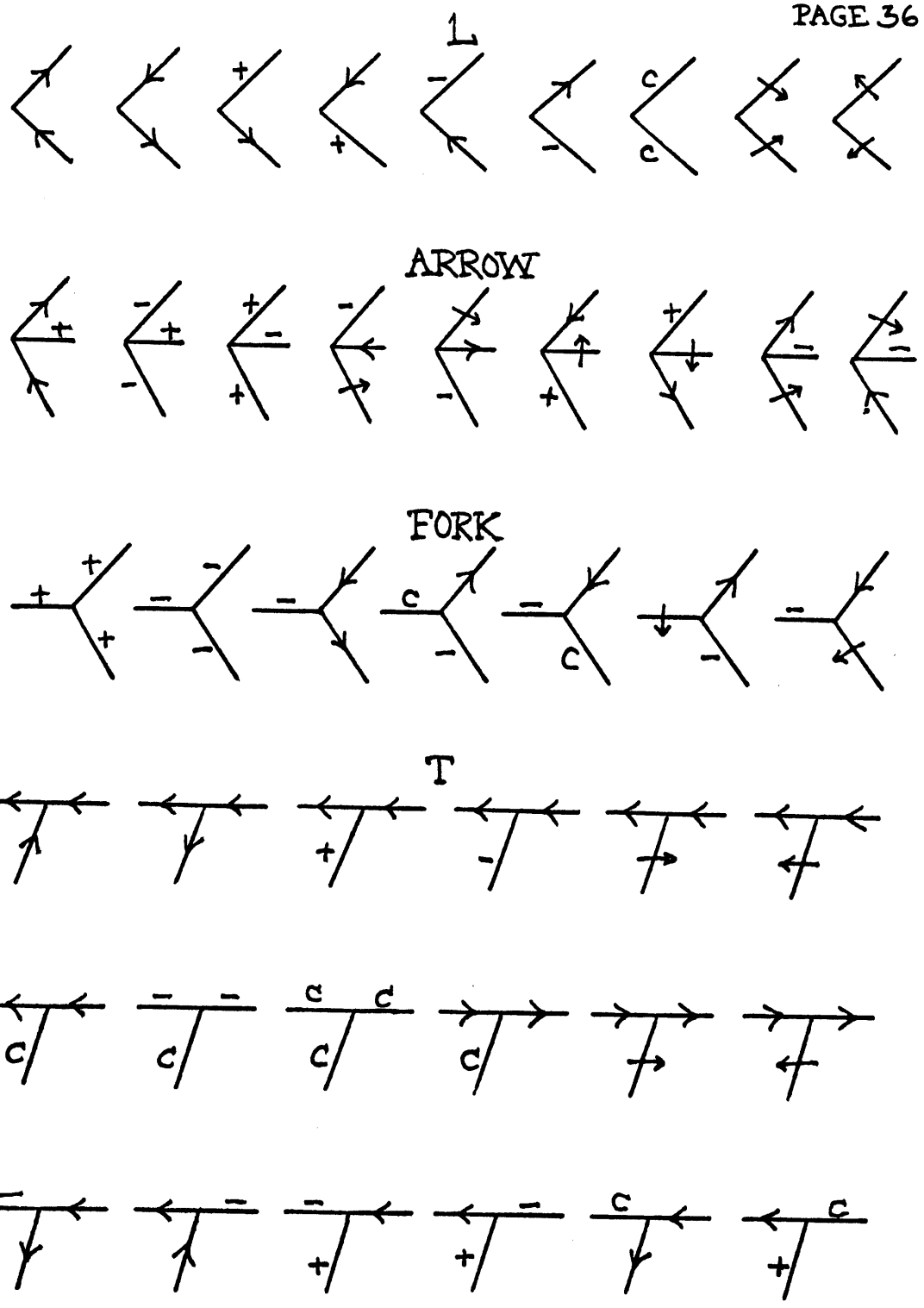
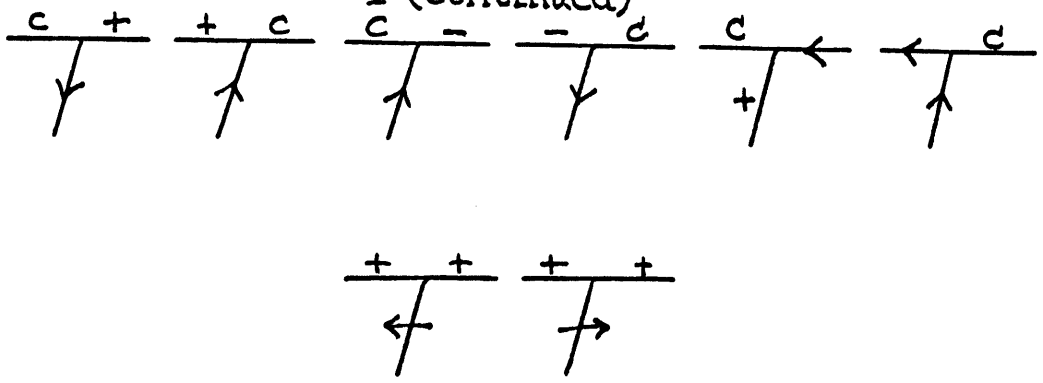
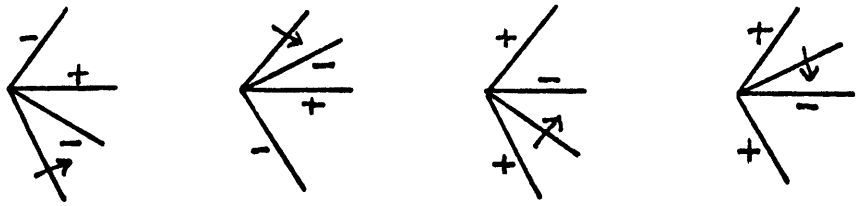


FIGURE 2.3

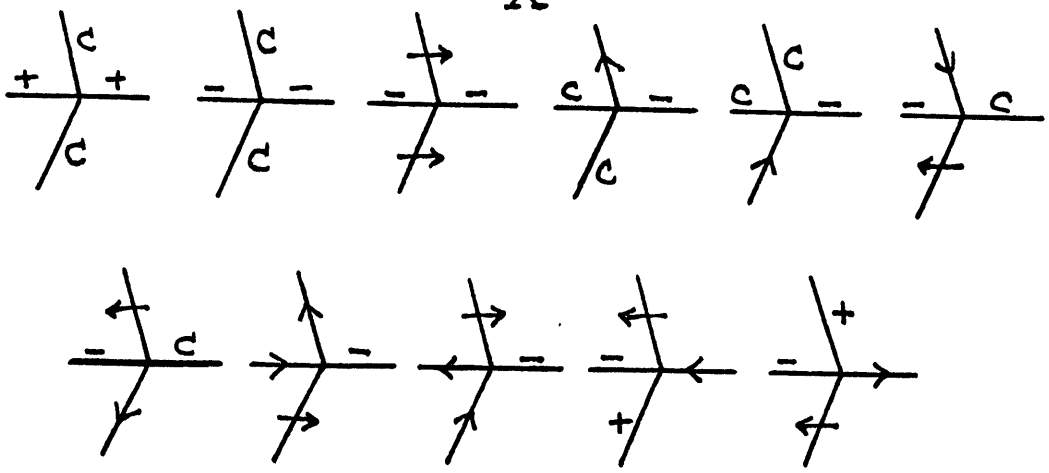
T (continued)



PEAK



X



XX

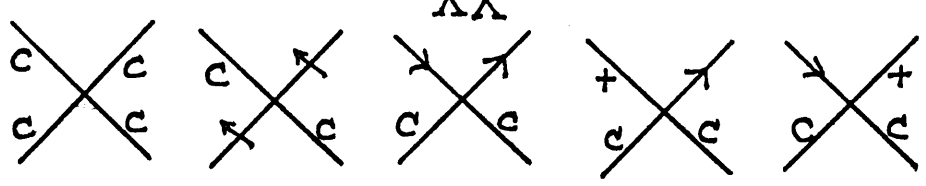


FIGURE 2.3

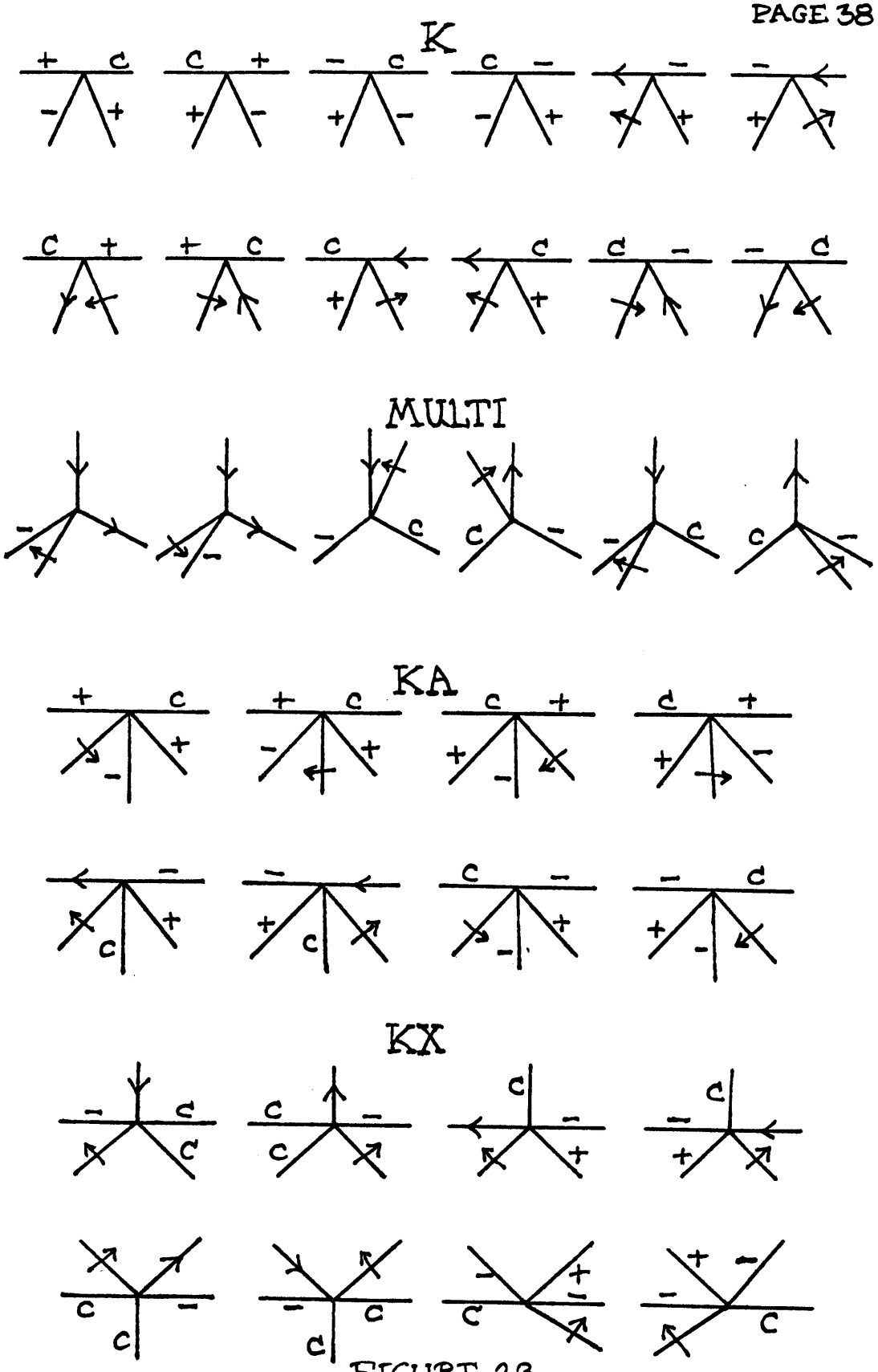


FIGURE 2.3

junctions in figure 2.3; I do not expect that it should be obvious to you how one could obtain these junctions. In general, for clarity, I have tried to use the word labeling to refer to the simultaneous assignment of a number of line labels. Labels thus refer to line interpretations, and labelings refer to junction or scene interpretations.

2.2 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:

4A. 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.

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

4C. If there are no more pieces in the "untried labels" bin, then

C2. 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

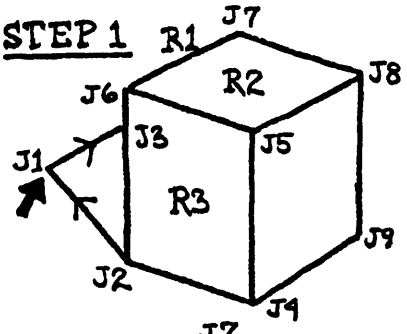
ii. Transfer the piece in space Nc-1 into its "tried labels" bin, and

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

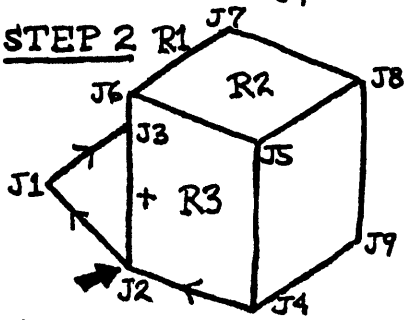
To see how this procedure works in practice, see figure 2.4. For this example assume that the pieces are piled so that the order in which they are tried is the same as the order in which the pieces are listed in figure 2.3. The example is carried out only as far as the first labeling obtained by the procedure. There is, of course, at least one other labeling, namely the one we could assign by inspection. The "false" labeling found first could be eliminated in this case by a program if it knew that R3 is brighter than R1 or that R2 is brighter than R1. It could then use heuristics which only allow it 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 help.

Clearly this procedure leaves many unsolved problems. In general there will be a number of possible labelings from which a program must still choose one. What rules can it use to make the choice? Even after choosing a labeling, in order

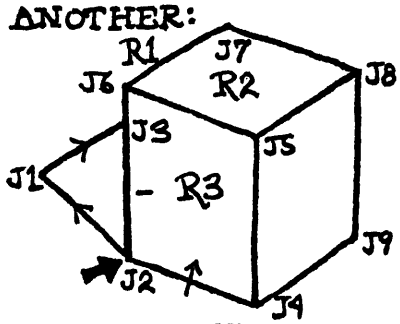
STEP 1



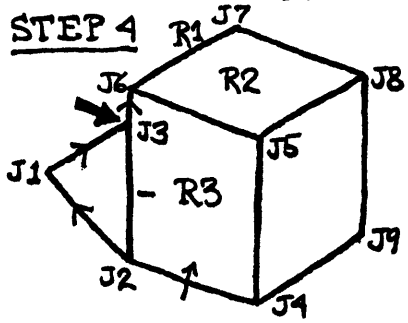
STEP 2



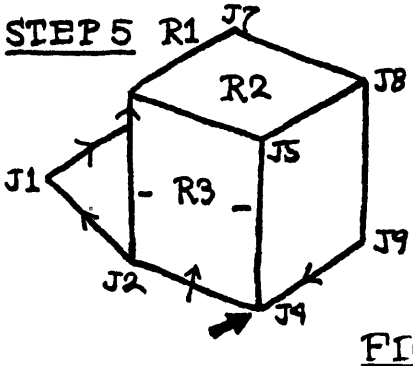
STEP 3 IMPOSSIBLE TO LABEL J3; ∴ REMOVE LABEL FROM J2 & TRY ANOTHER:



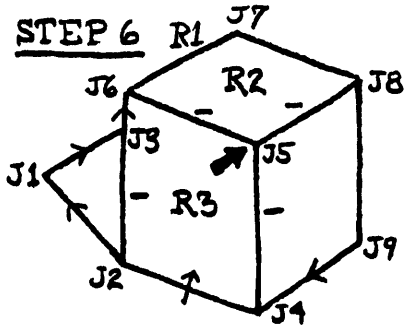
STEP 4



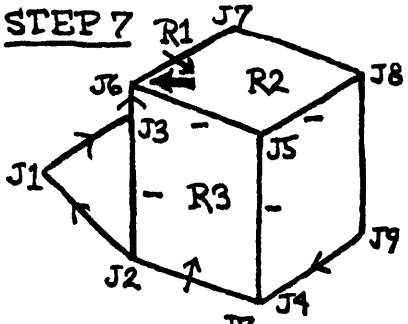
STEP 5



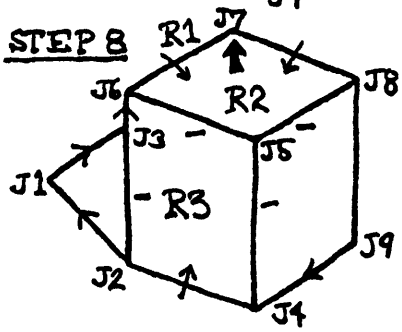
STEP 6



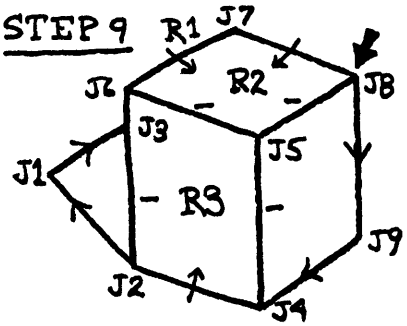
STEP 7



STEP 8



STEP 9



STEP 10

J9 HAS A LEGAL LABEL SO THE RESULT IS A POSSIBLE LABELING. NOTICE THAT IT IS NOT IN FACT A VALID INTERPRETATION; THE INFORMATION USED AND/OR THE DESCRIPTION IS INADEQUATE.

FIGURE 24

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.) a program must use rules of some sort to deduce the answers from the information it has.

I will argue that what is needed to find a single reasonable interpretation of a line drawing 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.

2.3 BETTER EDGE DESCRIPTION

So far I have classified 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. Figure 2.5 shows the results of appending a label analogous to the "obscuring

OLD LABELING:

NEW LABELING:

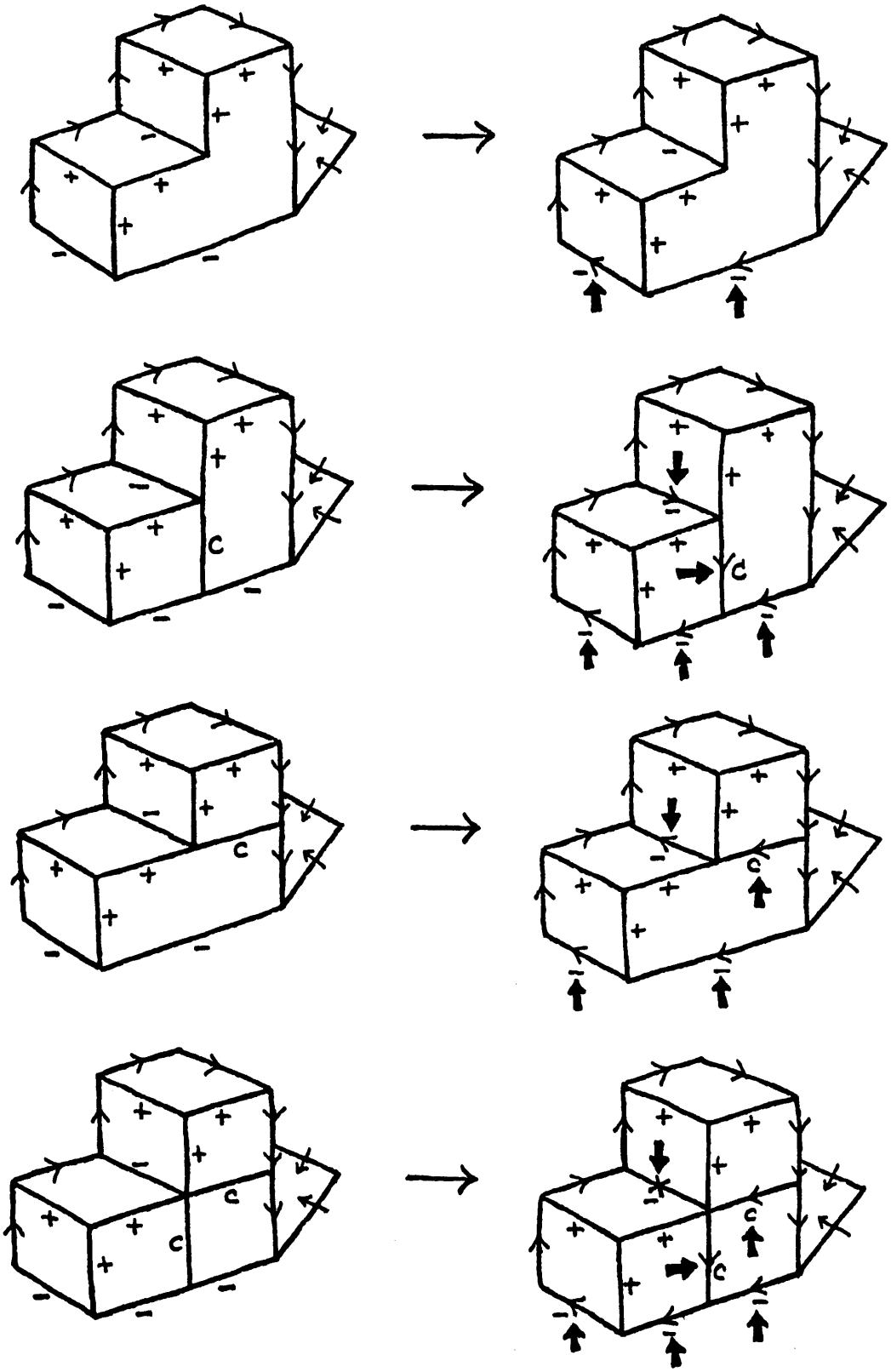


FIGURE 2.5

INTERPRETATION:

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

R1 —<
R2 — A SEPARABLE TWO-OBJECT CONCAVE EDGE; IF [OB(R1)] IS ABOVE [OB(R2)] THEN [OB(R2)] SUPPORTS [OB(R1)].

R1 —>
R2 — SAME AS ABOVE; IF R1 IS ABOVE R2, THEN [OB(R2)] OBSCURES [OB(R1)] OR [OB(R1)] SUPPORTS [OB(R2)].

R1 —X
R2 — A SEPARABLE THREE-OBJECT CONCAVE EDGE; NEITHER [OB(R1)] NOR [OB(R2)] CAN SUPPORT THE OTHER.

R1 C>
R2 — A CRACK EDGE; [OB(R2)] IS IN FRONT OF [OB(R1)] IF R1 IS ABOVE R2.

R1 C<
R2 — A CRACK EDGE; [OB(R2)] SUPPORTS [OB(R1)] IF R1 IS ABOVE R2.

SEPARATIONS:

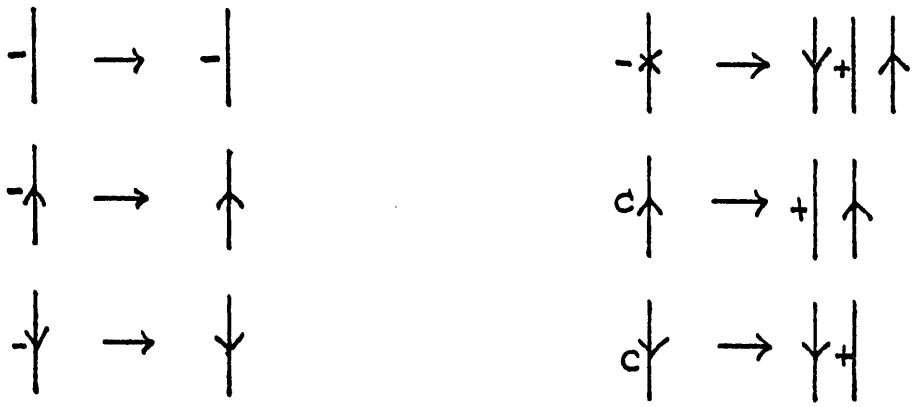


FIGURE 2.5

edge" mark to crack and concave edges. This approach is similar to one first proposed by Freuder (Freuder 1971a).

Each region can also be labeled 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.

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. Put another way, whenever a line can be assigned a single label which includes this lighting information, then a program has powerful constraints for the junctions which can appear around either of the regions which bound this line.

Figure 2.6 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.

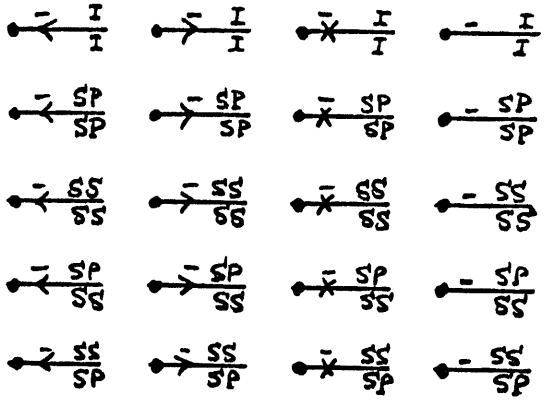
These tables can be used to expand the set of allowable junction labels; the new set of labels can have a number of entries which have the same edge geometries but which have different region illumination values. It is very easy to write a program to expand the set of labelings; the principles of its 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 2.6.

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 2.7. Thus whenever a scene has shadows and whenever a program can find a shadow causing junction in such a scene, it can greatly constrain all the lines and regions which make up this junction. In figure 2.7

CONCAVE (ANY)

1 | 2
−

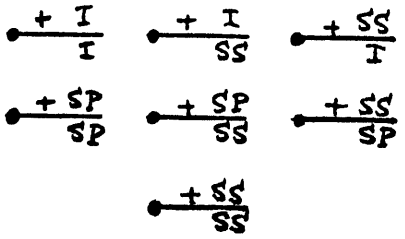
1 \ 2	I	SP	SS
I	YES	No	No
SP	No	YES	YES
SS	No	YES	YES



CONVEX

1 | 2
+

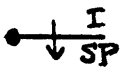
1 \ 2	I	SP	SS
I	YES	No	YES
SP	No	YES	YES
SS	YES	YES	YES



SHADOW C.C.*

1 | 2
→

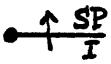
1 \ 2	I	SP	SS
I	No	YES	No
SP	No	No	No
SS	No	No	No



SHADOW CL.*

1 | 2
←

1 \ 2	I	SP	SS
I	No	No	No
SP	YES	No	No
SS	No	No	No



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

FIGURE 2.6

CRACK (EITHER)

C

1 \ 2	I	SP	SS
I	YES	No	No
SP	No	YES	No
SS	No	No	YES

OCCLUDE (EITHER)

1 2

OR

1 2

1 \ 2	I	SP	SS
I	YES	YES	YES
SP	YES	YES	YES
SS	YES	YES	YES

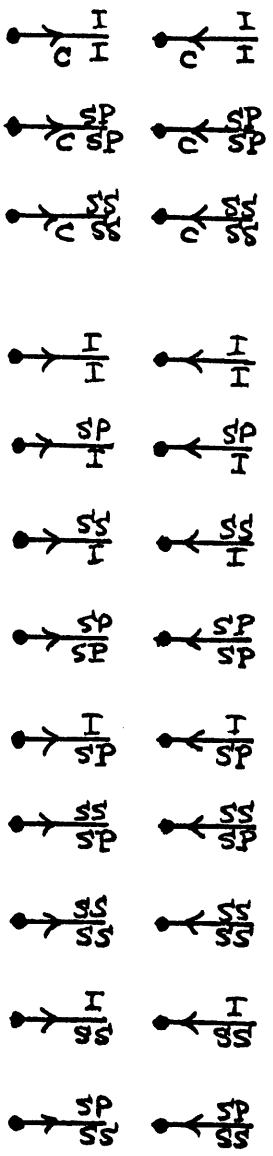
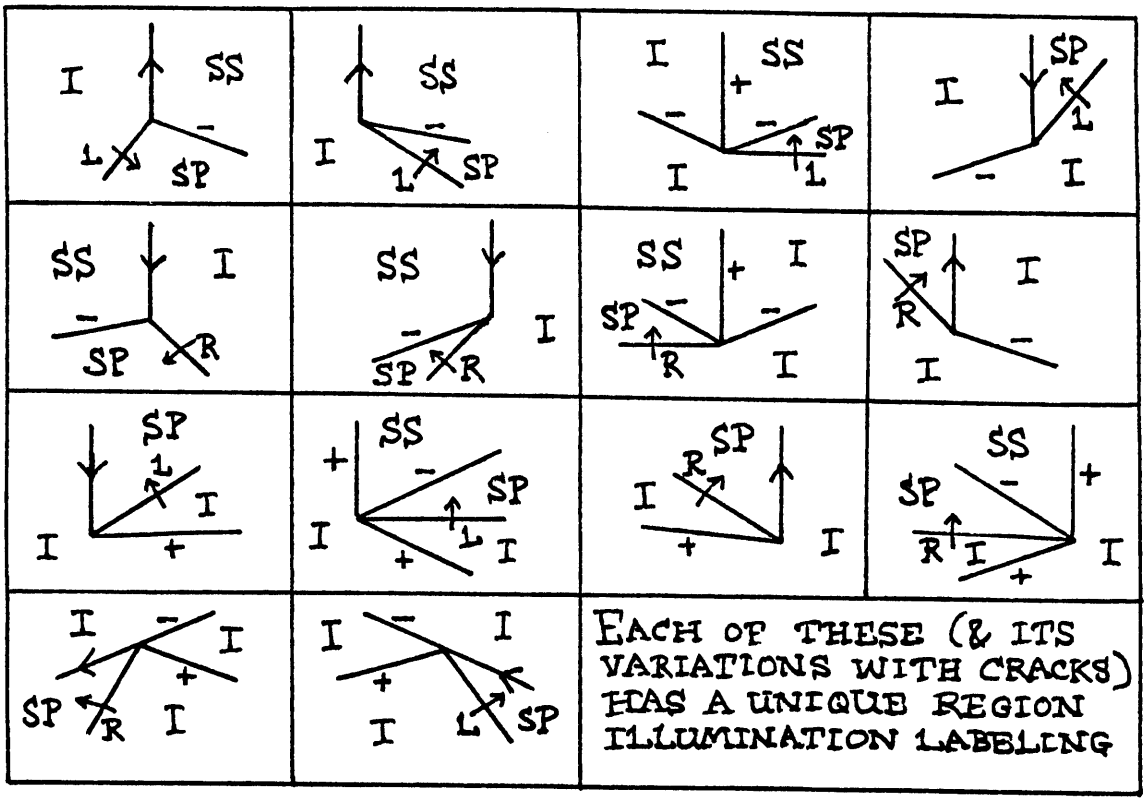
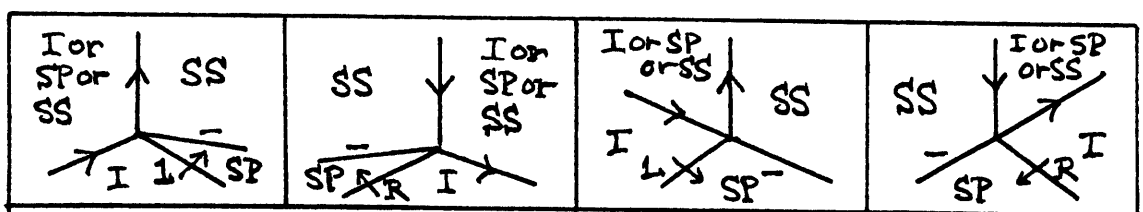


FIGURE 2.6



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



EACH OF THESE JUNCTIONS HAS THREE DIFFERENT REGION ILLUMINATION LABELINGS

FIGURE 2.7

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.

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, s obscuring edges and t convex edges will have to be replaced by $(20)^n(6)^m(3)^p(3)^q(9)^s(8)^t$ new junctions, or

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

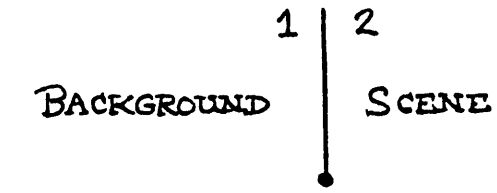
If (1) were true then the partition would be worthless, since no new information could be gained. If (2) were true, the situation would be greatly improved, since in a sense all the much more precise information was implicitly included in

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. The truth lies somewhere between the two extremes, but the fact that it is not at the extreme of (1) means that there is a net improvement. In Table 2.2 I compare the situation now to cases (1) and (2) above and also to the situation depicted in Table 2.1.

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 2.8. Because of this fact the number of junction labels which could be used to

JUNCTION TYPE	# OF POSSIBILITIES, EACH BRANCH LABELED AS INDEPENDENT	# IF OLD JUNCTIONS EXPANDED = $\sum_i (20^{n_i} \cdot 6^{m_i} \cdot 3^{P_i} \cdot 3^{8_i} \cdot 7^{t_i} \cdot 9^{5_i})$	OLD # OF JUNCTIONS	ACTUAL # OF NEW JUNCTIONS	OLD PERCENTAGE	NEW PERCENTAGE
		CASE(1)*	CASE(2)*		(%)	(%)
L	3249	702	9	92	18.4	2.8
ARROW	185,193	6885	9	86	2.6	0.05
FORK	185,193	22923	17	826	5.0	0.45
T	185,193	18903	26	623	7.6	0.34
PEAK	11,555,901	22680	4	10	0.2	$\sim 10^{-4}$
X	11,555,901	?	37	435	1.5	$\sim 3.8 \times 10^{-3}$
XX	11,555,901	?	5	128	0.2	$\sim 10^{-3}$
MULTI	11,555,901	?	24	160	1.0	$\sim 1.4 \times 10^{-3}$
K	11,555,901	?	12	213	0.5	$\sim 1.8 \times 10^{-3}$
KA	6.58×10^8	?	8	20	0.05	$\sim 3.0 \times 10^{-6}$
KX	6.58×10^8	?	12	76	0.07	$\sim 1.1 \times 10^{-5}$

TABLE 2.2 *SEE TEXT, PAGE 51



CAN ONLY BE LABELED IN
ONE OF THE FOLLOWING WAYS:

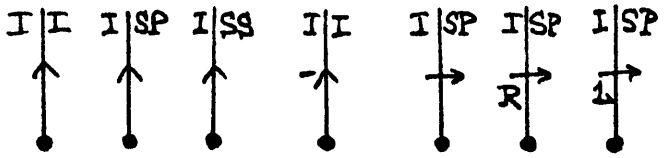


FIGURE 2.8

label junctions on the scene/background boundary can be greatly restricted.

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 = 18.4% to (7/57)(7/57) = 49/3149 = 1.6%. The results of these improvements will become obvious in the next section.

2.4 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 variation of this procedure which only requires one pass through the junctions. This procedure is similar to the one used to generate figure 2.9, 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.

It is not obvious that this should be the case. For example, if this filter procedure is applied to the simple line drawing shown in figure 2.4 using the old set of labels given in figure 2.3, it produces the results shown in figure 2.9. In this figure, each junction has labels attached which would not be part of any total labeling produced by a tree

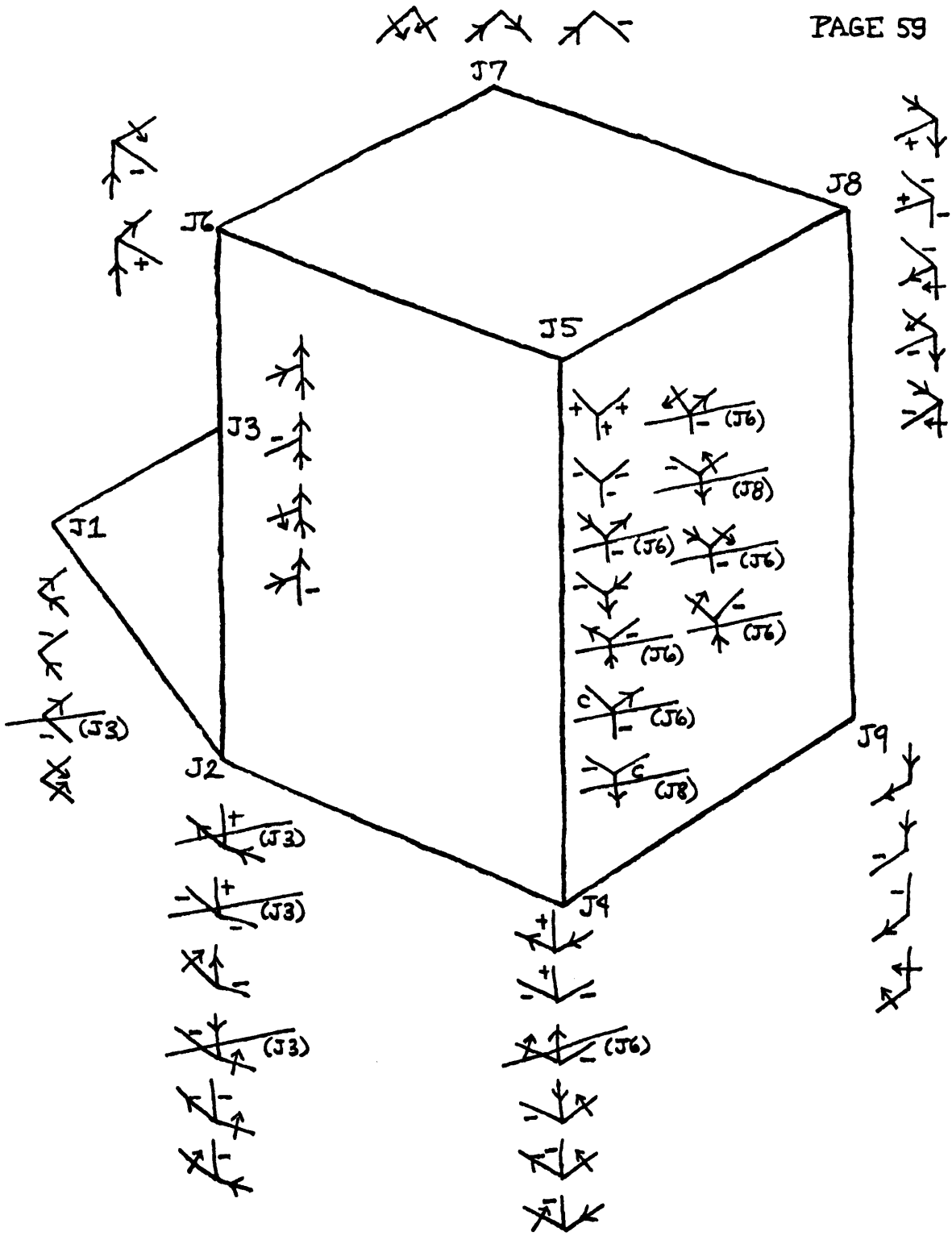


FIGURE 2.9

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 it is known 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 2.9.

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 local information in a line drawing so that a program can use a procedure which requires far less computation than does a tree search procedure. To see why this is so, notice that if the description the program uses is good enough, then many junctions must always be given the same unique label in each tree search solution; the filtering program needs to find such a label only once, while a tree search procedure must go through the process of finding the same solution on each pass through the tree.

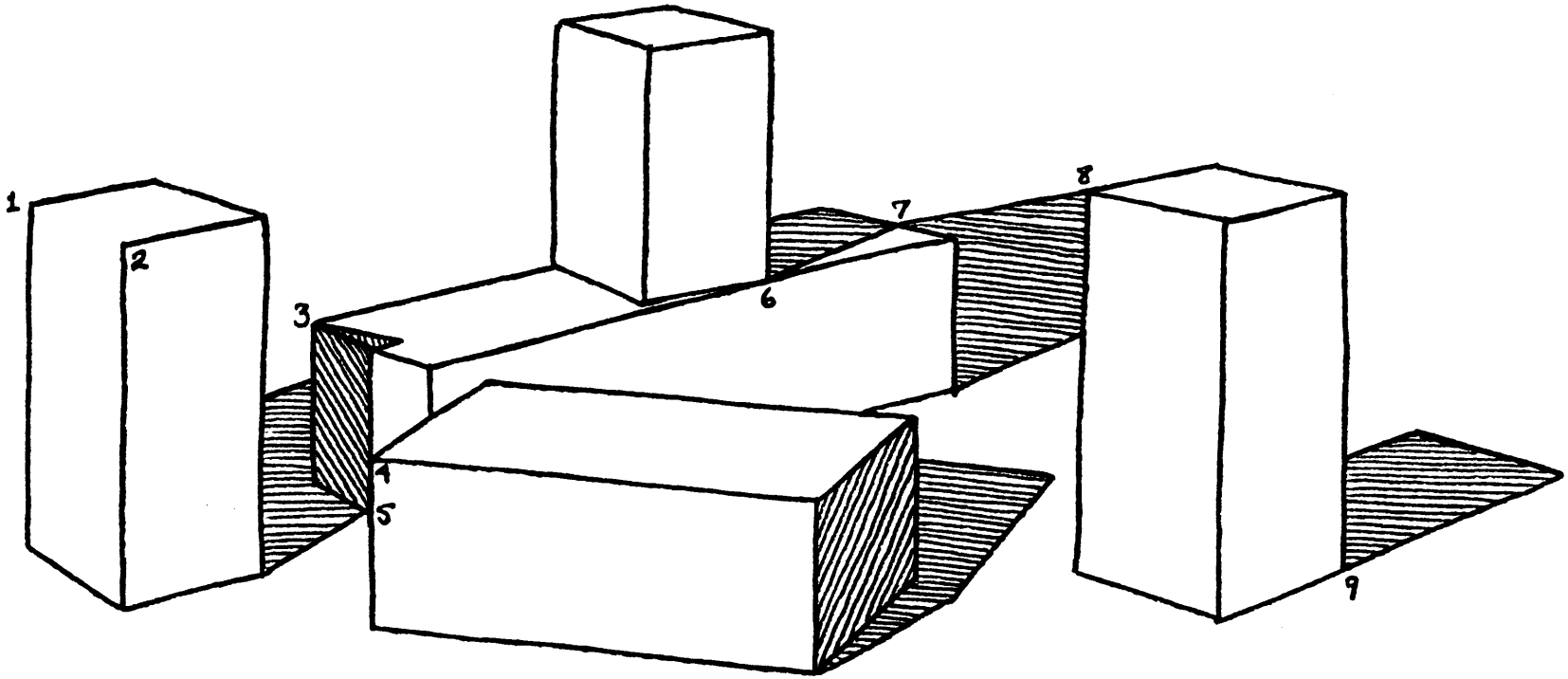
Quite remarkably, all these results are obtained using only the topology of line drawings plus knowledge about which region is the table and about the relative brightness of each region. No use is made (yet) of the direction of line segments (except that some directional information is used to classify 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.

2.5 HANDLING BAD DATA

So far I have treated this subject as though the program would always be given perfect data. In fact there are many types of errors and degeneracies which occur frequently. Some of these 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 2.10 shows a scene containing instances of each type of problem.

The program handles these problem junctions by generating labels for them, just as it does for normal junctions. 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



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 VERTEX

7,8 ACCIDENTAL ALIGNMENTS

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

FIGURE 2.10

2.5 HANDLING BAD DATA

So far I have treated this subject as though the program would always be given perfect data. In fact there are many types of errors and degeneracies which occur frequently. Some of these 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 2.10 shows a scene containing instances of each type of problem.

The program handles these problem junctions by generating labels for them, just as it does for normal junctions. 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

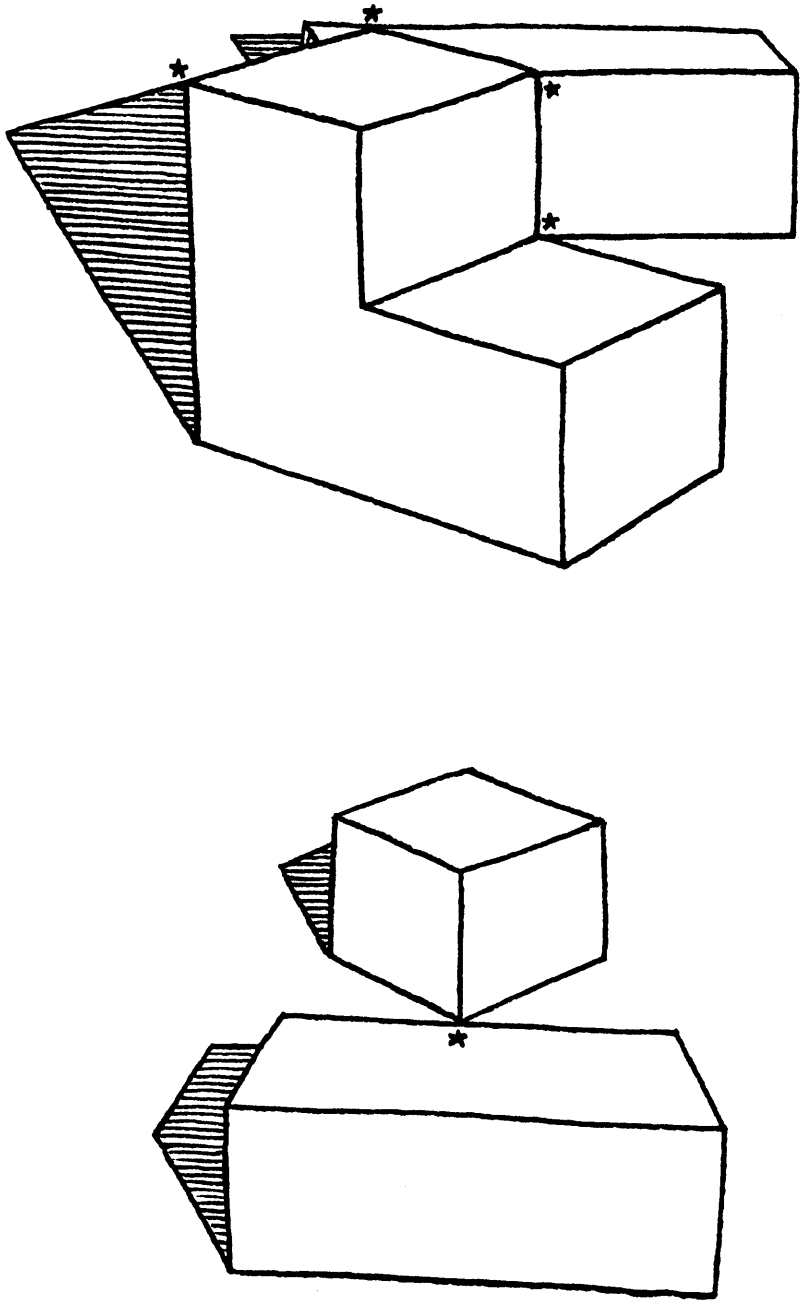


FIGURE 2.11

interpreting the scene as though the data were perfect and it would then not even get an indication that it should look for other interpretations.

2.6 ACCIDENTAL ALIGNMENT

Chapter 7 treats a number of different types of accidental alignment. Figure 2.11 shows three of the most common types which are included in the program's repertoire; 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 2.11a),

(2) cases where an edge which is between the eye and a vertex appears to intersect the vertex (see figure 2.11b), and

(3) cases where a shadow is projected so that it actually does intersect a vertex (see figure 2.11c).

2.7 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 much about them. There will have to be facilities in the program for filling in hidden surfaces and back faces of objects before missing lines can be treated satisfactorily.

In general the program will report that it isunable to label a scene 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 2.12 cannot be labeled at all unless the missing line junctions

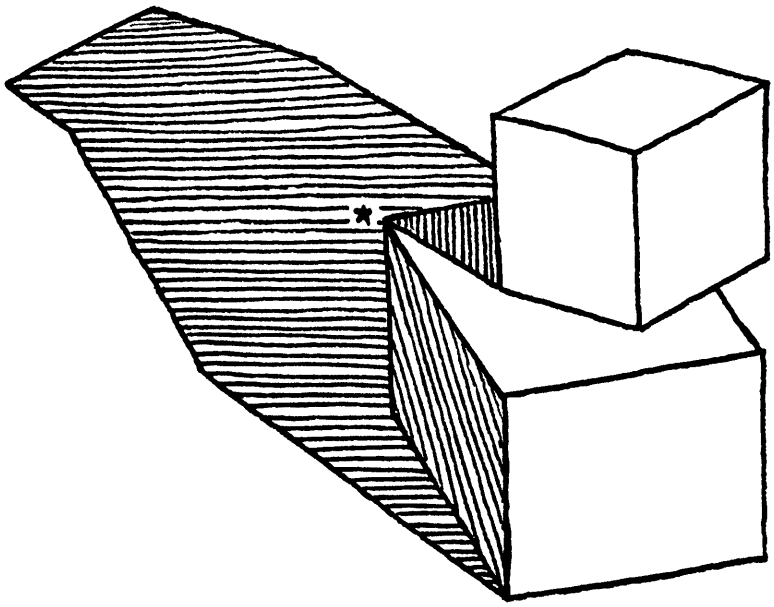


FIGURE 2.11

are included.

2.8 REGION ORIENTATIONS

Regions can be assigned labels which give quantized values for region orientations 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. It is impossible to do justice to the topic here, but region orientations are treated in considerable detail in Chapter 8.

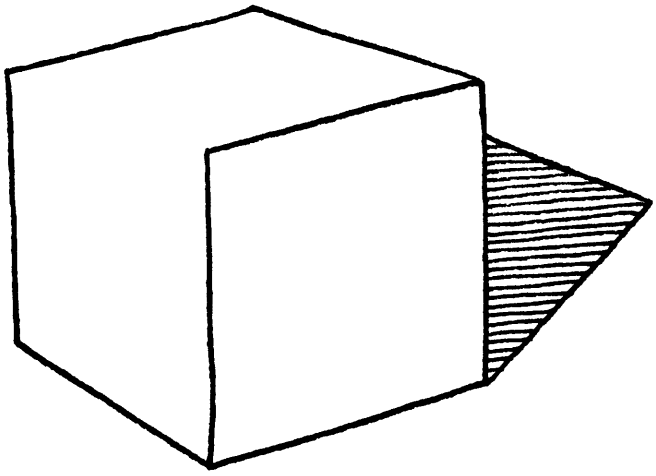


FIGURE 2.12

heuristics find a "plausible" interpretation if required. For example, one heuristic eliminates interpretations that involve concave objects in favor of ones that involve convex objects, and another prefers interpretations which have the smallest number of objects; this heuristic prefers a shadow interpretation for an ambiguous region to the interpretation of the region as a piece of an object.

In this chapter I show how to express the first type of knowledge, and give hints about some of the others. A large proportion of my energy and thought has gone into the choice of the set of possible line labels and the sets of possible junction labels. In this I have been guided by experiment with my program, since there are simply too many labels to hand simulate the program's reaction to a scene. The program, the set of edge labels, and the sets of junction labelings have each gone through an evolution involving several steps. At each step I noted the ambiguities of interpretation which remained, and then modified the system appropriately.

The changes have generally involved (1) the subdivision of one or more edge labels into several new labels embodying finer distinctions, and (2) the recomputation of the junction

3.0 TRIHEDRAL JUNCTION LABELS

The knowledge of this system is expressed in several distinct forms:

(1) A list of possible junction labels for each type of junction geometry includes the a priori knowledge about the possible three dimensional interpretations of a junction.

(2) Selection rules which use junction geometry, knowledge about which region is the table, and region brightness. These can easily be extended to use line segment directions to find the subset of the total list of possible junction labelings which could apply at a particular junction in a line drawing.

(3) A program to find the possible labelings; it knows how to systematically eliminate impossible combinations of labels in a line drawing and, as such, contains implicit knowledge about topology.

(4) Optional heuristics which can be invoked to select a single labeling from among those which remain after all the other knowledge in the program has been used. These

parts so that the types of trihedral vertex can be characterized by the octants of space around the vertex which are filled by solid material (Huffman 1971).

Dowson (Dowson 1971a) went a little further in discussing how one could write an algorithm to find all possible trihedral junctions and their labels (using the simple three-label model of Huffman and Clowes). In fact he never used his system to generate every class of junction geometry but was satisfied to show that it could generate the twelve labels which Huffman and Clowes originally used. These twelve labels represent four different ways of filling in the octants (where I have not counted ways of filling the octants which differ only by rotation as different).

Dowson's scheme is useful for visualizing how to generate the ten different ways of filling the octants which I use. Consider the general intersection of three planes as shown in figure 3.1. These planes divide space into octants, which can be uniquely identified by three-dimensional binary vectors ($\hat{x}$ $\hat{y}$ $\hat{z}$) where the x, y, and z directions are specified as shown. The vectors make it easy to describe the various geometries precisely. I can then generate all possible geometries and non-degenerate views by imagining

label lists to include these new distinctions. In each case I have been able to test the new scheme to make sure that it solves the old problems without creating any unexpected new ones. For example, the initial data base contained only junctions which (1) represented trihedral vertices (i.e. vertices caused by the intersection of exactly three planes at a point in space) and (2) which could be constructed using only convex objects. The present data base has been expanded to include all trihedral junctions and a number of other junctions caused by vertices where more than three planes meet.

Throughout this evolutionary process I have tried to systematically include in the lists every possibility under the stated assumptions. In this part of the system I have made only one type of judgement: if a junction can represent a vertex which is physically possible, include that junction in the data base.

3.1 EDGE GEOMETRY

The first problem is to find all possible trihedral vertices. Huffman observed that three intersecting planes, whether mutually orthogonal or not, divide space into eight

various octants to be filled in with solid material. There are junctions which correspond to having 1, 2, 3, 4, 5, 6, or 7 octants filled. Figure 3.2 shows the twenty possible geometries that result from filling various octants, and in Appendix 1 I have shown all the junction labelings (not including shadow variations) which can result from the geometries in figure 3.2A. The result of this process is 196 different junction labels. Figure 3.2B consists of the geometries which I have chosen not to use to generate junction labels. I have not included these geometries because each involves objects which touch only along one edge, and whose faces are nonetheless aligned, an extremely unlikely arrangement when compared to the other geometries. (In addition, some of the geometries are physically impossible unless one or more objects are cemented together along an edge or supported by invisible means.)

The four geometries recognized by Huffman, Clowes, and Dowson correspond to my numbers 1, 3, 5, and 7 in figure 3.2A.

In figure 3.3 I show how the 20 different labels with type 3 geometry can be generated. Basically this process involves taking a geometry from figure 3.2A, finding all the

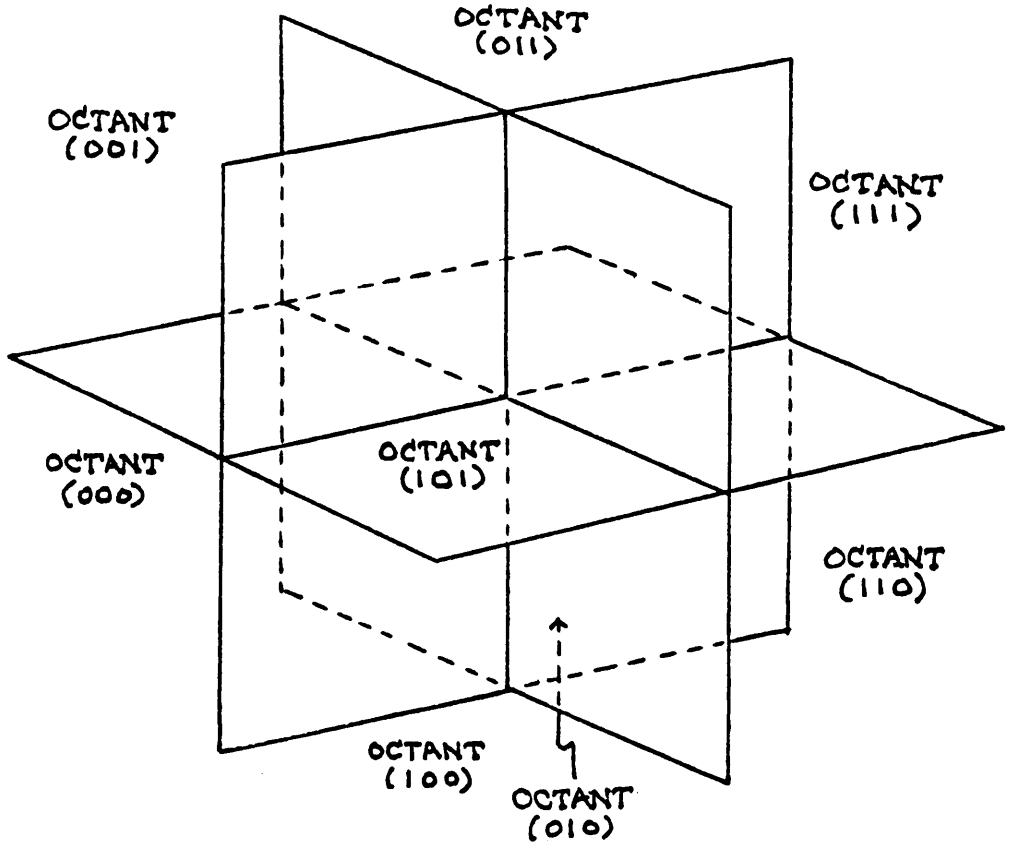
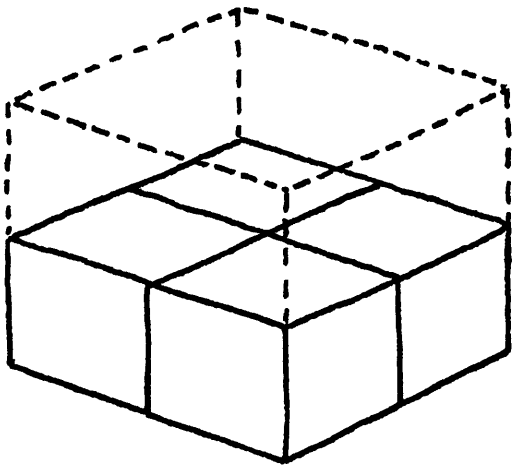

OCTANT
(011)
OCTANT
(001)
OCTANT
(111)
OCTANT
(000)
OCTANT
(101)
OCTANT
(110)
OCTANT
(100)
OCTANT
(010)

FIGURE 3.1

OCTANTS FILLED:

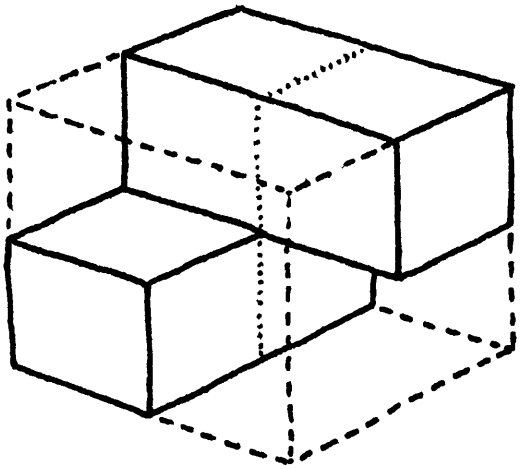
TOTAL NUMBER OF JUNCTION LABELS:

4
(CASE A)



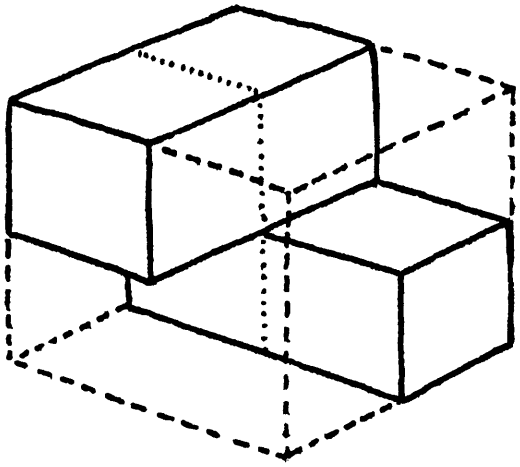
9

4
(CASE B)



15

4
(CASE C)



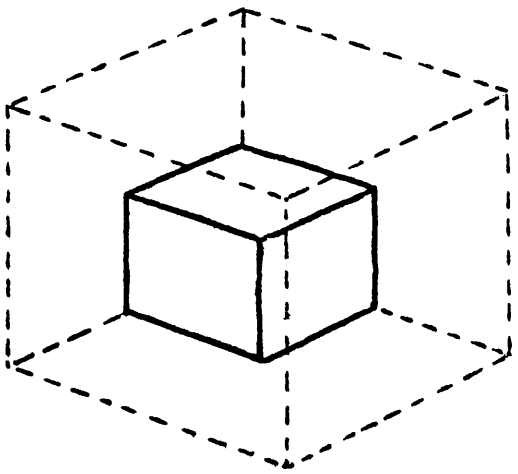
15

FIGURE 3.2A

OCTANTS FILLED:

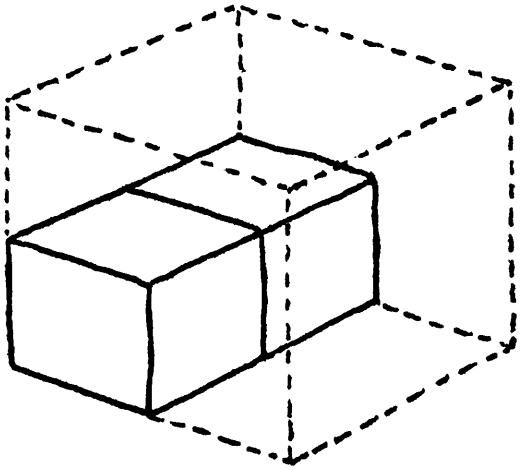
TOTAL NUMBER OF JUNCTION LABELS:

1



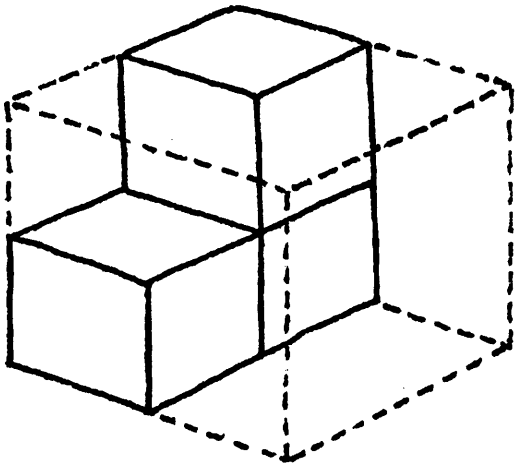
3

2



3

3



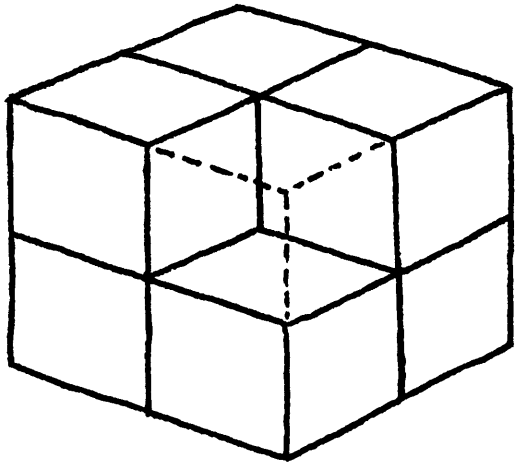
20

FIGURE 3.2A

OCTANTS FILLED:

TOTAL NUMBER OF JUNCTION LABELS:

7



21

FIGURE 3.2A

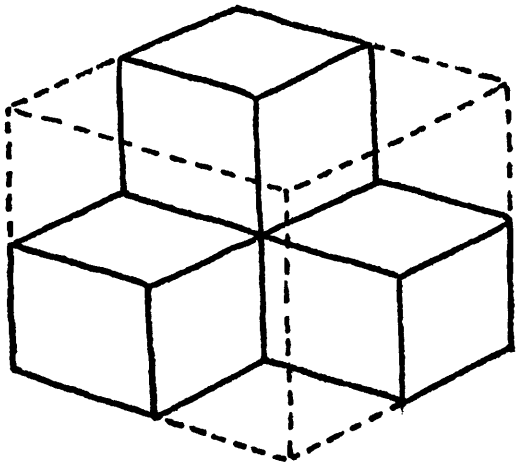
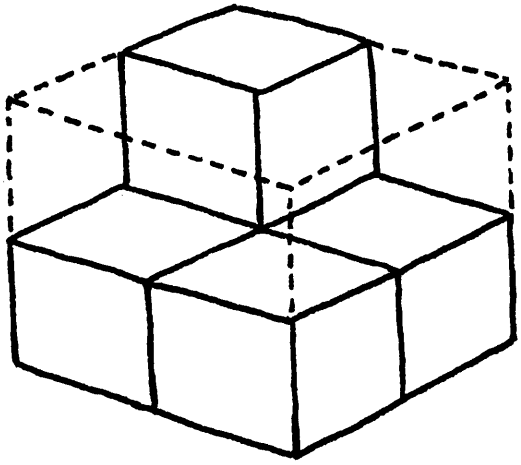
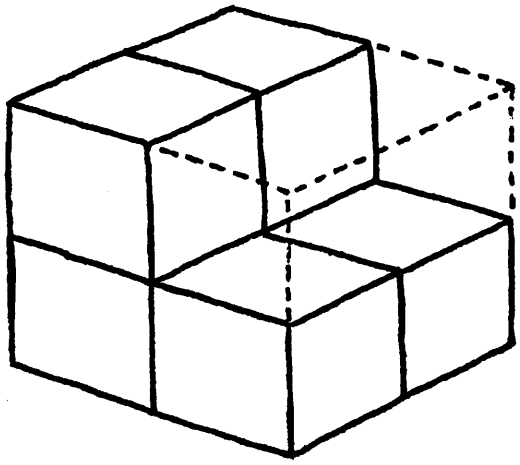
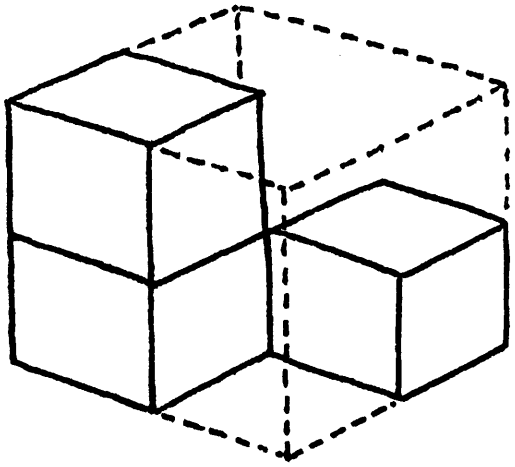
OCTANTS FILLED:		TOTAL NUMBER OF JUNCTION LABELS:
4 (CASE D)		8
5		56
6		46

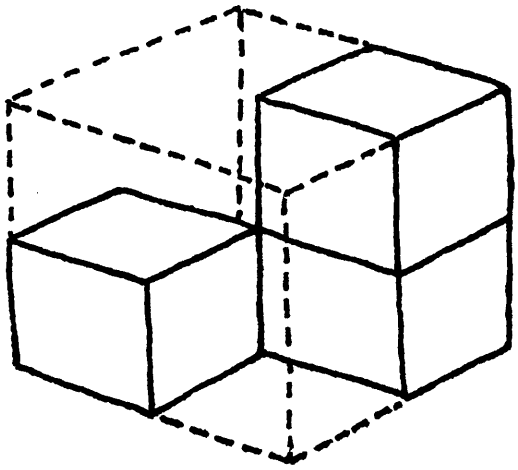
FIGURE 3.2A

OCTANTS FILLED:

3



3



4 or 5

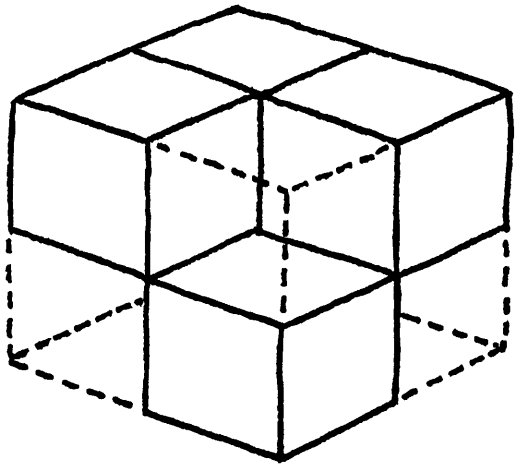
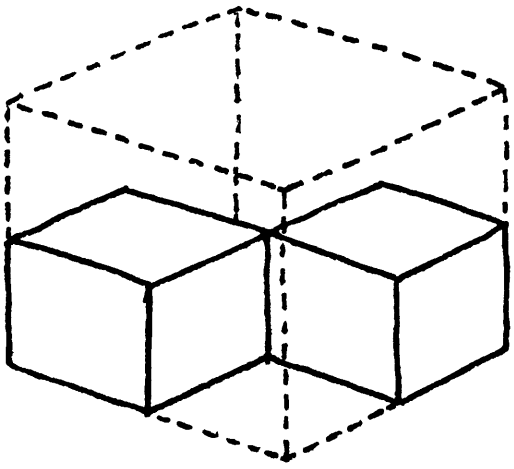


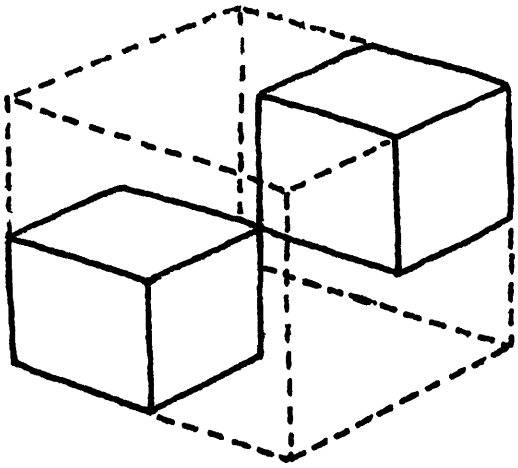
FIGURE 3.2B

OCTANTS FILLED:

2



2



3

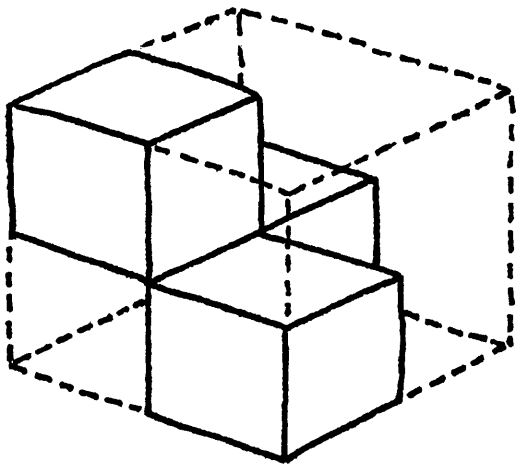


FIGURE 3.2B

OCTANTS FILLED:

5 or 6

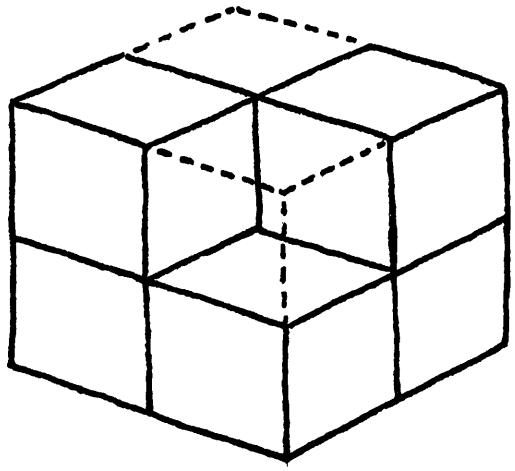
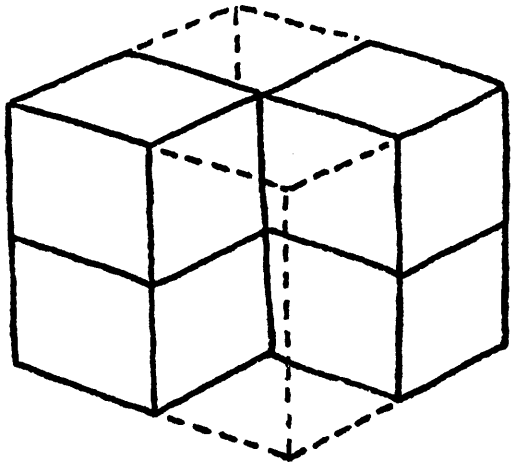


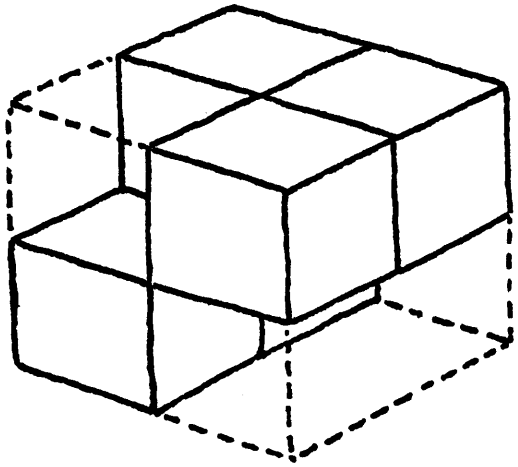
FIGURE 3.2B

OCTANTS FILLED:

4 or 5



5



5

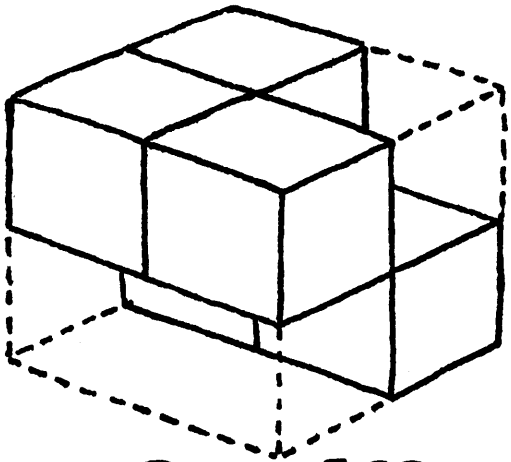
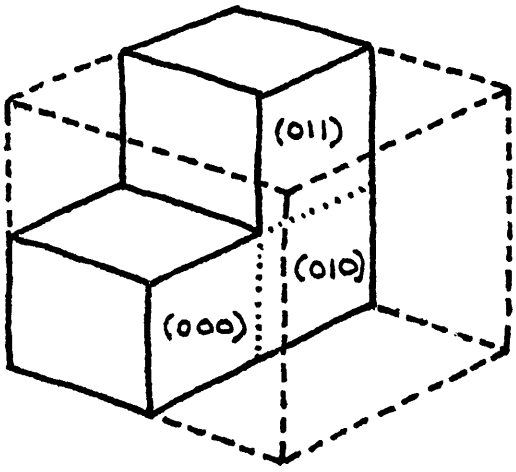


FIGURE 3.2B



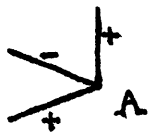
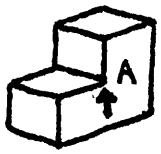
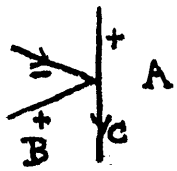
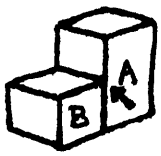
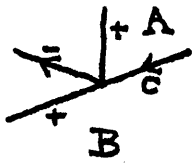
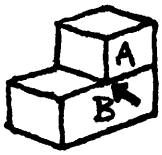
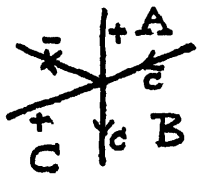
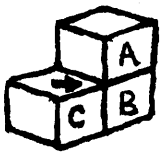
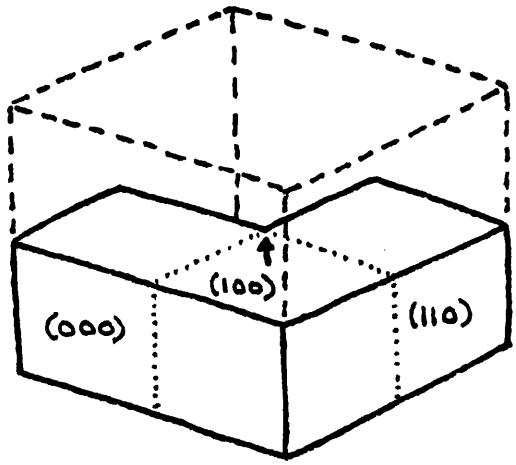
NAME OF JUNCTION (FROM APPENDIX 1):	APPEARANCE & LABELING OF JUNCTION:	# OF OBJECTS AT VERTEX	OBJECTS AT VERTEX ARE:	APPEARANCE OF CORRESPONDING OBJECT(S):
ARROW-3A		1	A = (000) ∪ (010) ∪ (011)	
K-3A		2	A = (010) ∪ (011) B = (000)	
K-3B		2	A = (011) B = (000) ∪ (010)	
KXX-3A		3	A = (011) B = (010) C = (000)	

FIGURE 3.3



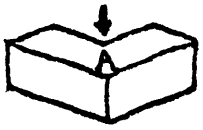
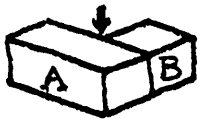
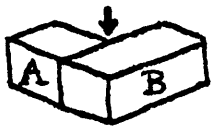
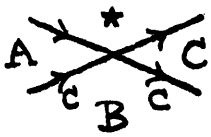
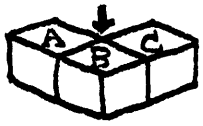
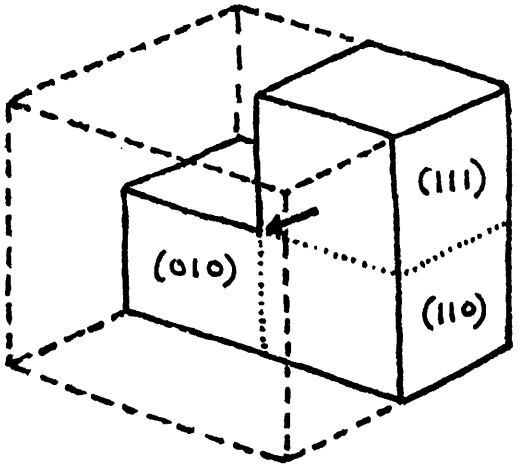
NAME OF JUNCTION (FROM APPENDIX 1):	APPEARANCE & LABELING OF JUNCTION:	# OF OBJECTS AT VERTEX	OBJECTS AT VERTEX ARE:	APPEARANCE OF CORRESPONDING OBJECT(S):
L-3A		1	A = (000) ∪ (100) ∪ (110)	
T-3A		2	A = (000) ∪ (100) B = (110)	
T-3B		2	A = (000) B = (100) ∪ (110)	
XX-3A		3	A = (000) B = (100) C = (110)	

FIGURE 3.3



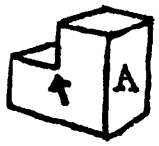
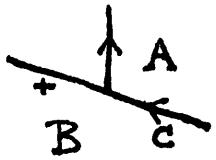
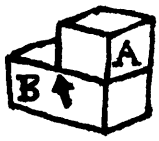
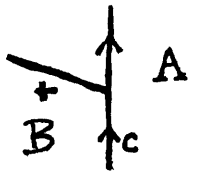
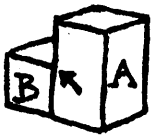
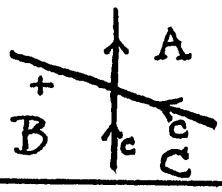
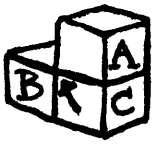
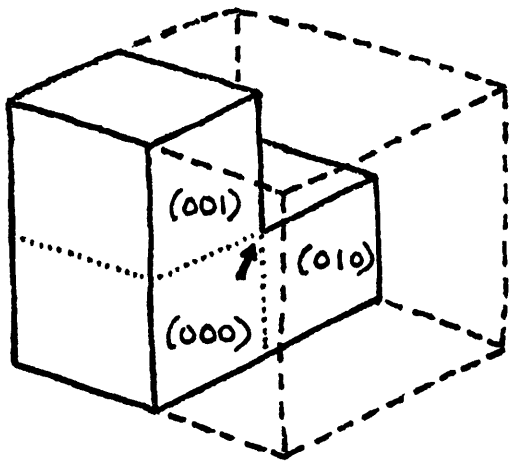
NAME OF JUNCTION (FROM APPENDIX 1):	APPEARANCE & LABELING OF JUNCTION	# OF OBJECTS AT VERTEX	OBJECTS AT VERTEX ARE:	APPEARANCE OF CORRESPONDING OBJECT(S):
L-3C		1	A=(111) ∪ (110) ∪ (010)	
T-3E		2	A= (111) B=(110) ∪ (010)	
T-3F		2	A=(111) ∪ (110) B=(010)	
XX-3C		3	A= (111) B= (010) C= (110)	

FIGURE 3.3



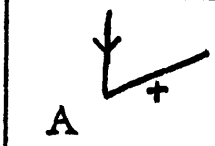
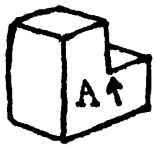
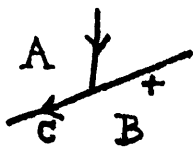
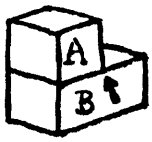
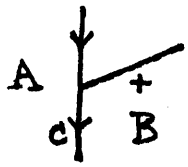
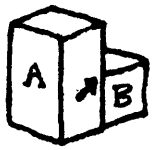
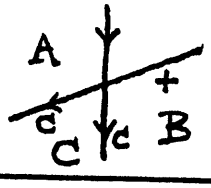
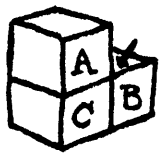
NAME OF JUNCTION (FROM APPENDIX 1):	APPEARANCE & LABELING OF JUNCTION:	# OF OBJECTS AT VERTEX	OBJECTS AT VERTEX ARE:	APPEARANCE OF CORRESPONDING OBJECT(S):
L-3B		1	A=(001) ∪ (000) ∪ (010)	
T-3C		2	A=(001) B=(000) ∪ (010)	
T-3D		2	A=(001) ∪ (000) B=(010)	
XX-3B		3	A=(001) B=(010) C=(000)	

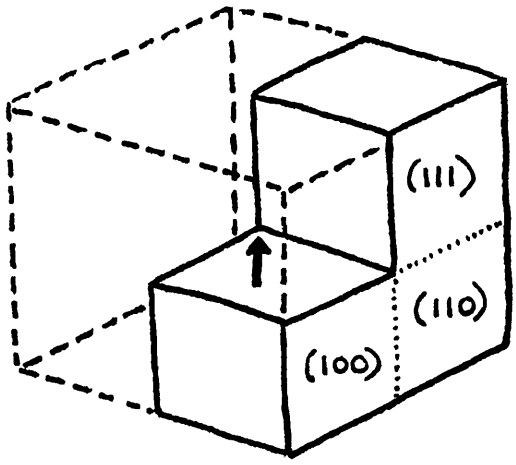
FIGURE 3.3

ways that the solid segments can be connected or separated, and finding all the possible views for each partitioning of the quadrants. To generate all the possible views one can either draw or imagine the particular geometry as it appears when viewed from each octant. From some viewing octants the central vertex is blocked from view by solid material, and therefore not every viewing position adds new labelings. Appendix 1 is obtained by applying this process to each of the geometries in figure 3.2A.

Whenever one of the regions at a junction could correspond to the background (i.e. the region is not part of one of the three planes which intersect at the vertex) I have marked the region with a star (★) both in figure 3.3 and Appendix 1. Later I will show how to use this information to aid the selection rules. Only 37 out of the 196 labels in Appendix 1 can occur on the scene/background boundary.

3.2 A USEFUL HEURISTIC

This section previews the general discussion later concerning how to choose a single labeling if ambiguities are still left at the end of the regular program's operation. The regular program keeps every conceivable interpretation.



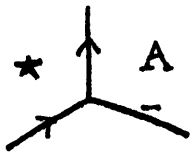
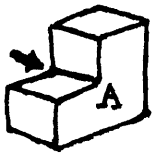
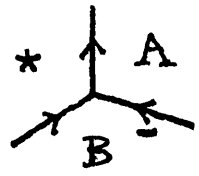
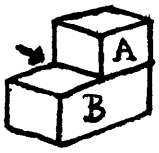
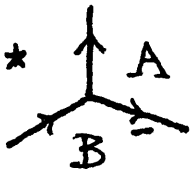
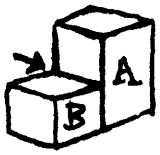
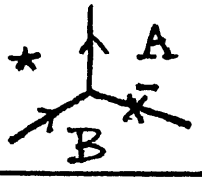
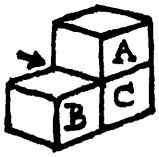
NAME OF JUNCTION (FROM APPENDIX 1):	APPEARANCE & LABELING OF JUNCTION:	# OF OBJECTS AT VERTEX	OBJECTS AT VERTEX ARE:	APPEARANCE OF CORRESPONDING OBJECT(S):
FORK-3A		1	A = (111) ∪ (110) ∪ (100)	
FORK-3B		2	A = (111) B = (110) ∪ (100)	
FORK-3C		2	A = (111) ∪ (110) B = (100)	
FORK-3D		3	A = (111) B = (100) C = (110)	

FIGURE 33

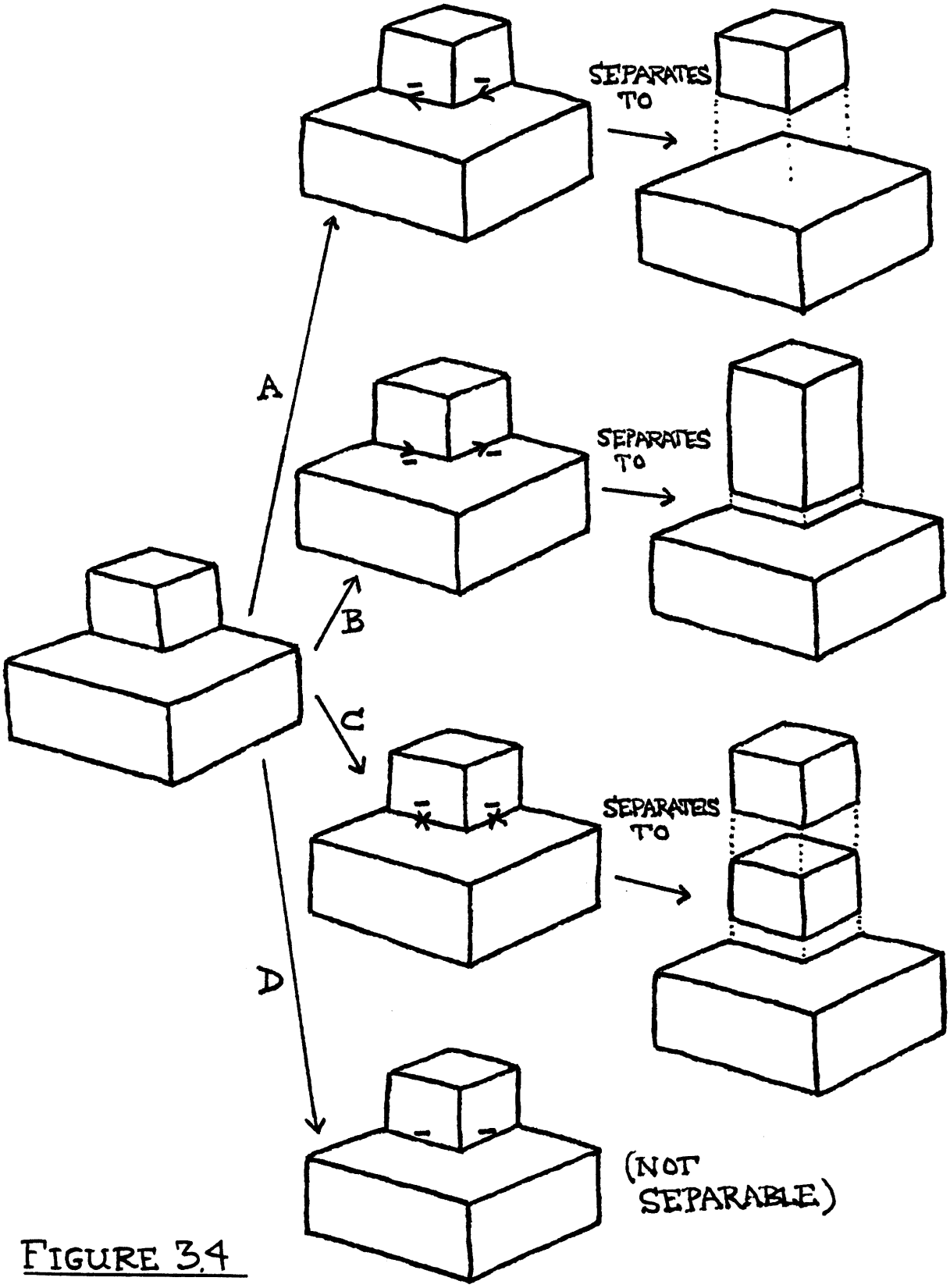


FIGURE 3.4

Clearly in some cases the scene is essentially ambiguous, i.e. human beings can interpret the scene in more than one way.

Given the line drawing shown in figure 3.4, how can a program decide which of the interpretations, A, B, C or D, is "correct"? In a picture there may be cues about how the objects should be separated in the details of the edges L-J1-J2 and L-J2-J3 of figure 3.4. But given only the line drawing of figure 3.4, the program will find the four interpretations listed. Because we generally prefer the scene interpretation which has the smallest number of convex objects, I have appropriately marked all junction labelings which include either concave edges (whether visible or not) or three-object edges. The output of the regular program is then a single label or list of labels for each junction. Obviously if there is only a single label, then there is nothing left to do. But if more than one label is left, it can purge labels corresponding to concave or three-edge junctions.

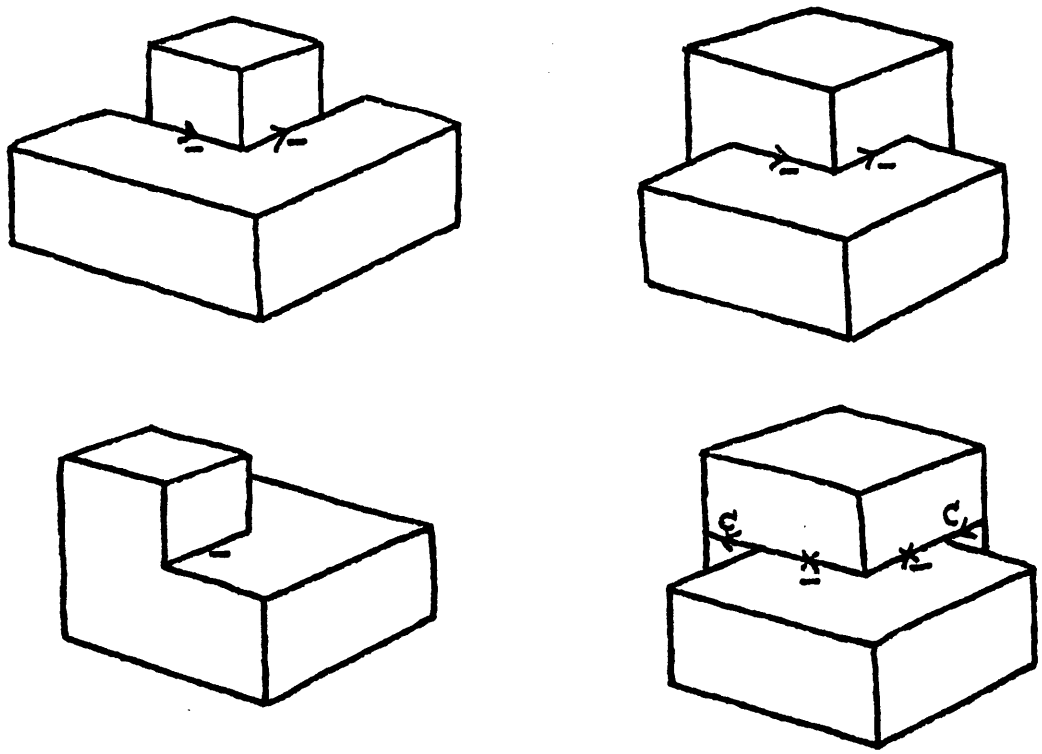
This heuristic correctly labels all the scenes shown in figure 3.5A, but finds the wrong labeling for figure 3.5B because it always prefers to interpret scenes as made up of

convex objects, and does not know enough to preclude the convex labeling in this case because object A in figure 3.5B has no support. Of course, for ambiguous scenes like figure 3.4 the heuristic selects interpretation A.

3.3 SHADOWS AT TRIHEDRAL VERTICES

To find all the variations of these vertices which include shadow edges, first note that vertices with 1, 2, 6 or 7 octants filled cannot cause shadows such that the shadow edges appear as part of the vertex. This can be stated more generally: in order to be a shadow-causing vertex (i.e. a vertex where the caused shadow edge radiates from the vertex) there must exist some viewing position for the vertex from which either two concave edges and one convex edge or one concave edge and two convex edges are visible. Consider the geometries listed in figure 3.2A. First, a shadow-causing edge must be convex. Second, unless there is at least one concave edge adjacent to this convex edge, there can be no surface which can have a shadow projected onto it by the light streaming by the convex edge. Finally, a junction which has one convex and one concave edge must have at least one other convex or concave edge, since the convex edge and concave edge define at least three planes which cannot meet

THE PROGRAM WITH THE CONCAVE EDGE REMOVAL HEURISTIC LABELS EACH OF THESE AS SHOWN:



THESE ARE ALL "CORRECTLY" LABELED

THIS LABELING IS IMPOSSIBLE BECAUSE OF GRAVITY:

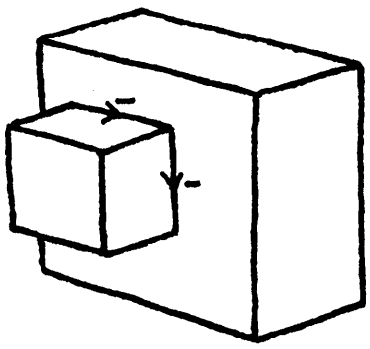
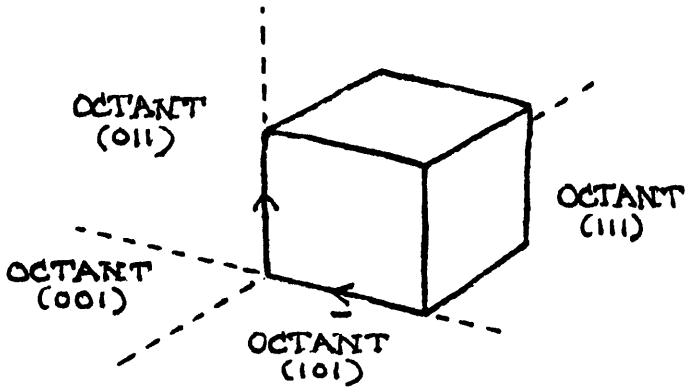


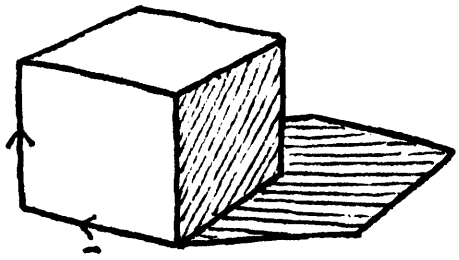
FIGURE 3.5

TO FIND ALL THE SHADOW POSSIBILITIES FOR A JUNCTION, FIRST IMAGINE IT AS PART OF AN OBJECT, AND DEFINE A COORDINATE SYSTEM CENTERED AT THE JUNCTION:

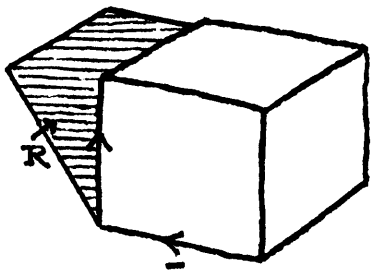


THEN IMAGINE THE LIGHT SOURCE TO BE IN EACH OF THE FOUR OCTANTS:

LIGHT IN (001):
(NO SHADOW EDGES VISIBLE AT VERTEX)



LIGHT IN (101):



LIGHT ON BOUNDARY OF (101) AND (111):
(DEGENERACY; LIGHT IN PLANE OF A.)

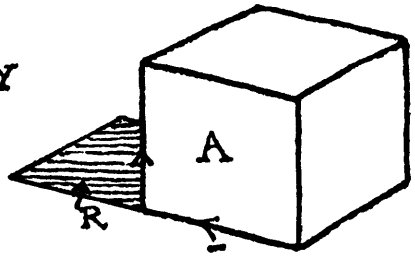
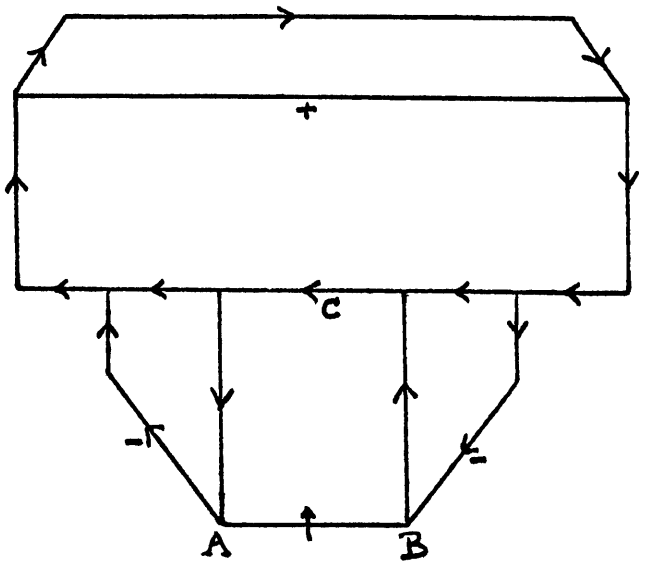


FIGURE 3.6

at any vertex with only two edges.

This immediately eliminates 73 out of 196 of the labels in Appendix 1 from consideration. Appendix 2 shows the shadow edges (if any) which can occur at each of the remaining vertices. Appendix 2 is constructed in the manner illustrated in figure 3.6; for each potential shadow-causing vertex, imagine the light source to be in each of the octants surrounding the vertex, and record all the resulting junctions. I have marked each shadow edge which is part of a shadow-causing junction with an "L" or "R" according to whether the arrow on the shadow edge points counterclockwise or clockwise respectively.

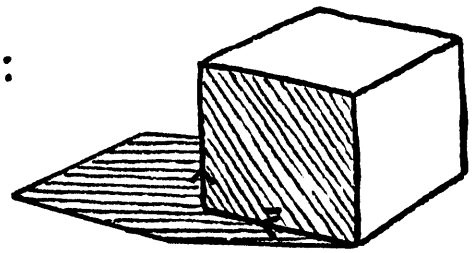
Any junction which contains either a clockwise shadow edge, marked "R," or a counterclockwise shadow edge, marked "L," is defined as a shadow-causing junction. The reason for distinguishing between the L and R shadow edges is that this prevents labeling an edge as if it were a shadow caused from both its vertices. Without this device there would be no way to prevent figure 3.7 from being labeled as shown, with line segment L-A-B interpreted as a shadow edge. (I use "L-" as a prefix to mean "line segment(s) joining the following points"; thus L-A-B is the line segment joining points A and



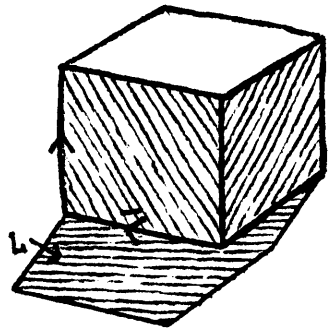
THIS INTERPRETATION IS PREVENTED BY ADDING "L" AND "R" MARKS TO THE SHADOW JUNCTIONS AT A AND B RESPECTIVELY SO THEY NO LONGER CAN MATCH.

FIGURE 3.7

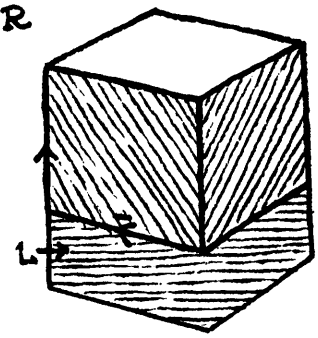
LIGHT IN (111):



LIGHT IN (011):



OR



OR

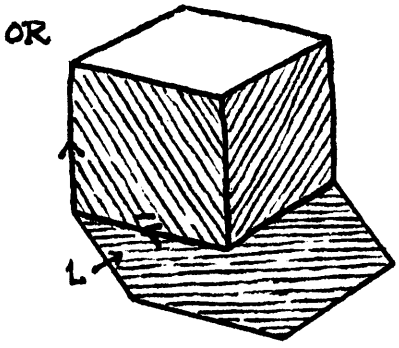
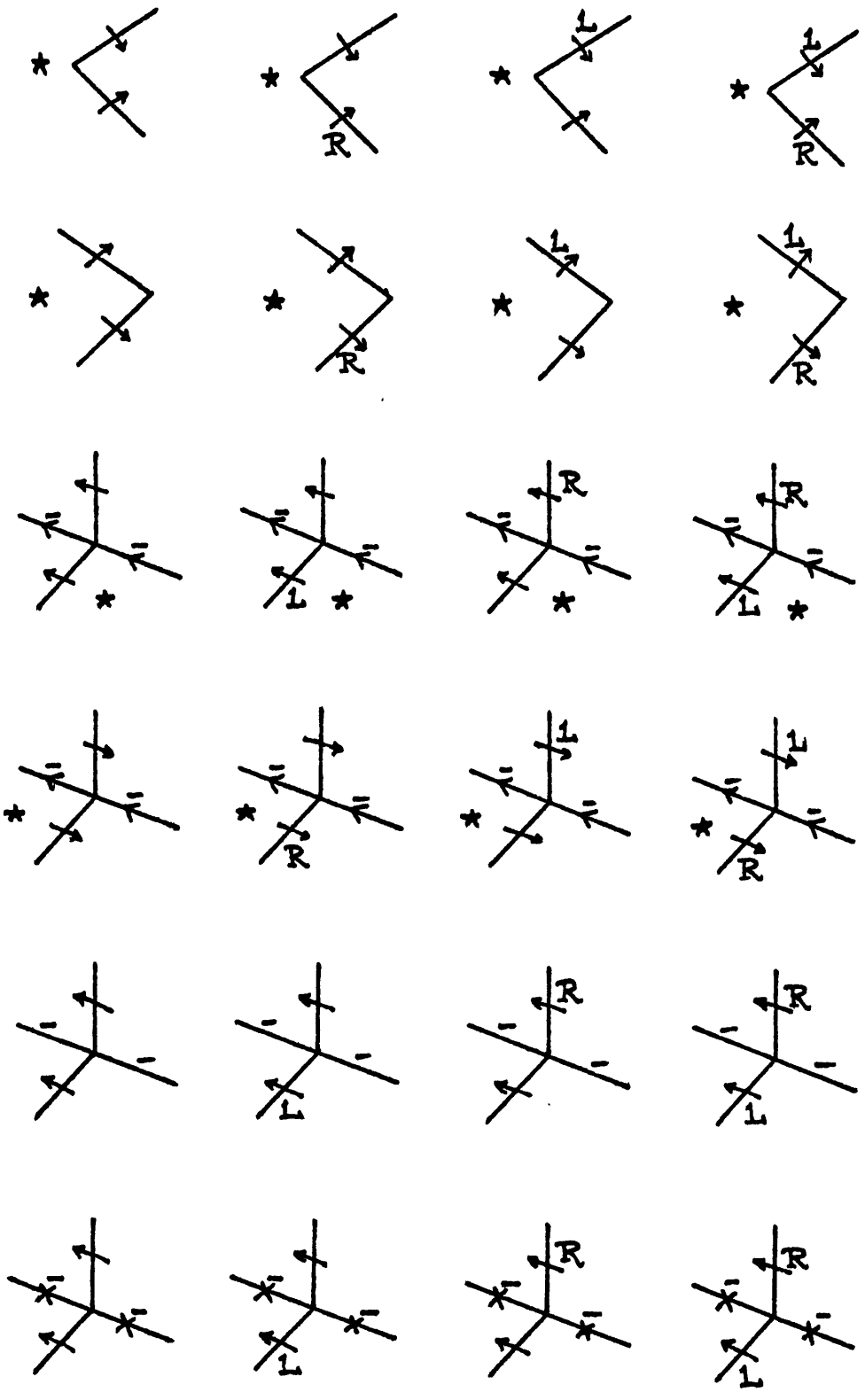


FIGURE 3.6

FIGURE 3.8A



B.) When the "L" and "R" marks are attached to each shadow causing junction, then the two shadow causing junctions at A and B in figure 3.7 no longer are compatible, and therefore the labeling shown will not be considered possible by the program.

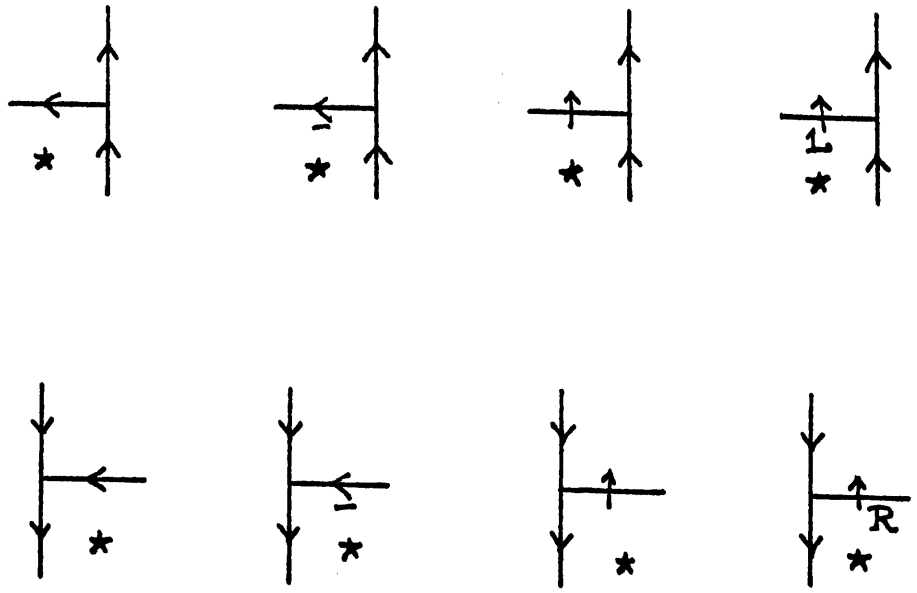
3.4 OTHER NON-DEGENERATE JUNCTIONS

I now must describe vertices which do not fall into the categories I have described so far. These include (1) all the rest of the combinations that shadow edges can form and (2) obscured edges.

In figure 3.8A I show all the other non-degenerate vertices which involve shadow edges, and in figure 3.8B I show all the obscured edges.

Later I return to the topic of junction labels and show how it is possible to also include junctions representing common degeneracies and accidental alignments as well as junctions with missing lines. In the degenerate cases I do not include every labeling possibility; instead I include the most common occurrences using certain observations about junctions. This is important since I do not want to limit the

THESE CAN OCCUR ON THE SCENE/BACKGROUND BOUNDARY:



THESE CAN ONLY OCCUR IN THE SCENE INTERIOR:

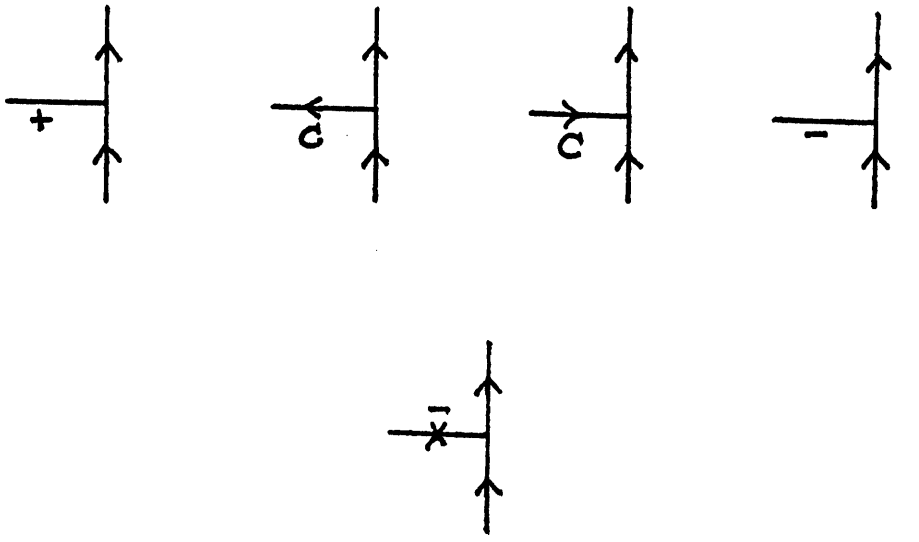


FIGURE 38B

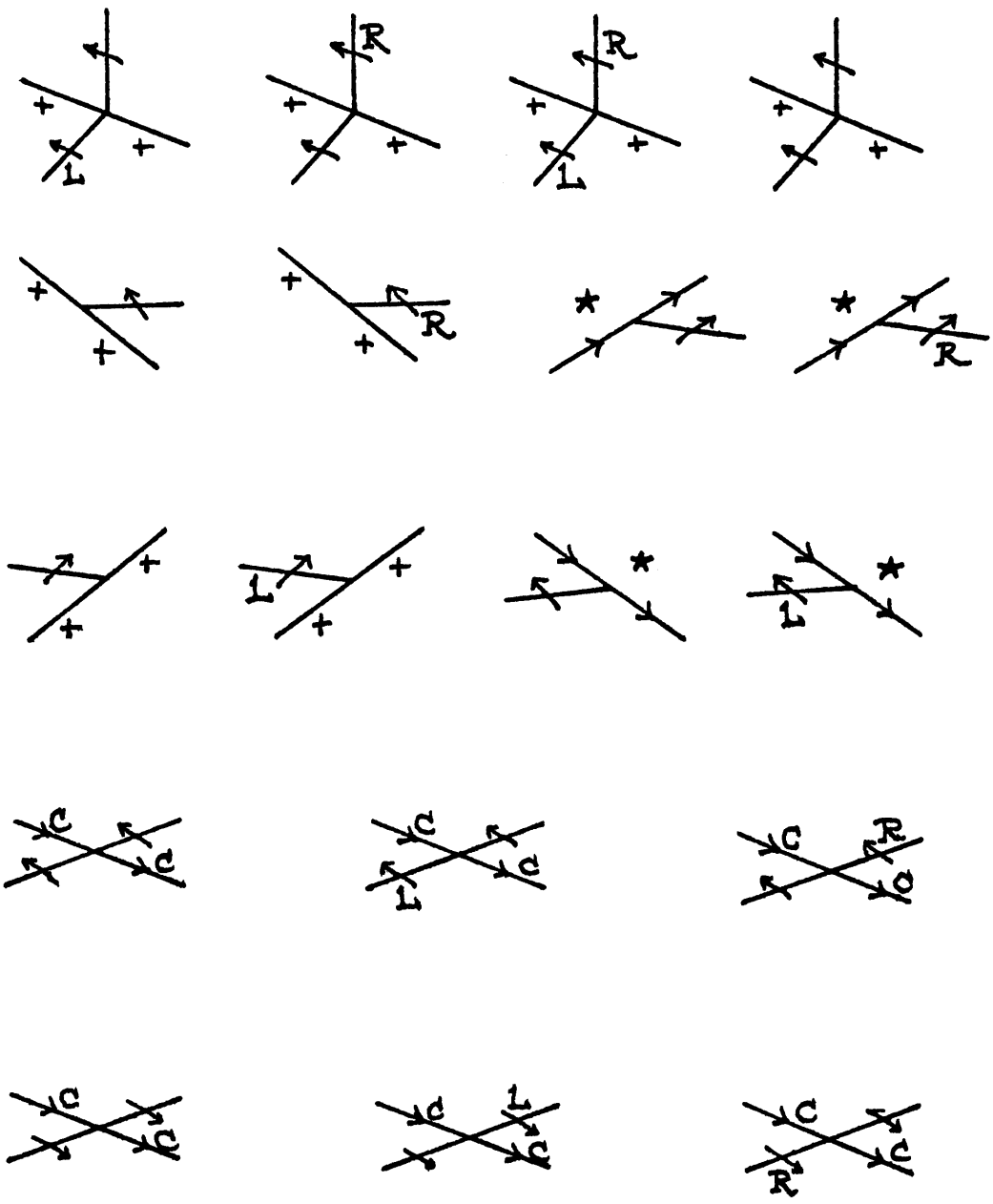


FIGURE 3.8A

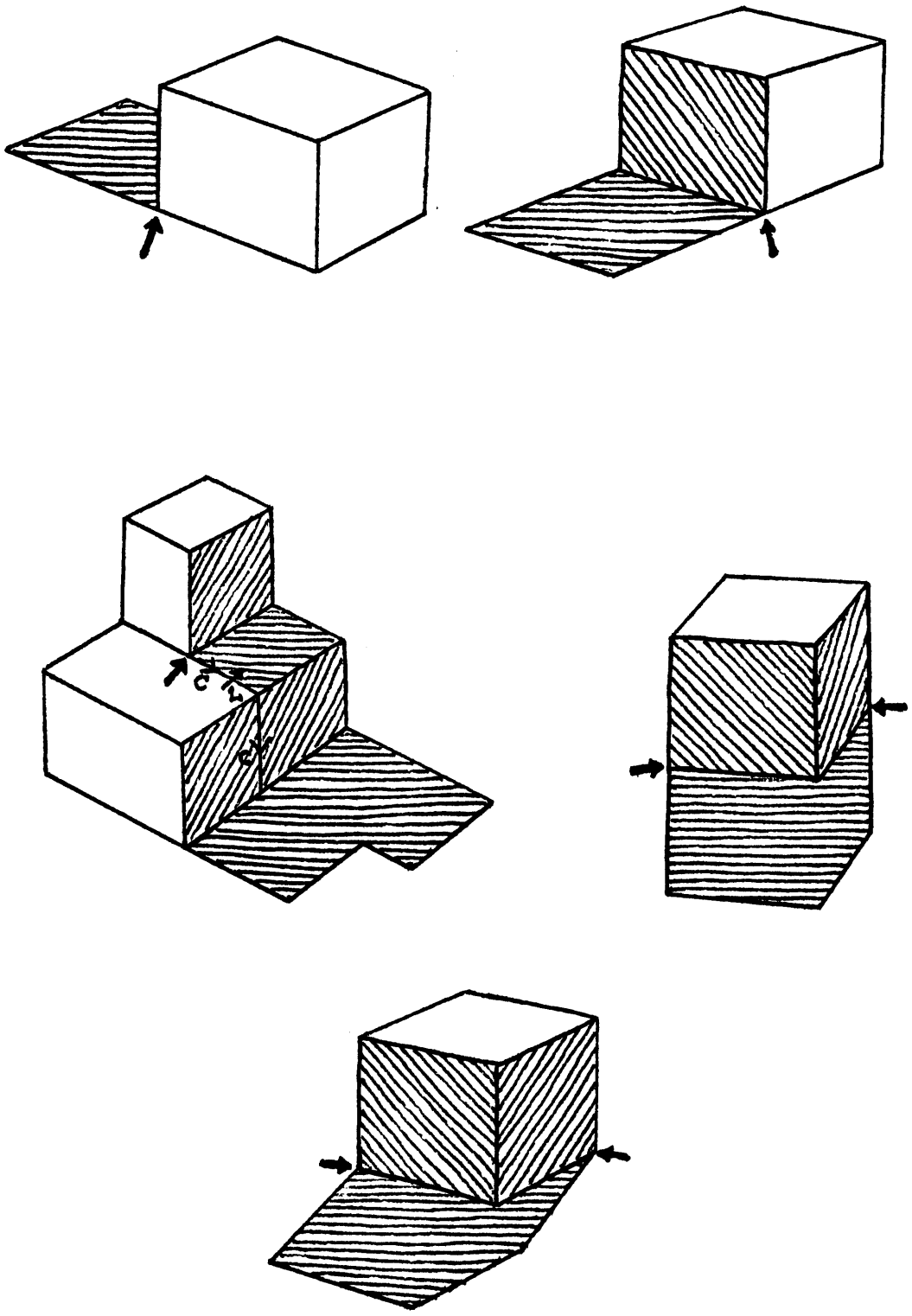


FIGURE 3.9

program to any particular set of objects. Fortunately certain types of junctions are rare no matter what types of objects are in a scene; for example, many junctions can only occur when the eye, light and object are aligned to within a few degrees, and when these junctions also contain unusual or aligned edges the combined likelihood of the junctions is low enough so that they can be safely omitted. As shown in Chapter 7, the program can still give information about junctions even if they do not have proper labelings listed in the data base, provided that not too many of these occur together in a single scene. Moreover, this approach is reasonable, since any additional ability to use stereo images or to move the eye or range-finding ability will allow a program to disambiguate most of these types of features.

3.5 A CLASS OF DEGENERACIES

As a final topic, I include one type of degeneracy which cannot be resolved by eye motion or stereo. This type of degeneracy results when the light source is placed in the plane defined by one of an object's faces. In this case, shadows are aligned with edges to produce junctions which are unlabelable given only the normal set of labels described so far. Two examples of such alignment are shown in figure 3.9A

The grand total number of legal trihedral junctions listed in this chapter is 505. The interesting thing in my estimation is that the number of junction labels, while fairly large, is very small compared to the number of possibilities if the branches of these junctions were labeled independently; moreover, even though I have not yet shown how to include various degeneracies and alignments, I believe that the set I have described already is sufficient for most scenes which a person would construct out of plane-faced objects, provided that he did not set out to deliberately confuse the program.

Since it may not be obvious what types of common vertices are non-trihedral, figure 3.10 contains a number of such vertices. Later sections show how to handle all of them.

and figure 3.9B and a complete listing of this type of junction is found in Appendix 3. I have excluded cases where a shadow edge is projected directly onto an edge of some other type (as in figure 3.9C). These cases are excluded since they would require me to define new edge labels which are of very limited value, although there is no technical difficulty in defining such edges and junctions. I also have excluded, for the time being, cases like the one shown in figure 3.9D, since the two junctions marked only appear to be T junctions when the eye is in the plane defined by the light source and the shadow-causing edge (L-A-B or L-C-D in figure 3.9D). If the eye is moved to the right, the shadow-causing junctions change to ARROWs or FORKs as illustrated in figure 3.9E. In contrast, notice that for the scenes shown in figures 3.9A and 3.9B, no change in eye position can make any difference in the apparent geometry of the shadow-causing junctions.

Later (in Chapter 6) I consider some of the common non-trihedral junctions which the program is likely to encounter. Some of these require me to define extra labels.

4.0 COMPLETING THE REGULAR DATA BASE

It would be hard to devise a program which could start with a few pieces of information and eventually yield the list of junctions described in Chapter 2. Moreover, even if such a program were written (which would indeed be theoretically interesting), it would be rather pointless to generate labels with it every time the labels are needed in an analysis. Instead the generating program could run once and save its results in a table. In this form the junction labelings table is a sort of compiled knowledge, computed once using a few general facts and methods. The knowledge in the current program is almost totally in this compiled form; this is the reason for its rapid operation, but I have paid a price for this speed in that I require a large amount of memory (about 14,000 words) to store the junction labelings. (All the rest of the labeling program occupies only about 4000 words of memory even though it is written in MICRO-PLANNER and LISP, neither of which are particularly noted for space efficiency.)

SOME COMMON NON-TRIHEDRAL VERTICES

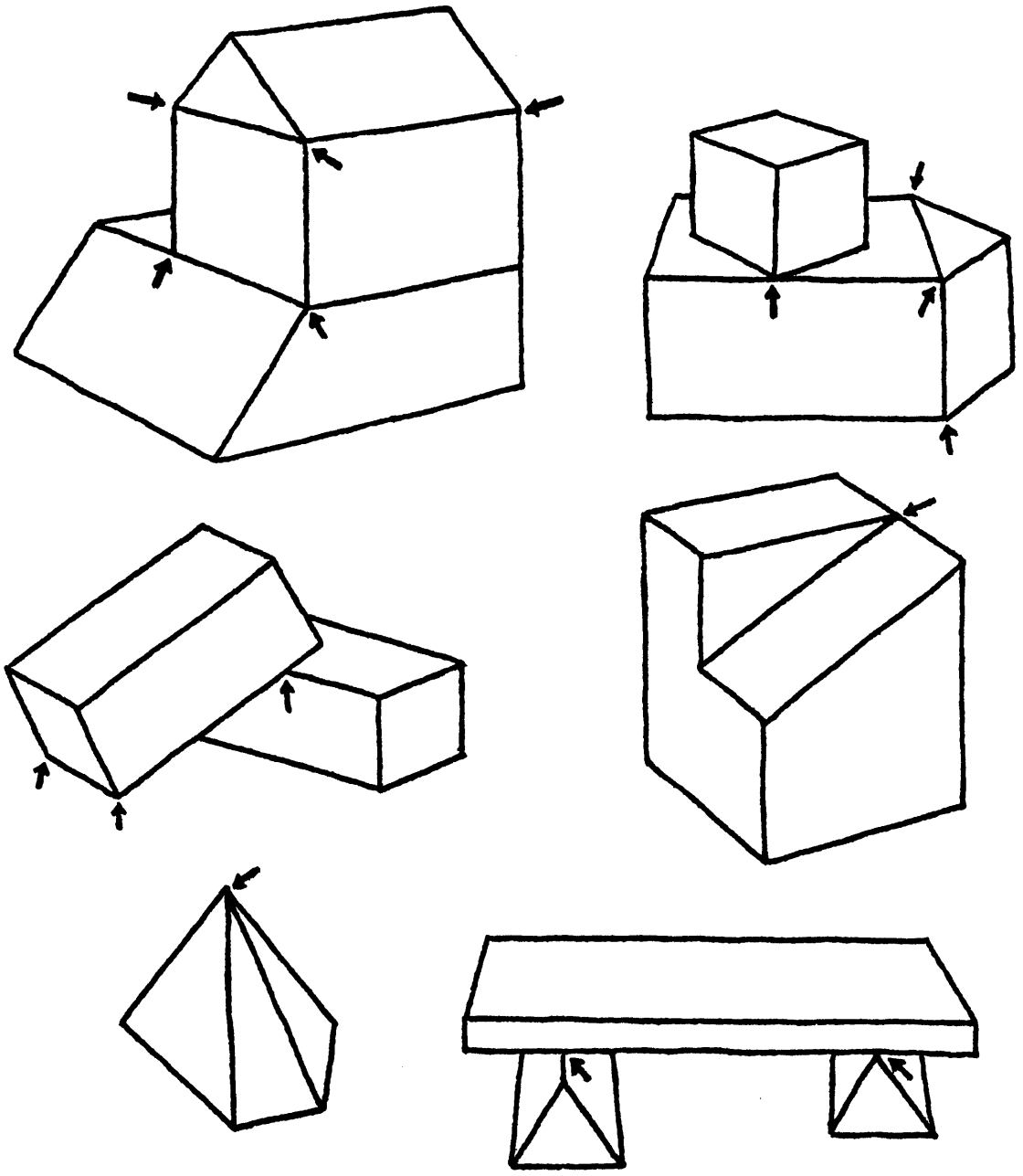


FIGURE 3.10

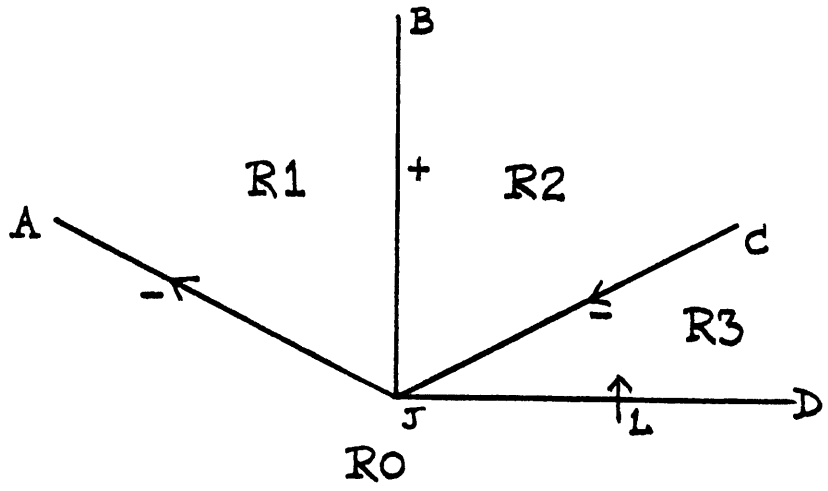


FIGURE 4.1

4.1 REGION ILLUMINATION ASSIGNMENTS

Given tables of allowable region illumination values (figure 1.6), it is easy to show how to write a program which expands the data base to include this information. Suppose that I wish to expand the labeling of the junction shown in figure 4.1 to include region illumination values. As coded for the data base, this labeling is:

(OCRM PLUS OCLM SHCCL)

where OCRM stands for OCclude Right Minus (see L-J-A in figure 4.1), PLUS represents the convex edge (see L-J-B in figure 4.1), OCLM stands for OCclude Left Minus (see L-J-C in figure 4.1), and SHCCL stands for SHadow CounterClockwise type L (see L-J-D in figure 4.1).

Each of these edges can separate regions which have the following values (the first element is the value of the region located counterclockwise with respect to the edge, the second element is the value of the region located clockwise with respect to the edge):

lists L1 and L2 that I defined earlier:

```
ILLUMINE (L1, L2)

 = ((I  I  I)     (I  I SS)
    (SP SP SP) (SP SP SS)
    (SP SS I)  (SP SS SP) (SP SS SS)
    (SS SP SP) (SS SP SS)
    (SS SS I)  (SS SS SP) (SS SS SS))

 = L4.
```

L4 is a list of triples which gives all the possible values for region illuminations in the regions R0, R1, and R2 in figure 4.1. To include R3, compute L5:

```
ILLUMINE (L4, L1)

 = ((I  I  I  I)
    (I  I SS SP)   (I  I SS SS)
    (SP SP SP SP) (SP SP SP SS)
    (SP SP SS SP) (SP SP SS SS)
    (SP SS I  I)
    (SP SS SP SP) (SP SS SP SS)
    (SP SS SS SP) (SP SS SS SS)
    (SS SP SP SP) (SS SP SP SS)
    (SS SP SS SP) (SS SP SS SS)
    (SS SS I  I)
    (SS SS SP SP) (SS SS SP SS)
    (SS SS SS SP) (SS SS SS SS))

 = L5.
```

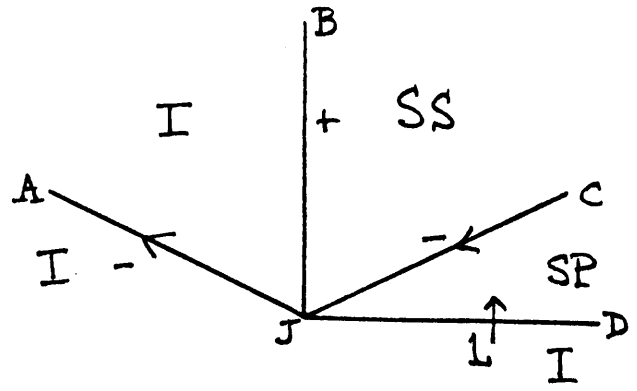
Now I only need to include the pairs for the line L-J-D, the shadow edge. Notice that very few of the possibilities for illumination can agree with R3 when R3 is forced to be a

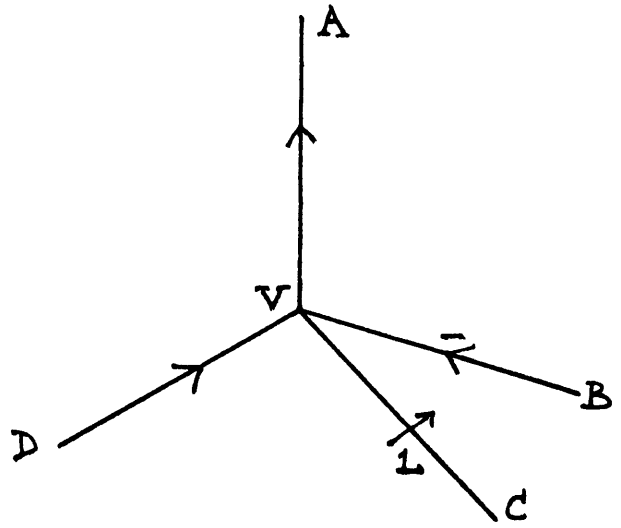
<The list of region illumination pairs for OCRM or OCLM>
= L1
= ((I I) (SP SP) (SP SS) (SS SP) (SS SS)).

<The list of region illumination pairs for PLUS>
= L2
= ((I I) (I SS) (SS I)
(SP SP) (SP SS) (SS SP) (SS SS)).

<The list of region illumination pairs for SHCCL>
= L3
= ((SP I)).

ILLUMINE is a function which takes two input lists as arguments, and returns a single output list. Each member of the output list is formed as follows: take a member of the second input list whose first element is the same as the last element of some member of the first input list. Concatenate these two and eliminate the duplication of the matching element. The ouput list is made up of every possible element which can be formed in this manner. While a verbal description may be somewhat difficult to understand, the function is not really very complicated, and I think the following example should make its operation clear. Using the





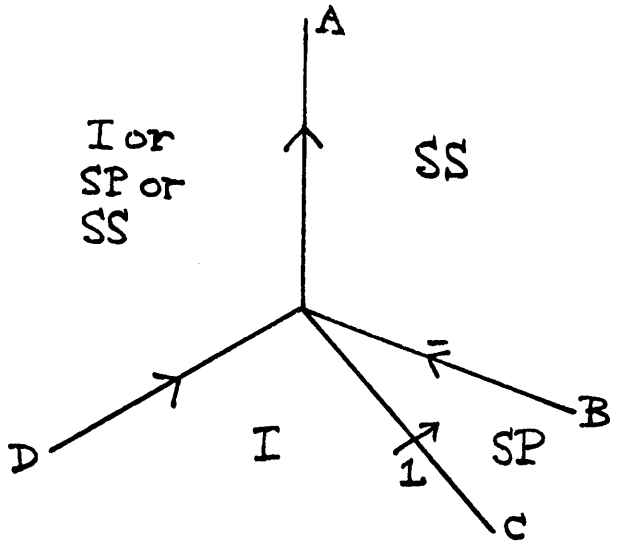


FIGURE 4.2

type SP region:

```
ILLUMINE (L5, L3)

 = ((I I SS SP I)
    (SP SP SP SP I)
    (SP SP SS SP I)
    (SP SS SP SP I)
    (SP SS SS SP I)
    (SS SP SP SP I)
    (SS SP SS SP I)
    (SS SS SP SP I)
    (SS SS SS SP I))

 = L6.
```

Now, to find the labelings for this junction, the last condition requires that since the first and last elements of each labeling in L6 both refer to R0, their values must be the same. Therefore I apply function FINALIZE, which only keeps members of a list whose first and last elements are the same:

```
FINALIZE (L6) = ((I I SS SP I)).
```

This represents the only possible region illumination labeling for this junction as shown in figure 4.2A. As I mentioned earlier, it is true in general that shadow-causing junctions (and a number of other junctions involving shadows) have only one possible region illumination labeling. The

In order to include illumination information in the data base, I merely append the region illumination value names to the name of each label. Thus I subdivide each label type (except shadow edge labels) into a number of possibilities, as shown in Table 4.1. As I mentioned in Chapter 2, expanding the number of line labels does not increase the total number of junction labels as much as one might imagine (see Table 2.2).

Fully 268 of the 505 labelings listed in Chapter 3, over half, have only one possible region illumination interpretation! The largest possible number of illumination interpretations for any junction is 3^n, where n is the number of junction branches. A number of T junctions actually have 27 interpretations (for example, this is true of any T made up of three occluding edges).

exceptions to this rule are shadow-causing junctions where one region segment of the junction is obscured by the vertex which gives rise to the junction. To understand this distinction, try finding the region illumination values for the junction in figure 4.2B as an exercise, especially if you are not entirely clear about the operation of ILLUMINE and FINALIZE. You will need the list of possible region illumination pairs for L-V-A and L-V-D in figure 4.2B; these edges can each be assigned any of the possible region illumination pairs:

```
<The list of region illumination pairs for OCR edges (such as
L-V-A) and OCL edges (such as L-V-D)>

 = ((I  I) (I SP) (I  SS)
    (SP I) (SP SP) (SP SS)
    (SS I) (SS SP) (SS SS))

 = L7.
```

Your answer should be:

```
   ((I SS SP I I)
    (SP SS SP I SP)
    (SS SS SP I SS))
```

The answer is illustrated in figure 4.2C.

NAMES OF OLD LABELS	APPEARANCE	NEW LABELS
OCR (OCCLUDE RIGHT)		OCR-II, OCR-ISP, OCR-ISS, OCR-SPI, OCR-SPSP, OCR-SPSS OCR-SSI, OCR-SSSP OCR-SSSS
OCL (OCCLUDE LEFT)		OCL-II, OCL-ISP OCL-ISS, OCL-SPI, OCL-SPSP OCR-SPSS OCL-SSI, OCL-SSSP OCL-SSSS
OLD TOTAL NUMBER: 15		NEW TOTAL NUMBER: 57

TABLE 4.1

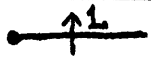
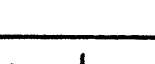
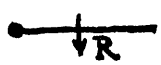
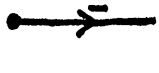
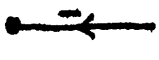
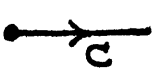
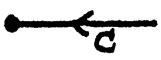
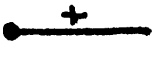
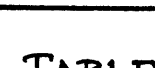
NAMES OF OLD LABELS	APPEARANCE	NEW LABELS
SHCC (SHADOW-COUNTER-CLOCKWISE)		SHCC-SPI
SHCCL		SHCCL-SPI
SHCCR		SHCCR-SPI
SHCL (SHADOW-CLOCKWISE)		SHCL-ISP
SHCLL		SHCLL-ISP
SHCLR		SHCLR-ISP
OCRM (OCCLUDE RIGHT-MINUS)		OCRM-II, OCRM-SPSP, OCRM-SPSS, OCRM-SSSP, OCRM-SSSS
OCLM (OCCLUDE LEFT-MINUS)		OCLM-II, OCLM-SPSP, OCLM-SPSS, OCLM-SSSP, OCLM-SSSS
M (CONCAVE JOINED)		M-II, M-SPSP, M-SPSS, M-SSSP, M-SSSS
MX (CONCAVE-3 OBJECTS)		MX-II, MX-SPSP, MX-SPSS, MX-SSSP, MX-SSSS
OCRCR (OCCLUDE RIGHT CRACK)		OCRCR-II, OCRCR-SPSP, OCRCR-SSSS
OCLCR (OCCLUDE LEFT CRACK)		OCLCR-II, OCLCR-SPSP, OCLCR-SSSS
PLUS (CONVEX)		PLUS-II, PLUS-ISS, PLUS-SSI, PLUS-SPSP, PLUS-SPSS, PLUS-SSSP, PLUS-SSSS

TABLE 4.1

TABLE 4.2

TOTAL NUMBER OF LABELS IN DATA BASE FOR EACH JUNCTION TYPE

JUNCTION TYPE	# OF LABELS BEFORE ADDING REGION ILLUMINATION	# OF LABELS INCLUDING REGION ILLUMINATION
L	24	92
ARROW	24	86
T	91	623
FORK	116	826
PEAK	10	10
K	42	213
X	129	435
XX	40	128
MULTI	96	160
KA	20	20
KX	60	76
KXX	25	121
SPECIAL	40	466
TOTALS	717	3256

4.2 SUMMARY OF THE DATA BASE

Although there are 505 labels listed in Chapter 3, the actual number of elements in the label lists for each junction will be larger than we might expect, since different permutations of labels count as different elements in some of the lists. The total number of list elements needed to represent the 505 labelings is 717, and this number expands to 3256 when the region illumination information is added to the labelings. Table 4.2 shows the number of elements in each list with and without region illuminated information. This table differs from Table 2.2 in that it includes only the differences in the list lengths which are caused by adding region illumination information.

A little cleverness is required to avoid duplicate labelings when including the different permutations of X junctions. This is because some X junctions give rise to two elements in the X labelings list, while the rest add only one element. Figure 4.3B shows an X junction which requires two elements to be added to the list, while figure 4.3C shows two labelings which each add only one element to the data base. Most shadow X junctions give rise to two elements in the data base, and most junctions without shadows give rise to one.

It is now possible to describe how the program handles each junction it encounters:

(1) If the junction is an L, ARROW, T, K, PEAK, X, KX, or KXX, it uniquely orders the junction's line segments (by choosing a particular line segment and considering the rest as ordered in a clockwise direction from this line segment).

(2) If the junction is a FORK, MULTI or XX, it chooses one line segment arbitrarily.

(3) It then fetches a list of labels which contains every possible set of assignments for the lines (excluding the possibilities of accidental alignments and degeneracies, and junctions with missing lines) and associates this list with the junction.

It makes absolutely no difference whether the program obtains this list from a table (the compiled knowledge case) or whether it must perform extensive computations to generate the list (the generated knowledge case). Similarly, it does not matter at all that various members of the list bear a particular relation to each other, e.g. as in the case of a

FIGURE 4.3A

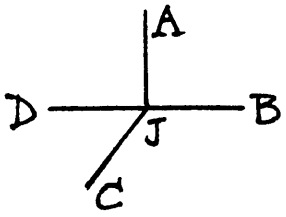
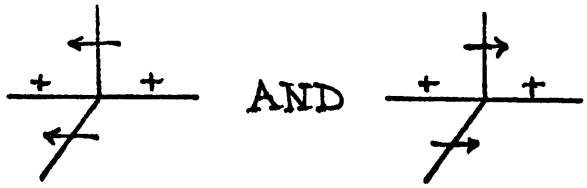
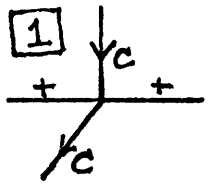


FIGURE 4.3B

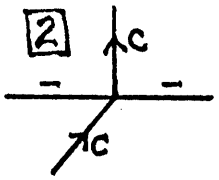


ARE BOTH POSSIBLE LABELINGS

FIGURE 4.3C

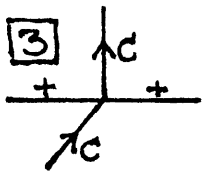


AND

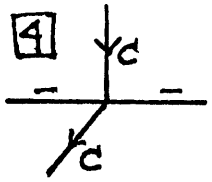


ARE BOTH POSSIBLE LABELINGS,

BUT

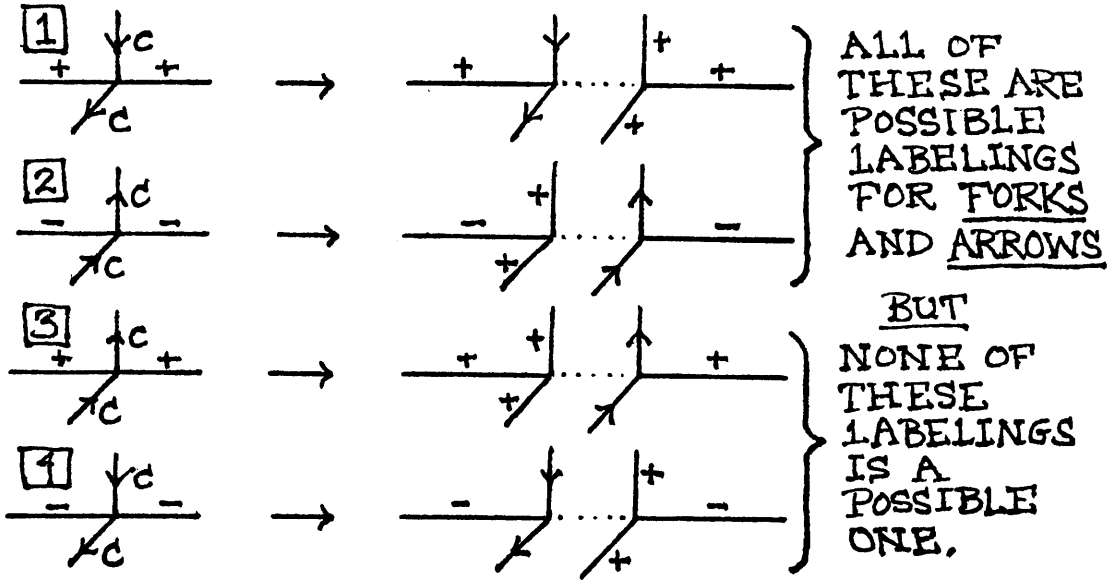


AND



ARE NOT.

TO SEE WHY, SEPARATE THE CRACK LABELS:

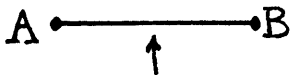


provide a unique labeling of edge geometry.)

FORK junction, where most elements of the list have two other elements which are permutations of the element. When I return to the issues of degeneracies, accidental alignments and missing lines, all I need to show is how the labelings corresponding to these cases can be added to the appropriate junction lists. The machinery to choose a particular element operates independently of just what the labelings actually are.

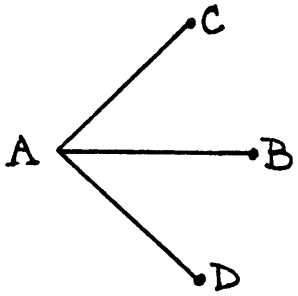
The only apparent exceptions are those labels marked to indicate that the vertices which cause them are either non-trihedral or concave or the result of alignment of surface and the light source. This information can be used optionally as the final step in the operation of the program if it is necessary to select a single labeling for an ambiguous junction. In such a case these marks enable the program to make a simple judgement about which interpretations are most likely. Of course if only single interpretations remain before the final step, or if I do not care that some junctions are not uniquely specified, then the program does not need to use these heuristics at all. (Such a case occurs when I only wish to find edge geometries and do not care about region illumination. Often ambiguous labels differ in the type of illumination for various regions but

FIGURE 5.1A

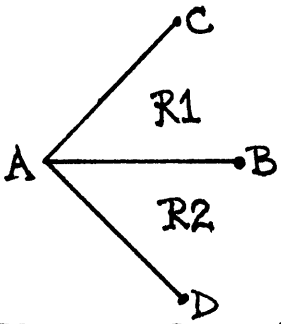


57 LABELS ARE POSSIBLE FOR L-A-B

FIGURE 5.1B

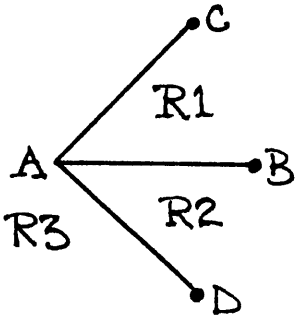


ONLY 19 LABELS ARE POSSIBLE FOR L-A-B IF IT IS KNOW TO BE THE MIDDLE BRANCH OF AN ARROW.



IF THE BRIGHTNESS IS KNOWN FOR R1 AND R2, THEN NO MORE THAN 18 AND AS FEW AS 15 LABELS WILL REMAIN POSSIBLE.

FIGURE 5.1C



IF THE BRIGHTNESS OF R3 IS ALSO KNOWN, THEN AS FEW AS 5 AND NO MORE THAN 18 LABELS WILL REMAIN POSSIBLE.

FIGURE 5.1D

5.0 SELECTION RULES

Now that I have shown how to generate a large number of possible labels for a junction, I will show how to go about eliminating all but one of them. The strategy for doing this involves:

(1) using selection rules to eliminate as many labels as possible on the basis of relatively local information such as region brightness or line segment directions, and

(2) using the main portion of the program to remove labels which cannot be part of any total scene labeling.

5.1 REGION BRIGHTNESS

If I know only that line segment L-A-B is a line in a scene, then it can theoretically be assigned any of the 57 possible labels. Once I know that L-A-B has an ARROW at one of its ends as shown in figure 5.1B, the number of possibilities drops to 19. Suppose that I know, in addition, the relative brightness of R1 and R2 in the neighborhood of L-A-B in figure 5.1C. There are three possibilities: (1) R1 is darker than R2, (2) R2 is darker than R1, or (3) the

Each junction from Chapter 3 which has a star in one of its segments is listed separately from junctions which have the same geometry but which cannot occur on the scene/background boundary. Thus the list of ARROW labels is divided into ARROW-B, a list made up of those labels which can occur on the scene/background boundary, and ARROW-I, made up of those which must occur on the interior of a scene. The total list of junctions which can also appear in the interior of a scene is found by appending ARROW-B to ARROW-I, since the scene/background labelings can appear on the interior of the scene as shown in figure 5.2. Table 5.1 lists the number of trihedral junction labels which can occur on the interior and on the scene/background boundary for each type of junction. Appendix 4 lists all of the junctions which can occur on the scene/background boundary including region illumination information. To obtain Appendix 4 I have assumed that the light source is positioned in one of the four octants of space above the support surface. This restriction means that the background is guaranteed to always be illuminated.

brightness of R1 is equal to the brightness of R2.

If (1) is true, I know for certain that if L-A-B is a shadow edge, then R1 must be the shadowed side and R2 the illuminated side. Obviously if (2) is true, then the opposite holds, i.e. R2 must be the shadowed side and R1 must be the illuminated side. If (3) is true, then it is impossible for L-A-B to be a shadow edge at all. (If I happen to also know that each object in a scene has all its faces painted identically with a non-reflective finish, then I can also eliminate more labels. In this case, if (1) is true, then L-A-B cannot be labeled as a convex edge with region R1 illuminated and R2 shadowed type SS, if (2) is true, then L-A-B cannot be labeled as convex with R2 illuminated and R1 shadowed type SS, and if (3) is true, then neither of these labels is possible.)

5.2 SCENE/BACKGROUND BOUNDARY REVISITED

It is easy to find all the junctions which can occur around the scene/background boundary. All that is necessary is to make a list of all the line segments which can occur along the boundary and then look for segments of junctions which are bounded by two members of this set.

TABLE 5.1 TOTAL NUMBER OF TRIHEDRAL JUNCTION LABELLINGS WHICH CAN APPEAR ON:

TYPE OF JUNCTION	THE INTERIOR OF A SCENE:	THE SCENE/ BACKGROUND BOUNDARY:
L	92	16
ARROW	86	12
T	623	96
FORK	826	26
PEAK	10	2
K	213	2
X	435	72
XX	128	3
MULTI	160	8
KA	20	—
KX	76	8
KXX	121	—
SPECIAL	466	—
TOTALS	3256	245

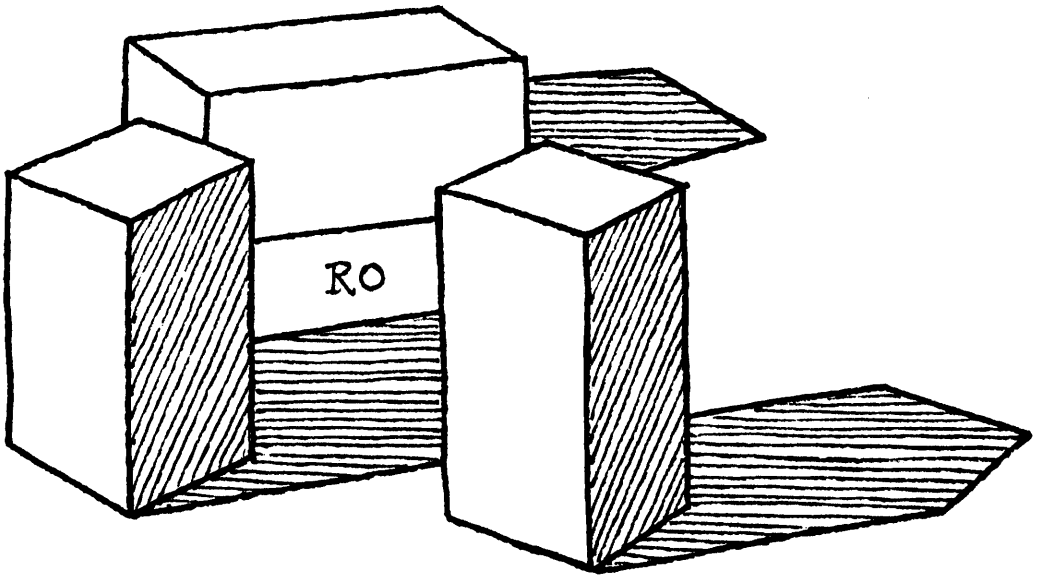


FIGURE 5.2

THE SAME JUNCTIONS AND EDGES CAN BORDER RO AS CAN APPEAR ON THE SCENE/BACKGROUND BOUNDARY.

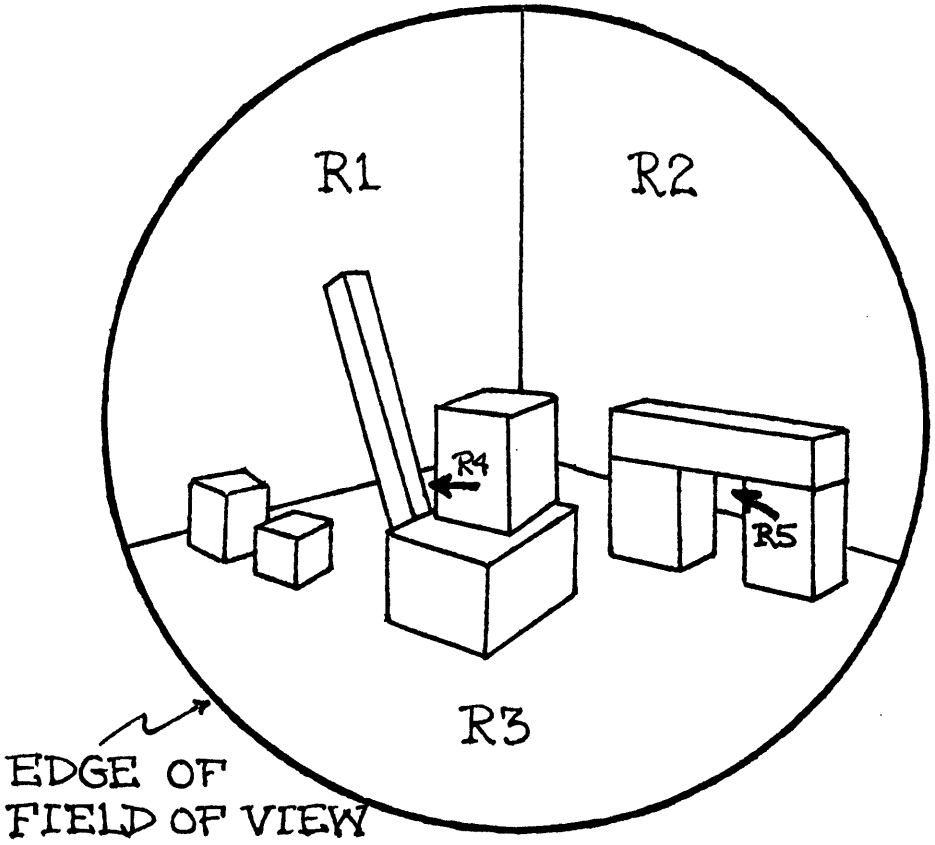


FIGURE 5.3

BY APPENDING ALL THE REGIONS WHICH TOUCH THE EDGE OF THE FIELD OF VIEW, WE OBTAIN ALL OF THE BACKGROUND EXCEPT THE SMALL REGIONS R4 AND R5. BY FINDING AND CONTINUING COLLINEAR OBSCURED LINE SEGMENTS (GUZMAN'S MATCHED T'S) THESE REGIONS CAN BE FOUND AND ADDED TO THE BACKGROUND ALSO.

Obviously, if I can determine which lines in the line drawing are part of the scene/background boundary, this knowledge can be used to great advantage. It is, in fact, not difficult to determine this boundary; any of several strategies will work. Two examples are:

(1) Look for regions which touch the edge of the field of view and append them all together, or

(2) Find the contour which has the property that every junction lies on or inside it (see Mahabala 1969).

Both of these methods require that the scene be completely surrounded by the background region or regions. As shown in figure 5.3, method (1) works even if the background is made up of more than one region.

Once the program has found which region is the background region, it can also find how each junction is oriented on the scene/background boundary. Some junctions always appear in the same orientation; for example, ARROW and PEAK junctions can only be oriented so that the background region is the region whose angle is greater than 180 degrees, and K junctions can only have the region whose

JUNCTION TYPE	DISTINGUISHABLE ORIENTATIONS (* INDICATES THE BACKGROUND REGION)

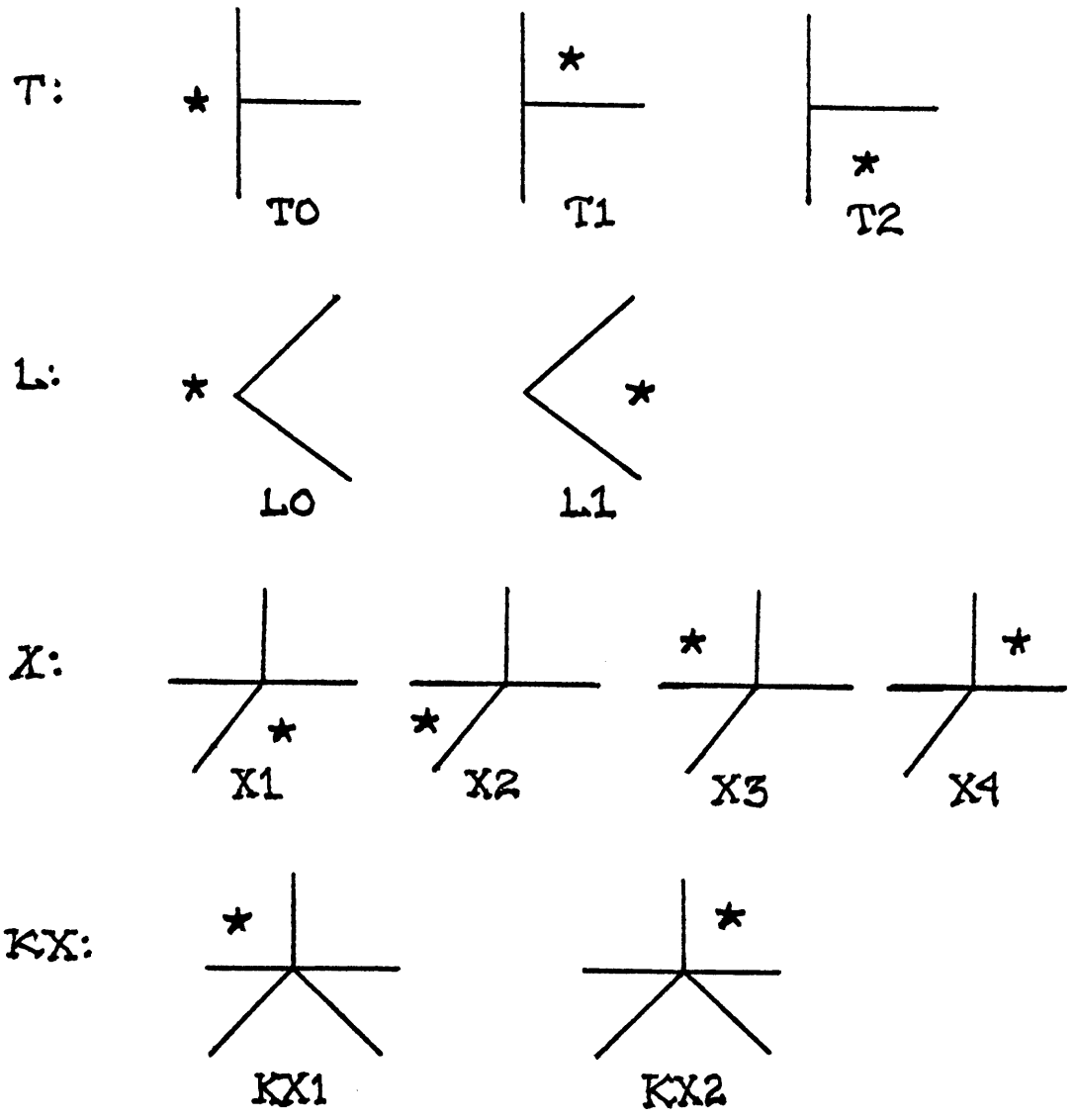


FIGURE 5.4

angle is 180 degrees as the background region (see Appendix 4).

Of course there is no way to easily define the orientations of FORK, XX, or MULTI junctions. However, as shown in figure 5.4, the L, T, X and KX junctions which appear on the scene/background boundary can be sorted according to which of their segments is the background region.

Consider figure 5.5. Each of the L, T, and X junctions is marked to indicate which orientation it has. Table 5.2 shows that this distinction makes a significant reduction in the size of the starting list of label assignments for these junctions.

5.3 EXTENDING THE SUPPORT SURFACE

Consider a problem posed by the scene shown in figure 5.6. If my labeling program is given this scene with the set of labels defined so far, the program will not find a unique labeling for L-C-D, even though it finds L-A-B to be a shadow edge, and therefore labels R1 as a projected shadow region.

TABLE 5.2	NUMBER OF LABELINGS IF IN THE SCENE INTERIOR:	NUMBER OF LABELINGS IF ON THE SCENE/BACKGROUND BOUNDARY**:	NUMBER IF ON SCENE/BACKGROUND BOUNDARY & ORIENTATIONS DISTINGUISHED:
T:	623	96	—
T0:	*	—	14
T1:	*	—	38
T2:	*	—	38
L:	92	16	—
L0:	*	—	9
L1:	*	—	7
X:	435	72	—
X1:	*	—	8
X2:	*	—	28
X3:	*	—	28
X4:	*	—	8
KX:	76	8	—
KX1:	*	—	4
KX2:	*	—	4

*THERE IS NO WAY TO DISTINGUISH A PREFERRED ORIENTATION IN THE INTERIOR OF THE SCENE.

**ASSUMING THAT I MAKE NO ORIENTATION DISTINCTIONS; THIS COLUMN COPIED FROM TABLE 5.1.

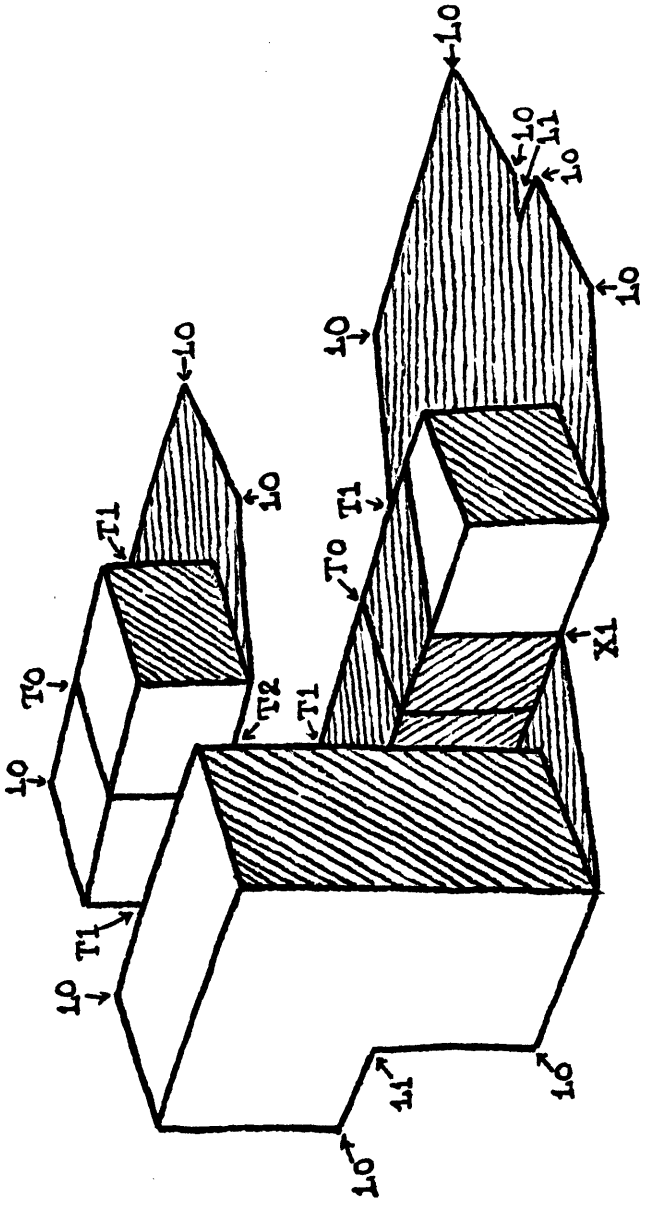
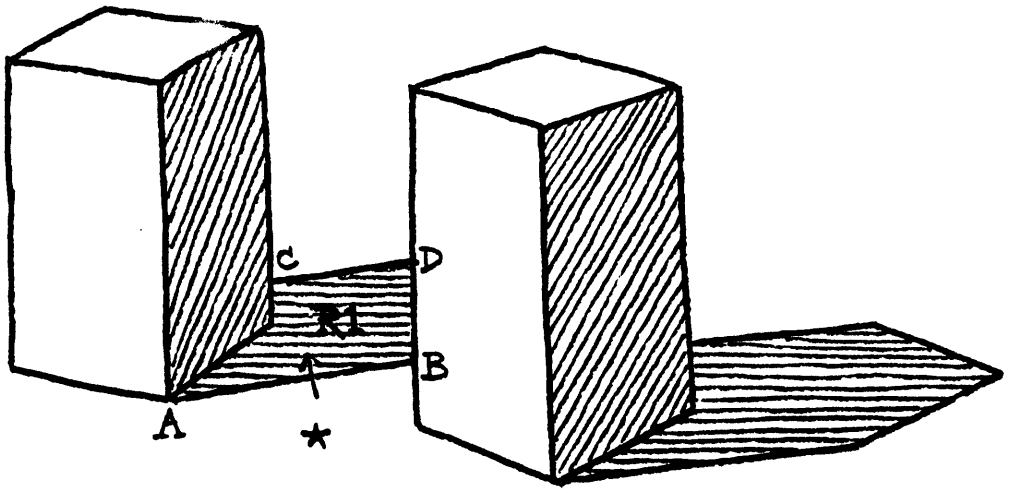
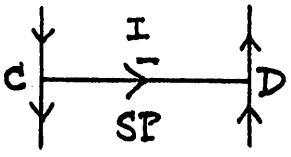


FIGURE 5.5

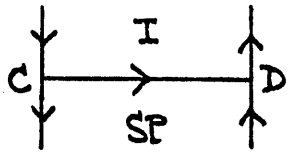
> At one time I thought that I would need to write a "demon" program which would check for shadow edges on the table, assert that such a shadow region is coplanar with the table, and then eliminate any edges other than planar ones whenever such a region shares an edge with the illuminated portion of the table. This type of approach seemed rather ad hoc to me, and started me thinking about how I could include region information as part of each junction label. There could be many added benefits to such an approach: it seemed clear that just as I was able to vastly reduce the number of labels from which to select possible ones by knowing that a junction was on the scene/background boundary, I should be able to reduce the number of labels for a junction which was interior to the scene but which had the table as one of its region segments.

Therefore I defined new labels as shown in Table 5.3 to denote any edge which has the table as one of its adjoining regions. Since I have restricted the light source to be in the quadrants of space above the support surface, I can be certain that any region which is part of the table can never be self-shadowed, type SS. I have used this fact in constructing Table 5.3. Any edge which touches or obscures the table is marked by appending a "T" to its name or printing a "T" next to the line segment. The old labels without "T" are understood to represent edges which do not have the table as either of their adjoining regions. The addition of these 24 edge types brings the total number of line labels to 81.

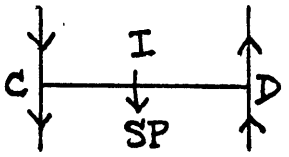




NOT A POSSIBLE LABELING; ELIMINATED BY THE PROGRAM.



POSSIBLE ACCORDING TO PROGRAM; INCORRECT.



POSSIBLE ACCORDING TO PROGRAM; CORRECT.

FIGURE 5.6

The tables which show the allowable region illumination pairs for these edges (analogous to figure 2.6) appear in Table 5.4.

To update the lists of junction labels, I must add to the present set:

(1) All the junctions listed in Appendix 4, but with "T" printed next to both line segments which bound the region containing the star. (These regions can be part of the background of the scene, i.e. the portion of the table which surrounds the scene and is illuminated.) Some of these junctions can also have other projected shadow regions which are part of the table, so that "T" must be added to line segments other than the two bounding the starred region. These junctions are listed in Appendix 5.

(2) All the junctions which can bound a projected shadow (type SP) region which is also part of the table.

Table 5.5 shows the situation now for the relative numbers of junctions which can occur on the scene/background boundary. While the numbers of labelings possible if the branches were labeled independently has increased sharply

TABLE 5.3

EDGES WHICH HAVE THE TABLE AS ONE ADJOINING REGION; STARRED REGIONS ARE PART OF THE TABLE WHICH IS ILLUMINATED AND CAN THEREFORE APPEAR ON THE SCENE/BACKGROUND BOUNDARY.

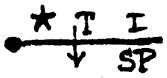
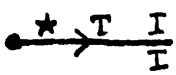
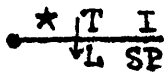
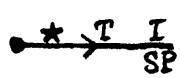
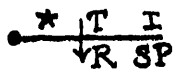
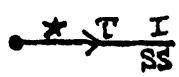
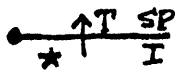
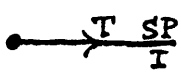
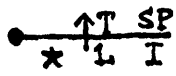
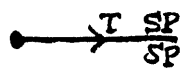
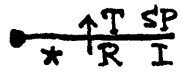
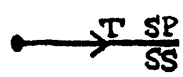
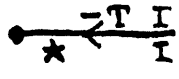
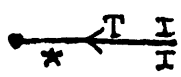
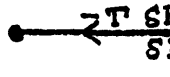
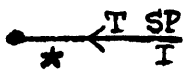
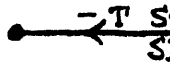
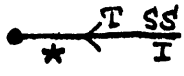
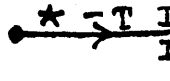
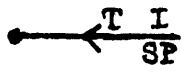
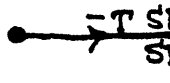
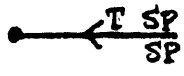
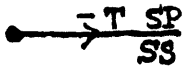
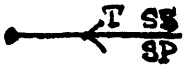
APPEARANCE	NAME	APPEARANCE	NAME
	SHCL-ISPT		OCR-IIT
	SHCLL-ISPT		OCR-ISPT
	SHCLR-ISPT		OCR-ISST
	SHCC-SPIT		OCR-SPIT
	SHCCL-SPIT		OCR-SPSPT
	SHCCR-SPIT		OCR-SPSST
	OCLM-IIT		OCL-IIT
	OCLM-SPSPT		OCL-SPIT
	OCLM-SSSPT		OCL-SSIT
	OCRM-IIT		OCL-ISPT
	OCRM-SPSPT		OCL-SPSPT
	OCRM-SPSST		OCL-SSSPT

TABLE 5.5	TOTAL NUMBER OF TRIHEDRAL JUNCTION LABELINGS, USING SET OF LABELS AUGMENTED BY "T" MARKING FOR:		NUMBER OF POSSIBILITIES IF EACH BRANCH LABELED
TYPE OF JUNCTION	THE SCENE INTERIOR	THE SCENE BACKGROUND BOUNDARY	INDEPENDENTLY (BASIS OF 81 LABELS)
L	118	16 {L0=9, L1=7}	6561
ARROW	109	12	521,441
T	809	96 {T0=14, T1=38, T2=38}	521,441
FORK	1012	26	521,441
PEAK	10	2	4.2×10^7
K	213	2	4.2×10^7
X	563	72 {X1=8, X2=28, X3=28, X4=8}	4.2×10^7
XX	152	3	4.2×10^7
MULTI	224	8	4.2×10^7
KA	20	—	3.3×10^9
KX	92	8 {KX1=4, KX2=4}	3.3×10^9
KXX	121	—	3.3×10^9
SPECIAL	466	—	$>2.6\times10^{11}$
TOTALS	3909	245	$>3.\times10^{11}$

TABLES OF ALLOWABLE ILLUMINATION PAIRS FOR NEW LABELS (CONTINUATION OF TABLE 2.6)

CONCAVE

1 | 2

-T

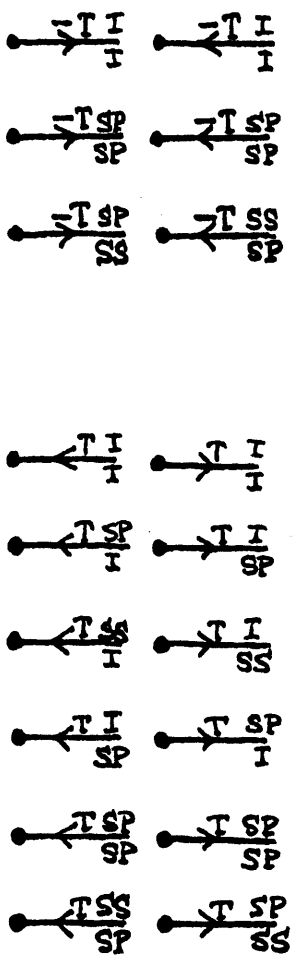
1 \ 2	I	SP	SS
I	YES	NO	NO
SP	NO	YES	YES
SS	NO	NO	NO

OBSCURE

1 | 2

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

SET OF ALLOWABLE LABELS:



ALL THE SHADOW COMBINATIONS ARE SAME AS THOSE SHOWN FOR SHADOWS IN FIGURE 2.6.

TABLE 5.4

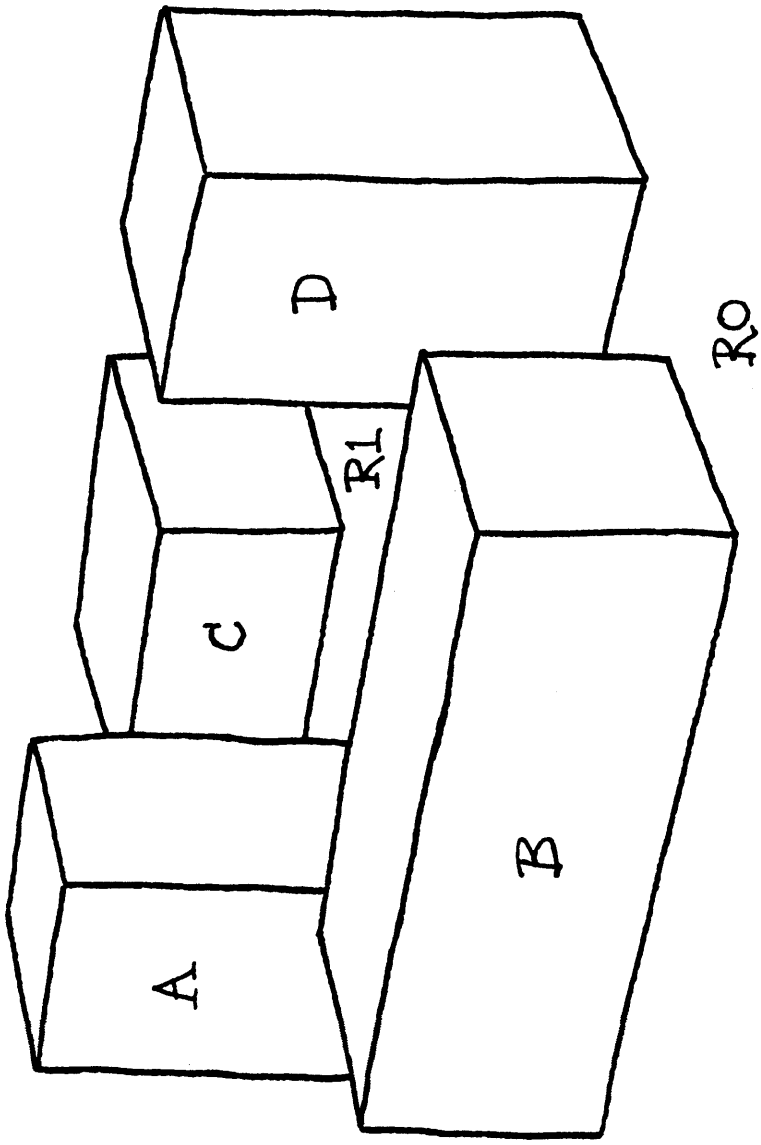


FIGURE 5.7

with the increase of the number of line labels from 57 to 81, the actual numbers of junction labelings has not changed for the scene/background boundary and has increased only moderately for the scene interior.

The value of these additions to the data base is especially pronounced for scenes like figure 1.2 where the table surface accounts for seven of the interior regions as well as the background region. In addition to the improvement for scenes of this sort, there are other benefits. Consider figure 5.7. How many objects are in this scene?

Now look at figure 5.8. Given figure 5.7, my program will return both interpretations: the one we would usually expect (region R as the table, with object C resting on the table) or the interpretation shown in figure 5.8. Thus the new labels enable the program to make finer distinctions than it could before. Notice that we could also use the table information to make another heuristic rule: if there are two interpretations of an interior region, one as the table and one as an extra object, choose the table interpretation. (This corresponds to choosing the simplest interpretation, i.e. the one with the fewest objects.)

5.4 DISCUSSION

This section is speculative; nothing in it is critical to an understanding of my program.

Underlying the previous section are some important kinds of distinctions between levels of understanding which I believe are worth pursuing at greater length at this point. There are several levels of understanding which a program can have about a particular property of scene features (e.g. "this region is part of the table"):

(1) the first level of understanding is that the program be able to express the fact that a given portion of the scene does or does not have the property. As an example, until the program had the labels which labeled regions that were part of the table, it could not express the difference between the two possible interpretations of figure 5.7.

(2) The next series of levels are ones where the program recognizes more and more instances of features which cannot have the property (and consequently recognizes more precisely where the property can apply). My program's hard

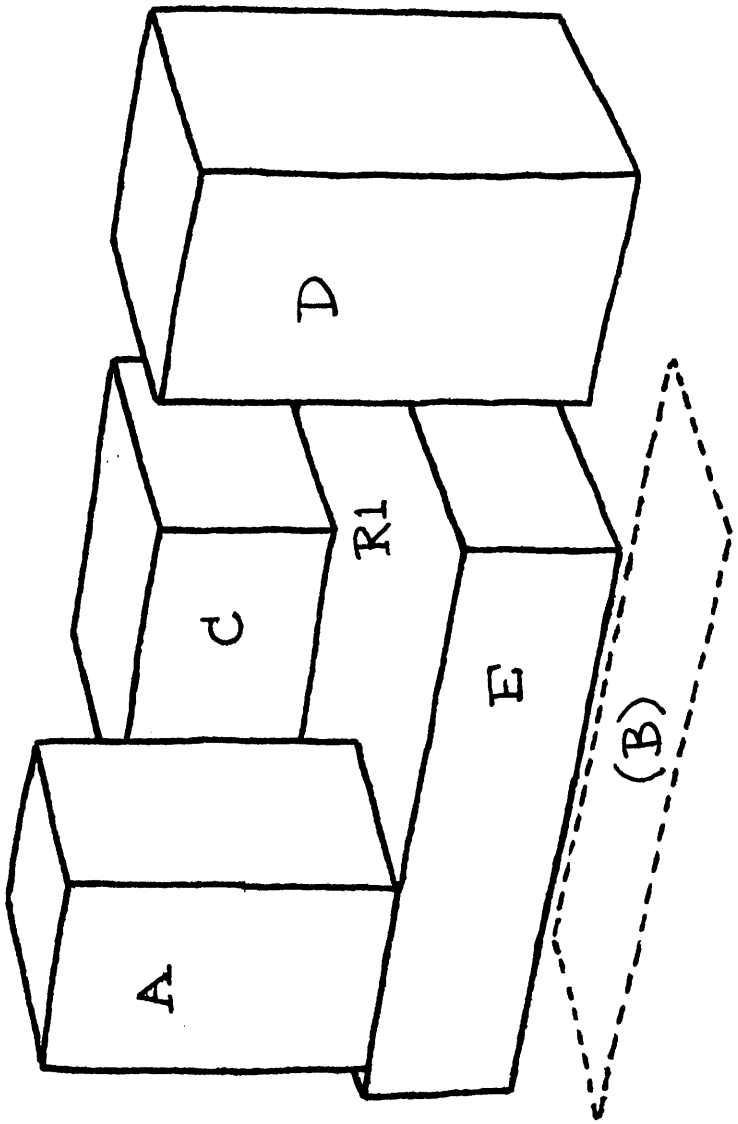


FIGURE 5.8

interpretations are possible, my suspicion is that I pick the one I expect to be true, and on the basis of this expectation I then choose a test (or tests) to eliminate all the (n-1) other possibilities. After performing this test, I then have knowledge which either supports my expectation or forces me to form or choose a new expectation.

The curious fact about my perception is that I only see one interpretation at a time even when I know that a scene is ambiguous. (Take for example the reversing illusion which alternates between a vase and two faces in profile, depending upon which regions are viewed as figures and which are viewed as background (Koffka 1935)).

Even when I have insufficient solid knowledge on which to base my interpretation of a scene, my expectation seems to carry the same force of conviction that solid knowledge would. Nonetheless, I can change my interpretations of scenes either when I am faced with new evidence (by a change in my relation to a scene or change in the scene) or if I am challenged about my interpretation (Are you sure?).

knowledge ends at this level; for some cases its understanding is sufficient to uniquely recognize a property, in other cases it is unable to select between two or more possibilities.

(3) I believe that the next levels of understanding are characterized by the ability to define a critical test (or series of critical tests) which will allow a program to eliminate remaining possibilities until only one is left. Such a test might be "if I remove the object in front, I will be able to see whether or not that region is connected to the table surface" or "if I move to the right, and if that region is part of the table, then I should be able to see an edge at point (x,y)". I claim that this must be the next level of knowledge since many line drawings simply do not contain cues which allow a program (or a person) to decide between various possibilities.

However, let me make a distinction between knowledge and expectation. Even if I am not allowed to make further tests, I still expect the scene to have a particular form. Moreover, I believe that this expectation, simulated by heuristic rules in my program, is instrumental in deciding just which critical tests I should make. For example, if n

5.5 AN EXAMPLE

I have now shown how to use selection rules to narrow down the choices for junction labels on the basis of various kinds of cues from the line drawing. To give an idea of how much these rules help, look at figure 5.9. Next to each junction I have listed the numbers of labels which are possible for it before and after applying the selection rules. I have assumed that the program knows that R0 is the support surface and that the circled numbers in each region indicate the relative brightness (the higher a number, the brighter the region). Notice that one junction, the peak on the scene/background boundary, can be uniquely labeled using only selection rules. Most of the interior junctions remain highly ambiguous.

Moreover, I am aware of ambiguity in another way; even though my own interpretation may carry a sense of conviction with it, and even though I don't usually change this interpretation without reason, I can easily understand how another person could interpret a scene in one way while at the same time I am seeing it in a different way, where I am using seeing to mean interpreting with conviction of truth.

I do not believe it is worthwhile to delve too much deeper into speculation about similarities between my system and human perception. For example, it doesn't seem to me to make much sense to try and decide whether people generate alternative interpretations when they are needed or whether (as in my program) they keep all the active alternative interpretations but are only aware of the expected one at any given time.

Nonetheless, I think that in connection with ambiguity, the notion of knowledge at "other levels" as the ability to eliminate interpretations, and the notion of expectation as the default choice of an interpretation when I run out of solid knowledge, are ideas of central importance.

6.0 THE MAIN LABELING PROGRAM

You will recall that I described at some length in Section 2.4 a "filter program" which systematically removes junction labels whenever there are no possible matches for the labels at adjacent junctions. Now that I have shown a good deal more about the junction labels and the use of the selection rules, I would like to treat this program again from a somewhat different perspective.

6.1 A SMALL EXAMPLE

Suppose that the program is working on a scene, a portion of which is shown in figure 6.1. Assume that the selection rules eliminate all labels for each type of junction except those shown at the bottom of the figure. Remember that the selection rules operate only locally, i.e. they give the same list of possibilities no matter how the labeling has proceeded or in what order the junctions are taken. All the step numbers refer to figure 6.2, which summarizes the succesive lists attached to each junction:

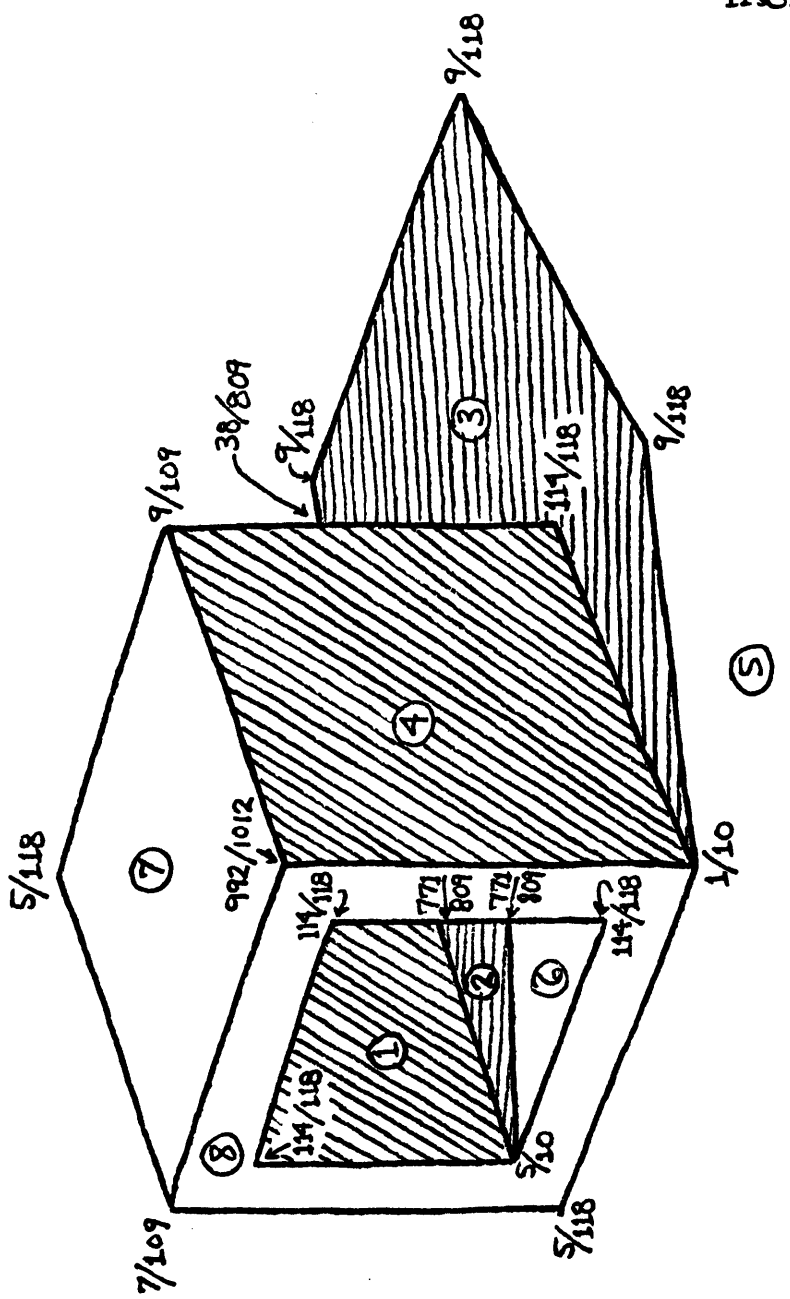
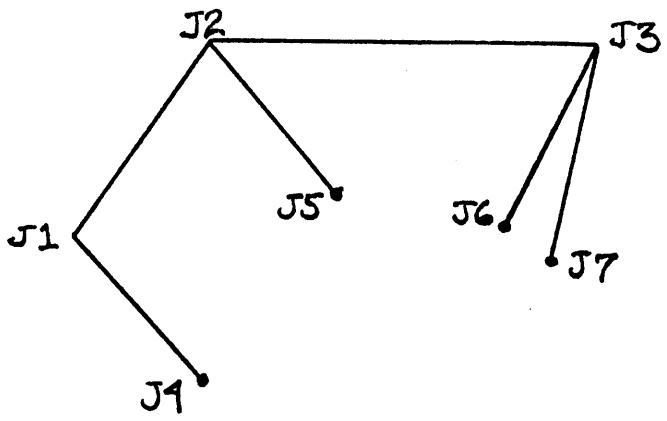

9/118
38/809
9/118
9/109
119/118
9/118
5/118
992/1012
1/10
114/118
771/809
771/809
114/118
114/118
5/10
7/109
5/118

FIGURE 59

	LABELS ASSIGNED TO		
	L1	L2	L3
START:	——	——	——
STEP 1:	——	((ABB)(ABC)(BCA)(FAD)(DBF))	——
STEP 2:	((AB)(AC)(AD)(BB)(BE)(CF)(FA)(GH)(EA)(EB))	(UNCHANGED)	——
STEP 3:	((AB)(AC)(AD)(BB)(BE)(CF)(FA))	(UNCHANGED)	——
STEP 4:	(UNCHANGED)	((ABB)(ABC)(BCA)(DBF))	——
STEP 5:	(UNCHANGED)	(UNCHANGED)	((ABA)(BCA)(GHI)(FBC)(DBF)(ABE)(DCE))
STEP 6:	(UNCHANGED)	(UNCHANGED)	((ABA)(BCA))
STEP 7:	(UNCHANGED)	((ABB)(ABC))	(UNCHANGED)
STEP 8:	((BB)(BE)(CF))	(UNCHANGED)	(UNCHANGED)
	NO MORE LABELINGS CAN BE ELIMINATED		
FINAL RESULT:	((BB)(BE)(CF))	((ABB)(ABC))	((ABA)(BCA))

TIME ↓

FIGURE 62



RESULTS OF SELECTION RULES FOR J1. (FIRST ELEMENT OF EACH LABELING REFERS TO L-J1-J2, SECOND TO L-J1-J4)

L1 = ((A B)
(A C)
(A D)
(B B)
(B E)
(C F)
(F A)
(G H)
(E A)
(E B))

RESULTS OF SELECTION RULES FOR J2. (FIRST ELEMENT REFERS TO L-J2-J3, SECOND TO L-J2-J5, THIRD TO L-J2-J1)

L2 = ((A B B)
(A B C)
(B C A)
(F A D)
(D B F))

RESULTS OF SELECTION RULES FOR J3. (FIRST ELEMENT REFERS TO L-J3-J7, SECOND TO L-J3-J6, THIRD TO L-J3-J2)

L3 = ((A B A)
(B C A)
(G H I)
(F B C)
(D B F)
(A B E)
(D C E))

FIGURE 6.1

Step 4: Now the program uses this same reasoning in the opposite direction. In what ways, if any, does the fact that J1 must be labeled from the list restrict the labels of adjacent junctions? Only J2 of the adjacent junctions has been labeled so far, so only J2 can be affected. The only labels which are possible for J2 are those elements of L2 which have as a third letter "A" or "B" or "C" or "F". Therefore, the program eliminates "(F A D)" as a possible label and L2 becomes

((A B B) (A B C) (B C A) (D B F)).

Can the program eliminate any other labels because "(F A D)" has been eliminated? No, since no other neighbors of J2 except J1 have been labeled, and the reason "(F A D)" was eliminated was because it had no counterpart at J1.

Step 5: The program now can move on to J3 and label it with L3.

Step 6: Each label for J3 must have a third letter equal to one of the first letters from a label in L2. These letter are "A", "B" and "D". Therefore the program eliminates "(G H I)", "(F B C)", "(D B F)", "(A B E)" and "(D C G)" from L3 and sets L3 to ((A B A) (B C A)).

Step 1: Suppose that the program starts with J2, and that all of the other junctions are unlabeled. Then the program assigns list L2 to J2, and since all the other junctions are unlabeled, it has no basis on which to eliminate any of the labels in L2. As far as the program knows, all of these labelings are still possible.

Step 2: Now suppose that it next labels J1 by attaching to it the list L1. When it checks the junctions adjacent to J1 it now can see that J2 has already been labeled.

Step 3: Therefore the program looks at J2 to find what restrictions, if any, have already been placed on line segment L-J1-J2. In this case, the restrictions are that L-J1-J2 must be labeled with either "B" or "C" or "A" or D" or "F", i.e. with any letter which appears third in an element of L2. Each element of L1 which does not have "B", "C", "A", "D", or "F" as its first letter can then be eliminated. Therefore the program drops "(G H)", "(E A)" and "(E B)" as possibilities and L1 becomes

((A B) (A C) (A D) (B B) (B E) (C F) (F A)).

(1) attaching labels,

(2) removing any of these labels which are impossible given the current context of this junction, and

(3) iteratively removing labelings from the context by allowing the new restrictions embodied in the list of labels for the junction to propagate outward from the junction until no more changes in the context can be made.

There are two points of importance:

(1) The solution the program finds is the same no matter where it begins in the scene, and

(2) the program is guaranteed to be finished after one pass through the junctions, where it performs the three actions listed above at each junction.

Given a line drawing with N junctions, a data base which has no more than M possible labelings for any junction, and a situation where any number of junctions from 0 to N have already been labeled, let condition C be one where for each possible line label which can be assigned to a line segment either

(1) there is at least one matching line label assigned

Step 7: What labels now are possible for J2? Since the only remaining labels for J3 both set L-J2-J3 to "A", the program eliminates "(B C A)" and "(D B F)" from L2 so that L2 becomes ((A B B) (A B C)).

Step 8: This time, a neighbor of J2, namely J1, has been labeled already, so the program must check to see whether eliminating the element of L2 has placed further restrictions on L1. Only elements of L1 which have a first letter "B" or "C" are possible labels now, so the program eliminates "(A B)", "(A C)", "(A D)", and "(F A)". L1 thus becomes ((B B) (B E) (C F)).

Since no other neighbors of J1 are labeled, the effects of this change cannnot propagate any further.

6.2 DISCUSSION

I think it is easiest to view the process of the program at each junction as having three actions:

possible that the set of labels for J can be reduced further because neighbor J2 has no match for one or more labels still attached to J. The program would then have to go back to line L-J-J1 again to see whether more labels could be eliminated from J1. By considering the effects of each of J's neighbors on J's labels first, the program guarantees that as many labels as possible have been eliminated from J's label list before using this list to recompute the lists for J's neighbors.

Condition C can now only be untrue along line segments joining J with its neighbors and, moreover, can only be untrue in one direction, i.e. J's neighbors may have unmatched labels, but not vice-versa. When the program eliminates the unmatched labels from each of J's neighbors, C is now satisfied on each line segment joining J to its neighbors and C can only be unsatisfied along the line segments joining J's neighbors with the neighbors of J's neighbors, and again only in an "outward" direction, i.e. the junctions two line segments away from J can have unmatched labels, but all those junctions one line segment away (J's neighbors) cannot have unmatched labels.

to the junction at the other end of this line segment, or else

(2) the junction at the other end of the line segment has not been labeled.

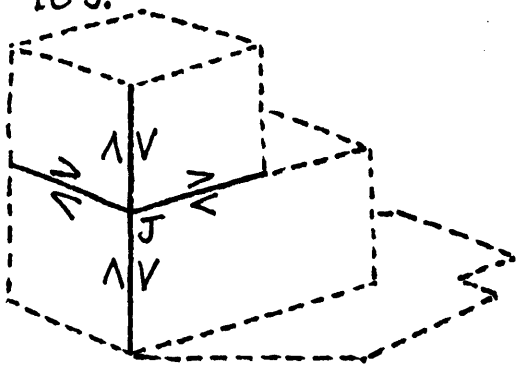
This condition C must be satisfied before the program moves on to a new junction; the program keeps track of the line segments on which the condition may not be satisfied.

When the program begins labeling a junction J, assume that C holds throughout the line drawing. When the junction, previously unlabeled, has labels added, the only line segments along which C can be violated are the line segments which join J to its neighbors, and it is possible for C to be unsatisfied in both directions on these segments (i.e. both J and J's neighbors may have unmatched line labels). Therefore, to make sure that the program needs to consider each line segment a minimum number of times, the program first uses the lists of possible labels specified by J's neighbors to eliminate all impossible labels from J.

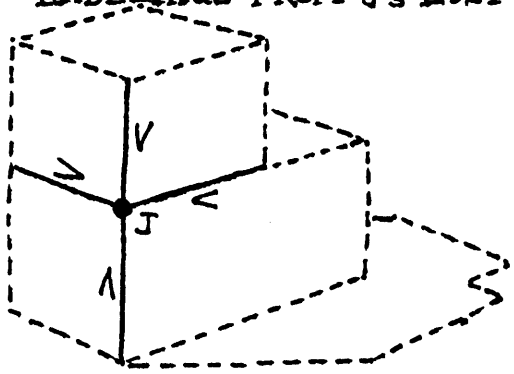
To see why this is the correct way to proceed, suppose that the program used J's initial set of labels to eliminate some labels from one of J's neighbors, J1. It is then

• INDICATES A JUNCTION WHICH HAS LABELINGS REMOVED AT THAT STEP

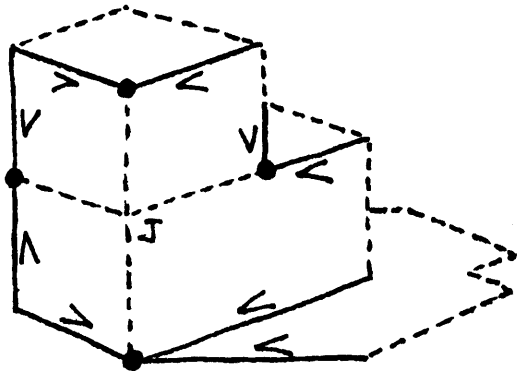
① ATTACH LABELINGS LIST TO J:



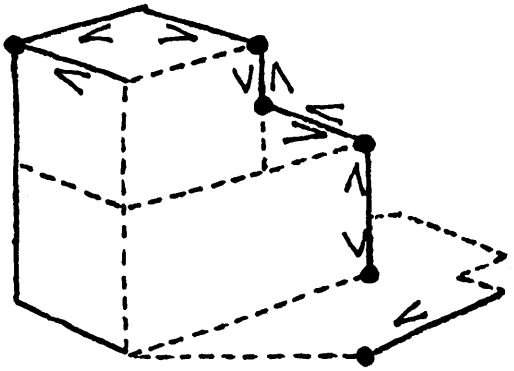
② ELIMINATE UNMATCHED LABELINGS FROM J'S LIST:



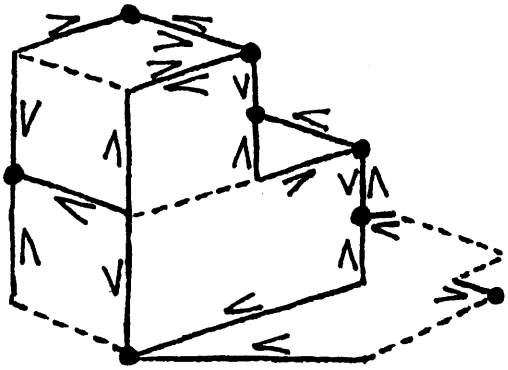
③ ELIMINE UNMATCHED LABELINGS FROM LISTS ATTACHED TO J'S NEIGHBORS:



④ ELIMINATE UNMATCHED LABELINGS FROM LISTS ATTACHED TO J'S NEIGHBORS' NEIGHBORS:



⑤ ELIMINATE UNMATCHED LABELINGS FROM LISTS ATTACHED TO J'S NEIGHBORS' NEIGHBORS' NEIGHBORS:



⑥ ETC:

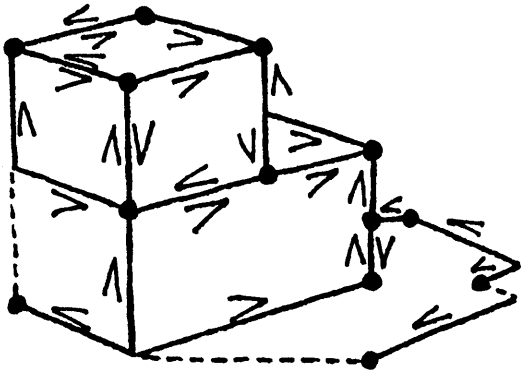
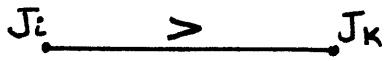


FIGURE 6.3

The line segments on which C does not hold continue to spread outward to the neighbors of junctions two segments away from J, then junctions three segments away from J, etc., but only as long as labels are being removed from any junctions. As soon as the program reaches a step where no labels are removed from any junction, then the program knows that condition C must be satisfied everywhere in the scene, and it can move on to the next unlabeled junction.

Figure 6.3 traces a situation which could occur on successive steps in a line drawing where all junctions except J have been labeled already. I have filled in the line segments along which condition C could be violated at each stage of the program's iterations. The mark ">" indicates which junction can have unmatched labels; it is used like the same sign meaning "greater than", so that you can read



as "the number of labels at Ji is greater than the number of labels at Jk", i.e. Ji may have labels which are not matched by ones at Jk.

THIS FIGURE IS EXACTLY THE SAME AS EXCEPT THAT J1 IS EITHER UNLABELED OR J1 HAS ALREADY BEEN LABELED UNIQUELY, SO J1 CAN NEVER HAVE MORE LABELINGS THAN J2, J3 OR J4.

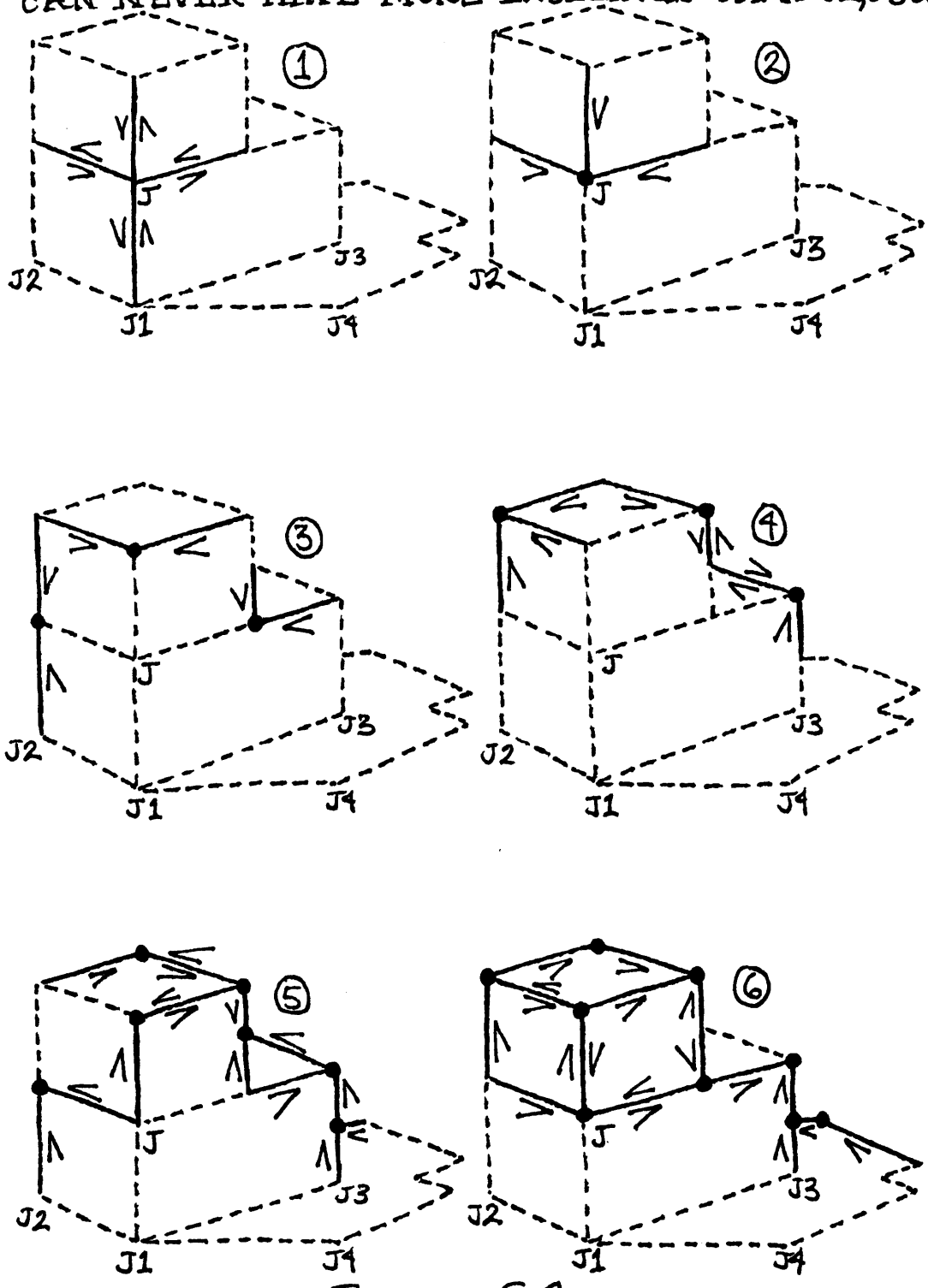


FIGURE 6.9

The violations of C can spread outward to eventually touch any line segment of a line drawing, but only if the number of labels can be reduced at each junction on some path between the junction the program is currently labeling and the line segment. If any of the junctions in Figure 6.3 were unlabeled or if a unique label had already been found for the junction, then no violations of C could propagate through that junction.

Figure 6.4 represents just such a situation. The line drawing is assumed to be completely labeled except for junction J, but this time J1 already has been uniquely labeled. Thus it can never be the case that J1 has unmatched labels. Notice that Figure 6.4 also represents equally well the case where J1 has not yet been labeled.

One final point: the process is guaranteed to terminate, since if there are N junctions and no more than M labels possible for any one junction, the process can never go on for more than M x N steps at the very worst. This is important since the restrictions can propagate back to the junction which initiated the process. To see that the possibility of cycles does not create any difficulties, consider the following trick. Suppose that as soon as the

(2) next label all junctions which bound regions that share an edge or junction with the background.

To see why the program is faster when it eliminates as many possibilities as early as it can, I must first give some idea about the amounts of computation needed for various phases of the program. The basic operation involves removing unmatched labels from junction lists. The removal is done in the following manner:

Assume that the junction whose list of labels must be reduced is called J2, that its neighbor is J1, and that for any label in the lists of either J1 or J2, the first line label represents the line joining them. Thus if (A B C) is one possible junction labeling in J1's list, then "A" is the line label that this junction labeling would assign to line L-J1-J2, and similarly, if (D E F) is a labeling from J2's list, the "D" is the line label which refers to L-J2-J1.

Since J2's list is the one to be reduced, first look at J1's label list and make a list which consists of the labels which J1 can apply to L-J2-J1. Notice that I have up to now glossed over the fact that for most lines, the label appears different depending on which end of the line we choose as a

starting junction has been checked against each of its neighbors, that all the remaining labels are removed from it. The restrictions can then spread outward only until no more changes can be made; now look at the process as though the junction were being labeled for the first time with the set of junctions just removed as its starting junction set. This process can then be repeated as often as necessary, but the number of times can never be greater than the initial number of labelings assigned to the junction, since the process terminates if no more labels can be removed from the list of possibilities.

6.3 CONTROL STRUCTURE

While the program can start at any junction and still arrive at the same solution, the amount of time required to understand a scene does depend on the order in which the junctions are labeled. The basic heuristic for speeding up the program is to eliminate as many possibilities as early as possible. Two techniques which help accomplish this end are to

(1) label all the junctions on the scene/background boundary first, since these have many fewer interpretations that interior junctions do, and

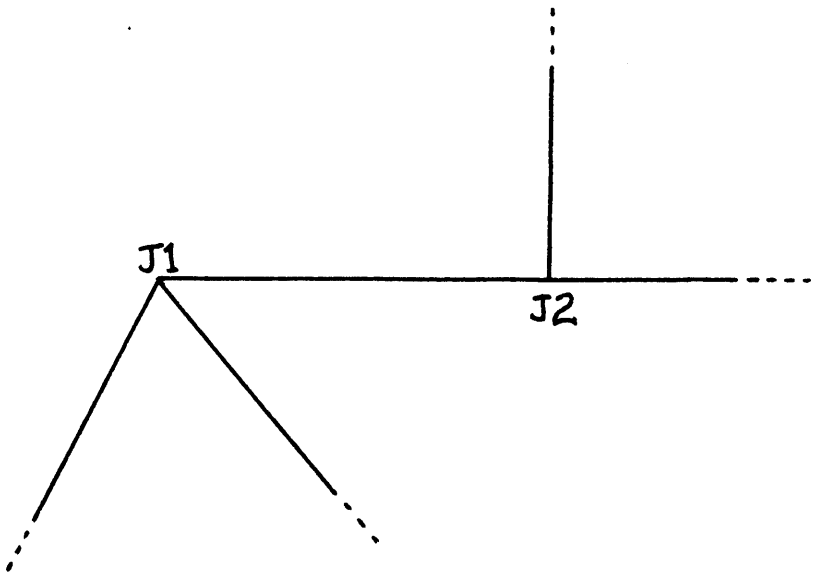
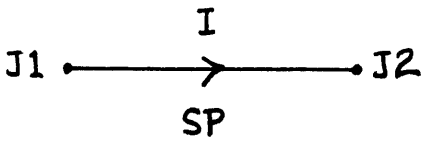


FIGURE 6.5

reference point. Thus if line L-J1-J2 is labeled



then from J1's end it appears to be labeled as "OCR-ISP" and from J2's end it appears to be labeled as "OCL-SPI" (for OClude Right-Illuminated/Shadow-Projected and OCLude Left-Shadow-Projected/Illuminated respectively). Therefore what we really want is the list of the opposites of the first elements of each label for J1. Suppose that I am given the scene portion shown in figure 6.5. If J1's list of labelings is:

```
((OCR-II PLUS-II OCL-II)
 (OCR-ISP PLUS-SPI OCL-II)
 (OCRM-II PLUS-II OCLM-II)
 (SHCLR-ISP OCR-SPI OCLM-II))
```

Then the list that I need to compare J2's labels to is:

```
L1 = ((opposite (OCR-II))
      (opposite (OCR-ISP))
      (opposite (OCRM-II))
      (opposite (SHCLR-ISP)))

   = (OCL-II
      OCL-SPI
      OCLM-II
      SHCCR-SPI)
```

branches to see if any labels for adjacent junctions can be removed. Thus it must compute m lists analogous to L1 above and each of these lists has n members. Now when each of these lists is compared to the label lists for adjacent junctions, the program must make an average of n/2 tests for equality for each labeling that is retained, and n tests for equality for each labeling that is removed (for the case where it looks through an entire list and finds no match). Therefore for each portion of the process the amount of computation involved is at least proportional to n.

Because the amount of computation is at least proportional to n, it is undesirable to label interior junctions first, since most of these have much larger initial values for n than do scene/background boundary junctions. Not only does it take more computation to propagate any reductions through these junctions, but each reduction is likely to be smaller as well; if two adjacent junctions can each be labeled in n ways out of a total of N theoretically possible ways, then the expected number of labelings they have in common is n^2/N. (This number is obtained by summing the probability of a match for each of the n labels at one junction; thus

J2's label list can then be compared to L1; the condition which must be satisfied by a labeling of J2 in order for it to be a possible labeling is that the line label it would assign to L-J1-21 be a member of the list L1. Continuing with this example, suppose that J2's labeling list is:

```
L2 = ((OCL-II OCR-II OCR-II)
      (OCL-ISP OCR-SPI OCR-II)
      (OCL-ISS OCR-SSI OCR-II)
      (OCL-SPI OCR-II OCR-ISP)
      (OCL-SPSP OCR-SPI OCR-ISP)
      (OCL-SPSS OCR-II OCRM-II)
      (OCL-II OCR-II OCRM-II)
      (OCRM-II OCRCR-II OCML-II))
```

Then the labeling list for J2 after comparing L2 to L1 is:

```
L2'= ((OCL-II OCR-II OCR-II)
      (OCL-SPI OCR-II OCR-ISP)
      (OCL-II OCR-II OCRM-II))
```

Now I return to the original claim, that it is desirable to remove as many labels as possible as early as possible. Suppose a junction J has m+1 branches and n+q labels, and in the process of labeling, q of these labels are eliminated by a propagating reduction which comes in on one of J's branches: this requires the program to compare n+q labels for members in a list. The program now has to check each of J's

I included this section so that an interested reader could get a better feeling for the operation of the program and also to suggest some ideas for extensions of this program. For example, if my labeling program were connected to a line-finding program such as Shirai's, my program could be adapted to provide intelligent guidance for deciding where to look next in a scene on the basis of which features had already been found (Shirai 1972).

Another idea which might be interesting to follow up is a possible parallel between the reasons why it is better for my program to start on the scene/background boundary and the observed fact that people presented with a figure on a background for short periods of time see detail first on the figure/ground boundary and require longer viewing durations to see details in the figure interior suggesting that our perception proceeds from the outside inward (Koffka 1935).

I mentioned at the beginning of this paper that the amount of time (and therefore computation) is roughly proportional to the number of line segments in a scene. This may not seem to fit with the obvious fact that there is really nothing to prevent the effects caused by labeling a single junction to propagate to every portion of a line

$$\sum_{i=1,2,\ldots}^{i=n} (n/N) = n \times (n/N) = n^2/N.)$$

Typically scene/background boundary junctions have about 1/10 the number of possible labels an interior junction can have, so that the expected number of labelings to scene/background junctions will have in common is only 1% of the expected number for two interior junctions. Similarly, it is worthwhile to label next interior junctions which are connected to junctions on the scene/background boundary, since the expected number of labelings in common for these pairs is only 10% of the number for interior junctions. Finally, as I mentioned earlier, it is worthwhile to label all the junctions surrounding regions which touch the scene/background boundary, since these regions contain all the "best" kinds of junctions, and because a chain of junctions which closes on itself tends to be far more restricted in its possibilities than a chain of the same length which does not. (I will not attempt to prove that this is so; I think it is fairly obvious that the effect is true, although the proof of the effect is not. It is much more obvious for a tree search procedure than for this one.)

6.4 PROGRAM PERFORMANCE

The program portions I have now described are adequate for labeling scenes without accidental alignments, non-trihedral vertices or missing lines. Within this range there are still certain types of features which confuse the program, but before showing its limits, I will show some of its complete successes. In all the scenes that follow, I assume that the program knows which region is the background region, and that it also knows the relative brightness of various regions. The program operates nearly as well without these facts but not as rapidly. Figure 6.6 shows a number of scenes for which the program produces unique labelings or is only confused about the illumination type of one or two regions (as in figure 6.6D and 6.6I). By varying someof the region brightness values or omitting them, the program could also be similarly confused in this way for the tops of objects in figures 6.6A, 6.6B, 6.6E, 6.6G and 6.6H. In general, the program is not particularly good at finding the illumination types for regions unless the regions are bounded by concave edges. This confusion has a physical basis as well. In all the diagrams I have drawn these top surfaces as though they were parallel to the table so that the should all

drawing.

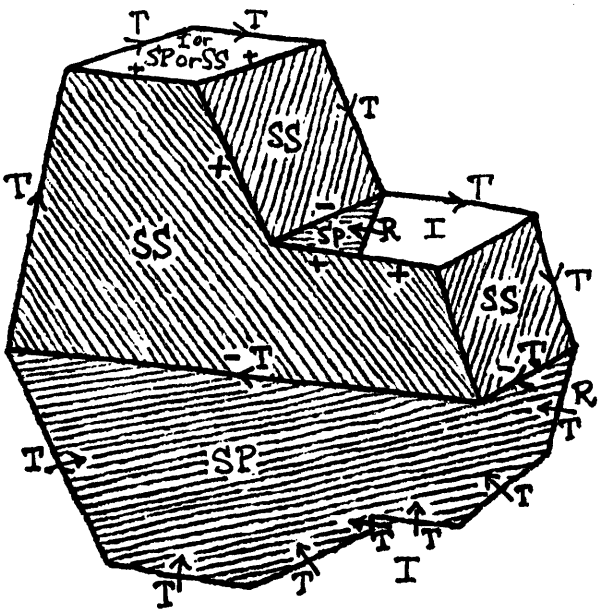
There are good physical reasons why this seldom happens. The basic reason is that some junctions simply do not propagate effects to all their neighbors, and so the effects tend to die out before getting too far. The prime type of junction which stifles the spreading effects is the T junction.

In most T junctions, the labelings of the upright and crossbar portions are independent. Even if we know the exact labeling of the crossbar portion we are unlikely to be able to draw any conclusions about the labeling of the upright and vice-versa. Since objects are most commonly separated by T junctions, the effects of labeling a junction are for the most part limited to the object of which the junction is a part, and to the object's shadow edges, if any.

Another reason why effects do not propagate far is that when junctions are unlabeled or when they are uniquely labeled, they do not propagate effects at all. (This reason was illustrated in figure 6.4.) Thus when few junctions are labeled and when most of the junctions are labeled the effects of adding restrictions tends to be localized.

FIGURE 6.6

D:


T
T
I or SP or SS
+
+
SS
T
+
T
SS
SP
R
I
+
+
SS
T
-T
-T
R
T
T
SP
T
T
T
T
I
T
T

E:

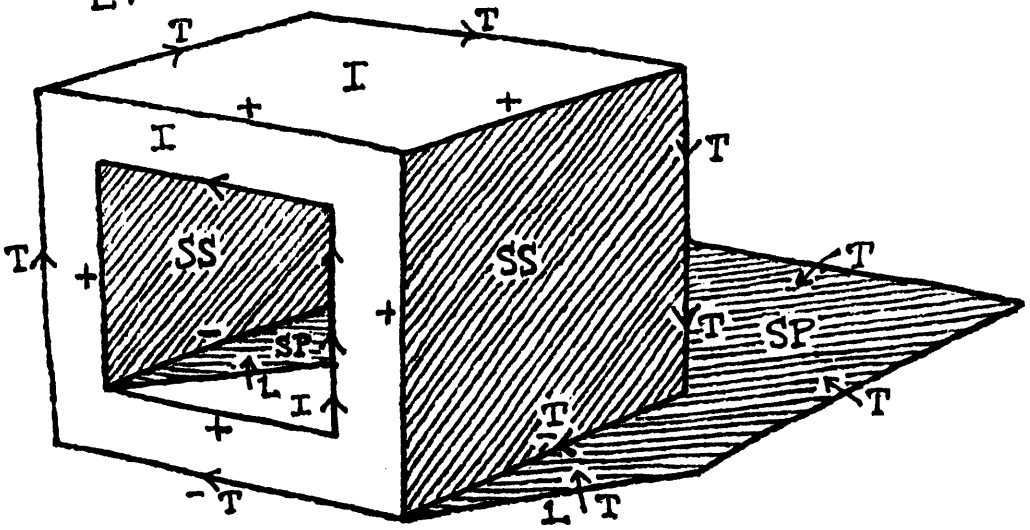
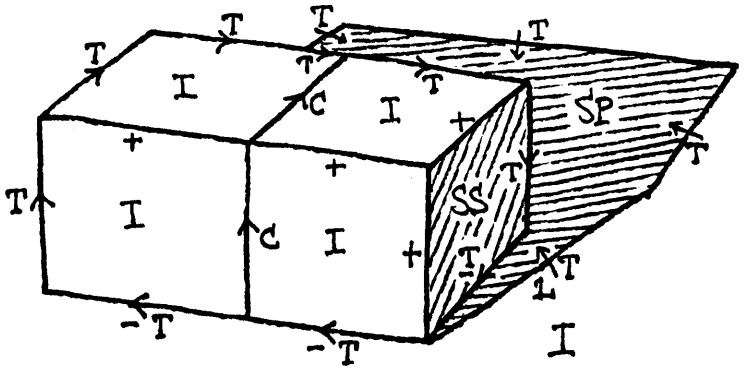
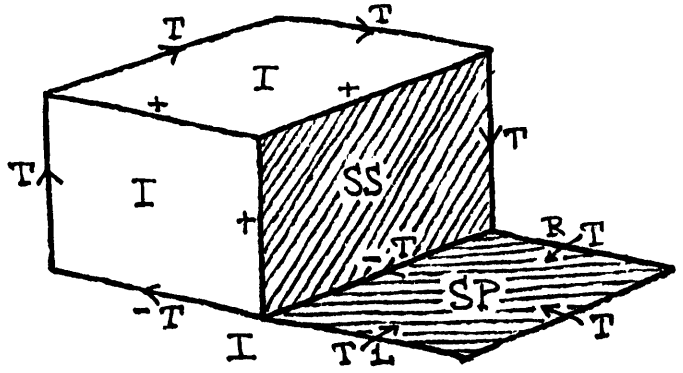

T
T
I
+
+
I
T
T
SS
SS
+
+
T
T
SP
SP
L
I
+
T
T
-T
L
T

FIGURE 6.6

A:



B:



C:

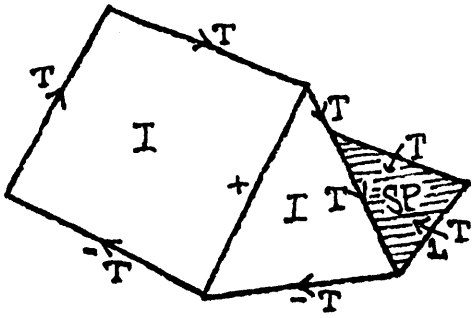


FIGURE 6.6

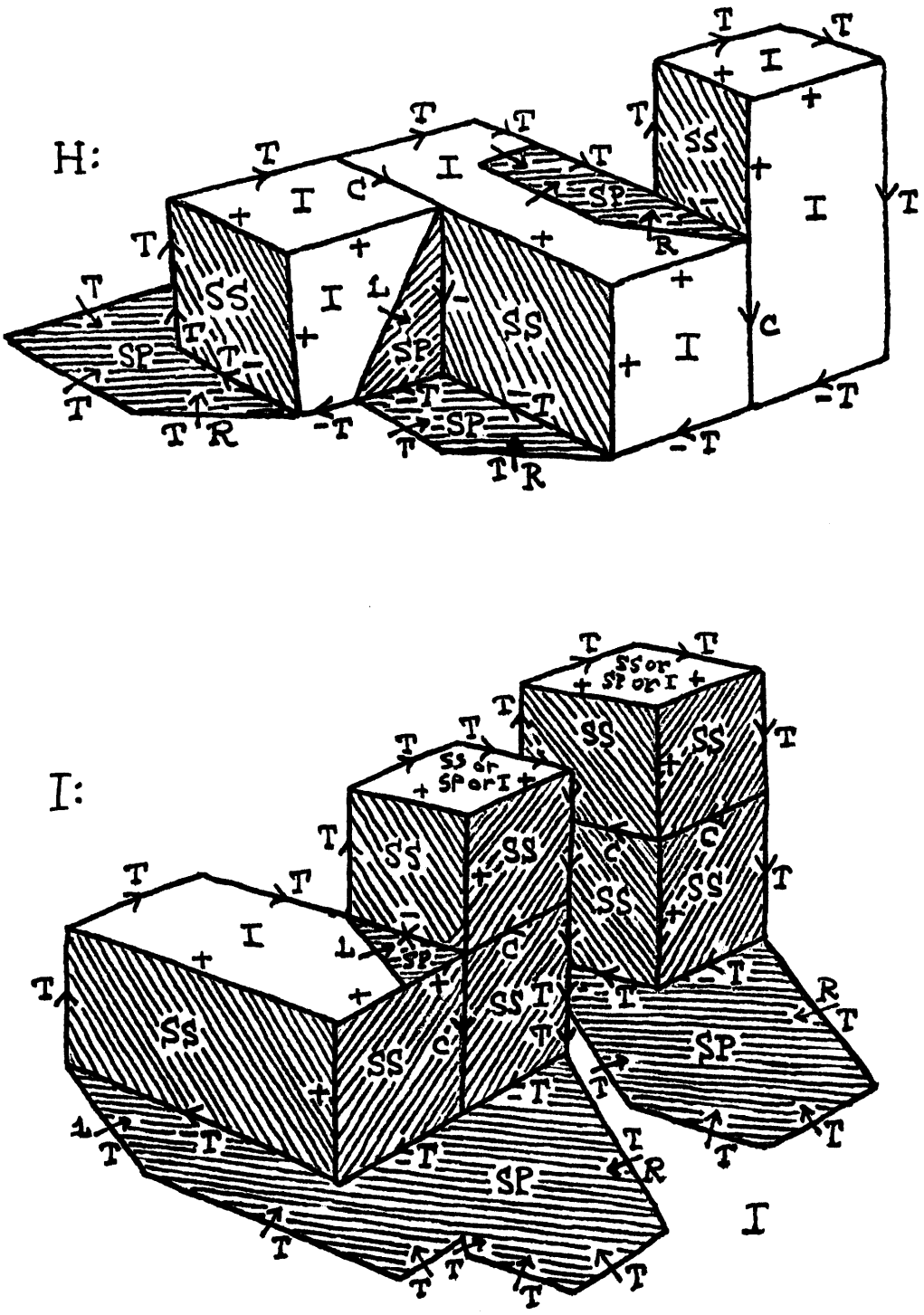
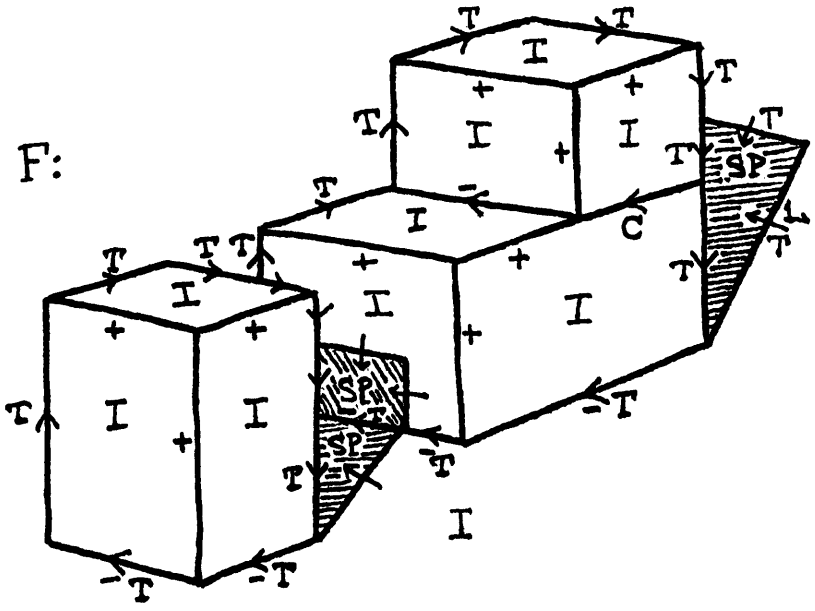
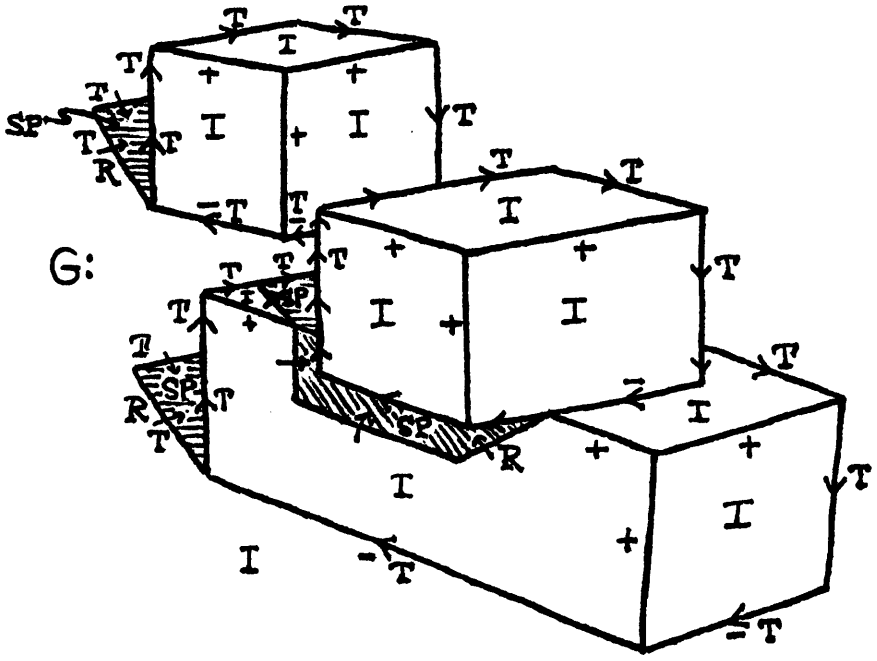


FIGURE 6.6

F:



G:



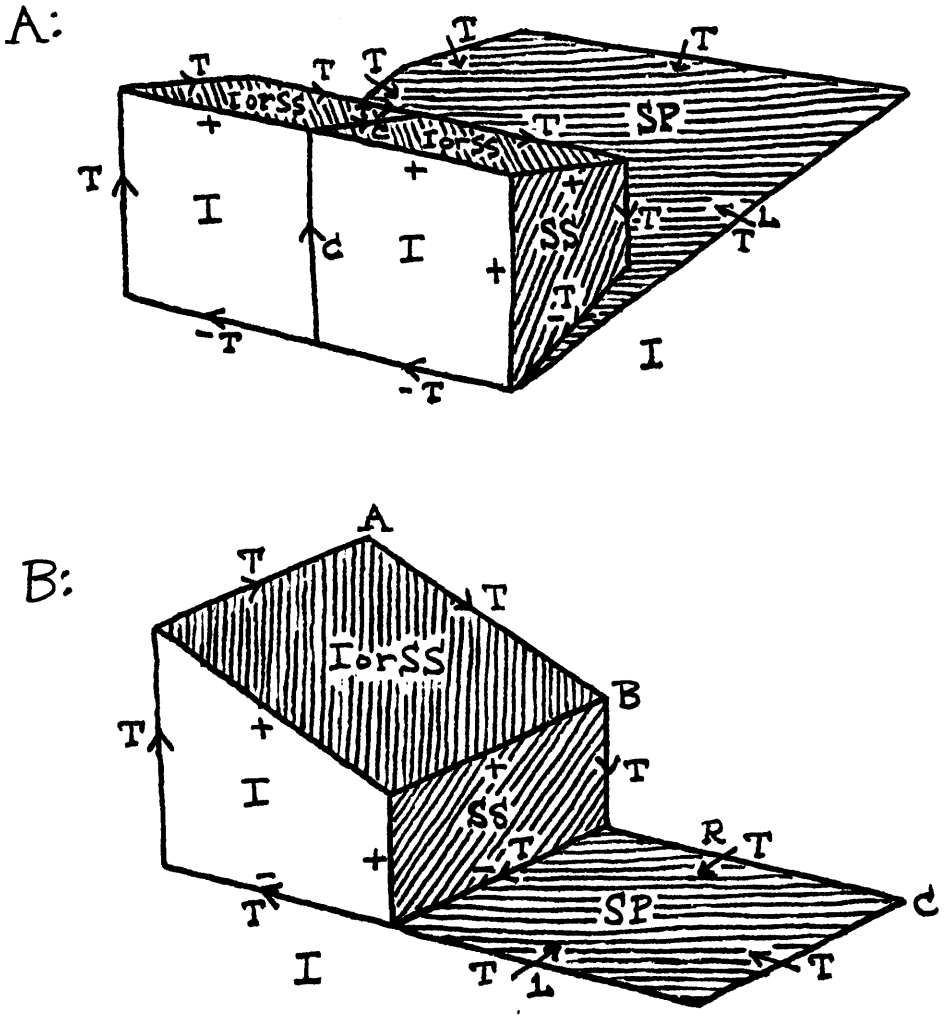


FIGURE 6.7

be labeled as type I (Illuminated), but since the program I have described so far uses only the topology of a line drawing, it has no way of distinguishing the scenes I have drawn from others which should be labeled differently. For example, in figure 6.7 I have redrawn figures 6.6A and 6.6B so that the top surfaces are type SS (Self-Shadowed), but the figures are topologically identical.

To decide whether a surface is self-shadowed or illuminated, one must be able to associate shadow corners with the vertices which cause them. In figure 6.7B, if C is caused by B, then the top of the block is illuminated, and if C is caused by A then the top of the block is self-shadowed. To verify that A causes C, place a straight edge on the figure. There is an interesting optical illusion in this figure; it appears to me that the top surface of the block in figure 6.8 should be type SS, but in fact if you use the straight-edge test I described, you will find that it is actually illuminated. (I did not put in any shading, to prevent biasing the choice.)

In any case, I think that the issue here is not serious, since the program still finds the correct edge labels for all edges. In general I doubt that anyone will be too interested

in finding the illumination values exactly; in the program they serve primarily as labeling aids, not as ends in themselves. However, before going on to something else, I would like to use this topic to illustrate a situation I have encountered several times in the process of performing this research. I noticed early in my study of scenes that if all shadow corners and their causing vertices in a given scene are connected by straight lines, these lines have roughly the same slope throughout the scene, provided that the light source is reasonably far away from the scene compared to the scene size. I thought that this fact might aid me a great deal in finding shadows. What I did not see was that until I could locate shadows and their causing vertices, I couldn't connect the two to find the characteristic slope; but if I could find the shadows and vertices, then I knew how to solve the problem already, and so I would not need to find this slope at all! There is at least one type of case where this slope is important, as I describe in the next section, but for the most part the topology of scenes provides adequate clues for finding shadows.

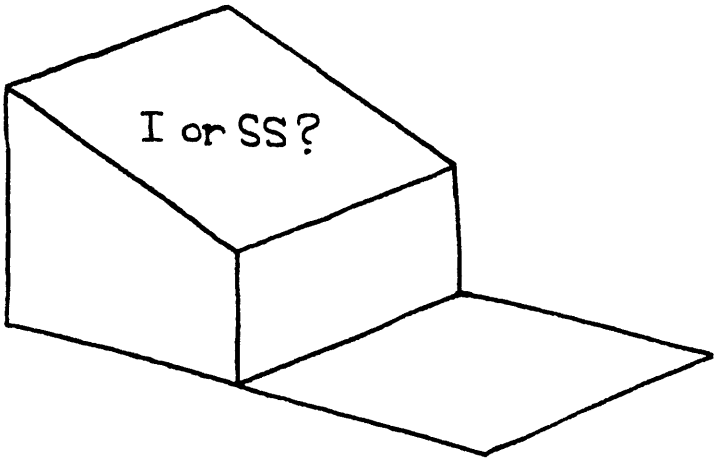
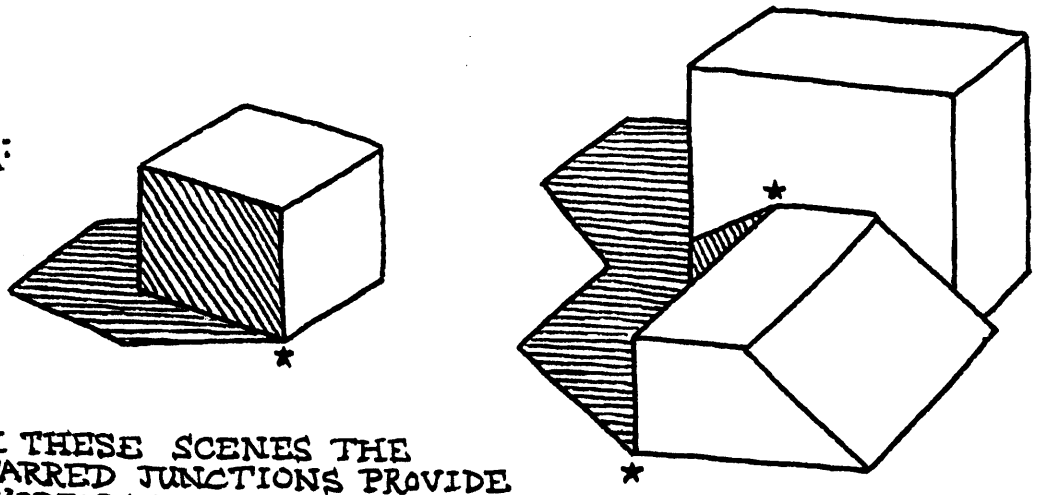


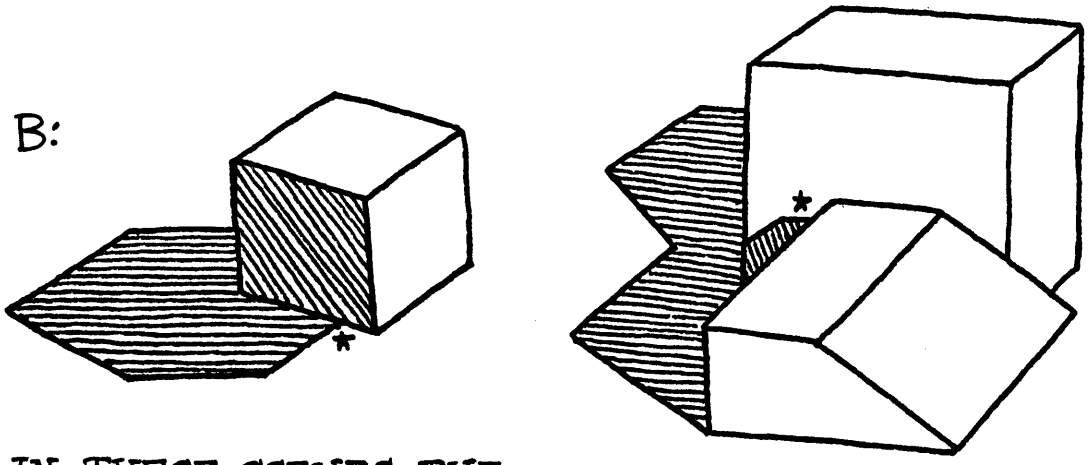
FIGURE 6.8

A:



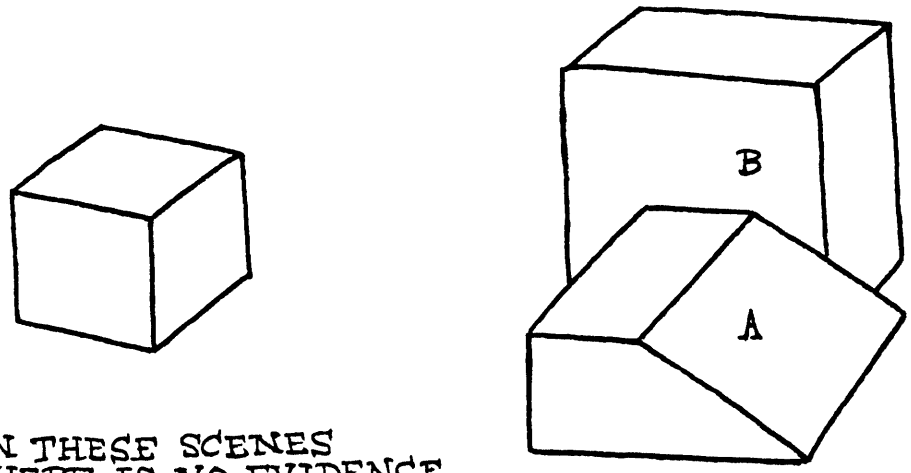
IN THESE SCENES THE
STARRED JUNCTIONS PROVIDE
EVIDENCE THE TWO OBJECTS
OR THE OBJECT AND TABLE TOUCH.

B:



IN THESE SCENES THE
STARRED JUNCTIONS PROVIDE
EVIDENCE THAT THE TWO OBJECTS
OR THE OBJECT AND TABLE DO NOT TOUCH.

FIGURE 6.9



IN THESE SCENES
THERE IS NO EVIDENCE
TO USE TO RELATE THE
OBJECTS TO EACH OTHER
OR TO THE TABLE; IT IS
NOT POSSIBLE TO DECIDE
WHETHER THEY TOUCH OR NOT.

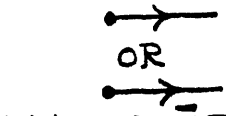
FIGURE 6.9

In figure 6.10A, each of the segments marked with a star can be interpreted either as an obscuring edge or as a concave edge, though in most cases choosing one or the other for some line segment forces other segments to be interpreted uniquely, as shown in figures 6.10B and 6.10C.

As in the previous section, there are scenes which are topologically identical which can help to show why the program finds all these labelings as reasonable interpretations. Figure 6.11 shows five scenes which are topologically identical to the labeled scenes shown in Figure 6.10C; in each of these scenes, the labeling shown seems to me to be the most reasonable one or at least a plausible one.

Figure 6.12 shows the next problem case. Such a case occurs when we can see only enough of an object so that it is not possible to tell whether the region is a shadow or an object. If it happens that the ambiguous region is brighter than the background (or what would be the illuminated portion of a partly shadowed surface of the feature occurs on the interior of a scene), then the program can eliminate the possibility that the region is a shadow. Unfortunately, if the ambiguous region is darker than its neighbor, it cannot tell whether the region is a shadow region or a dark object.

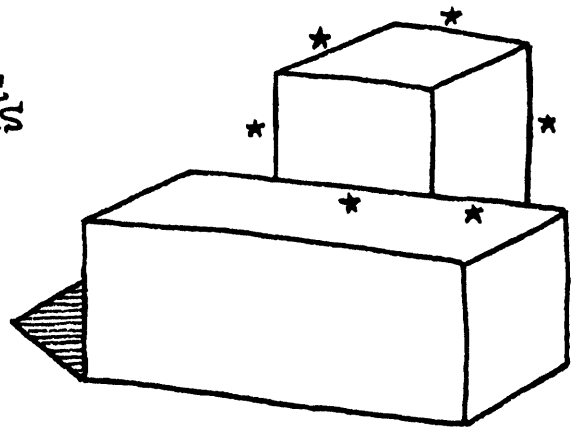
EACH STARRED LINE CAN BE LABELED AS



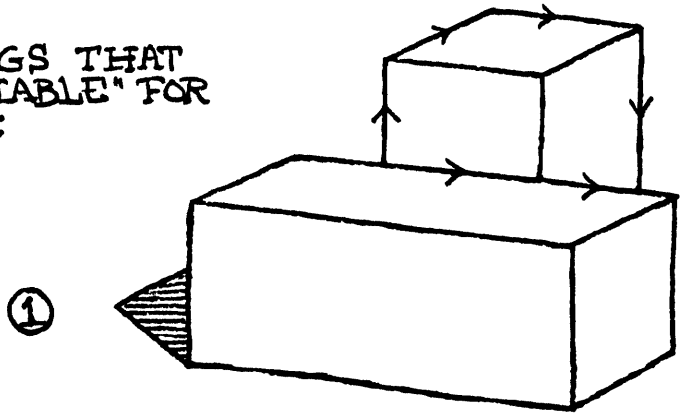
OR

.

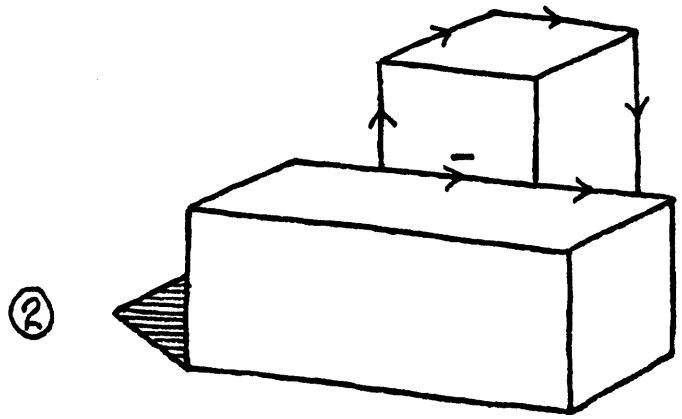
ALL UNMARKED LINES ARE LABELED UNIQUELY.



TWO LABELINGS THAT SEEM "REASONABLE" FOR THIS SCENE:



①



②

FIGURE 6.10

ALL THE REMAINING "UNREASONABLE" LABELINGS

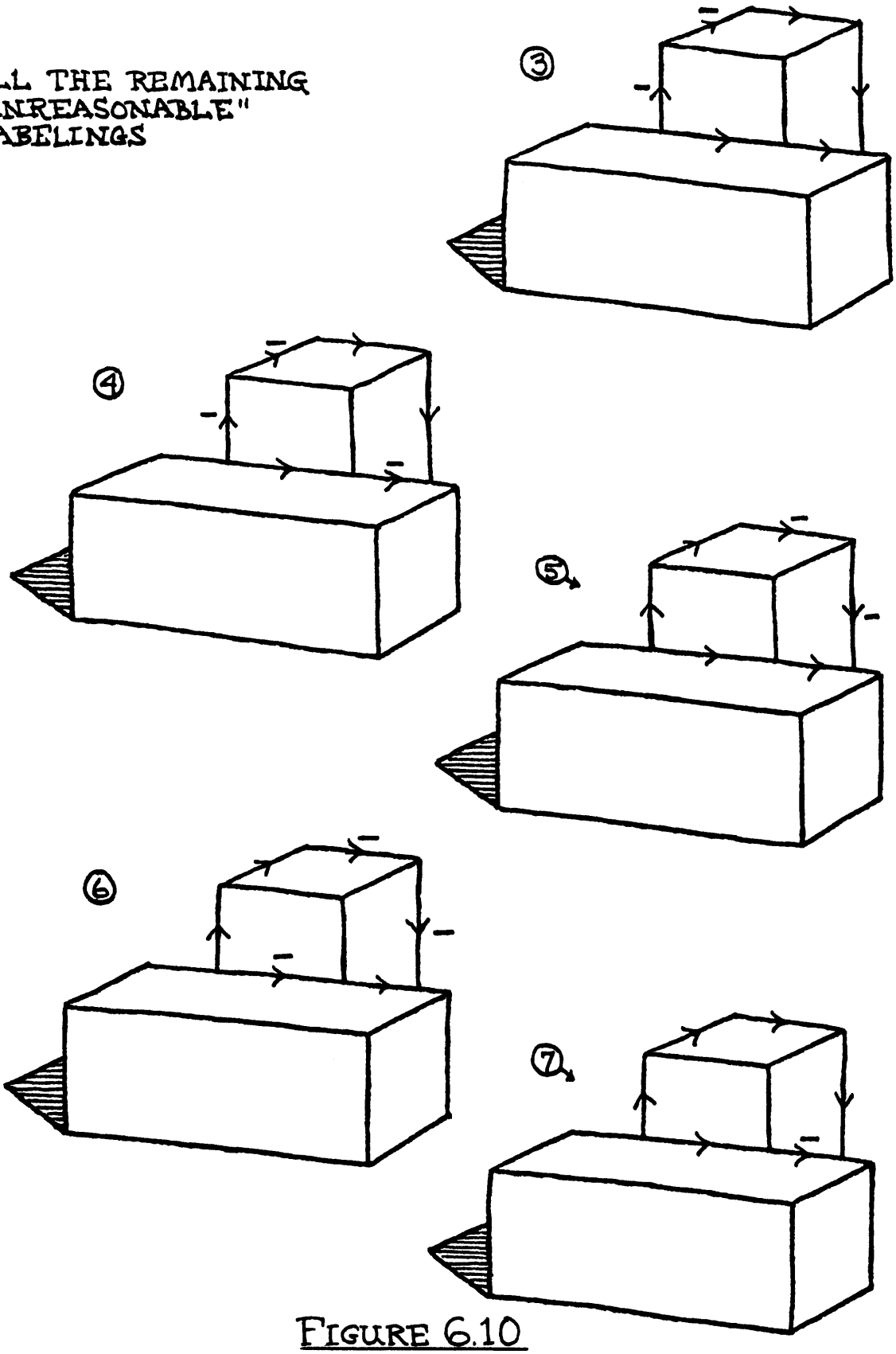


FIGURE 6.10

THE NUMBERS CORRESPOND TO THOSE IN FIGURE 6.10C.

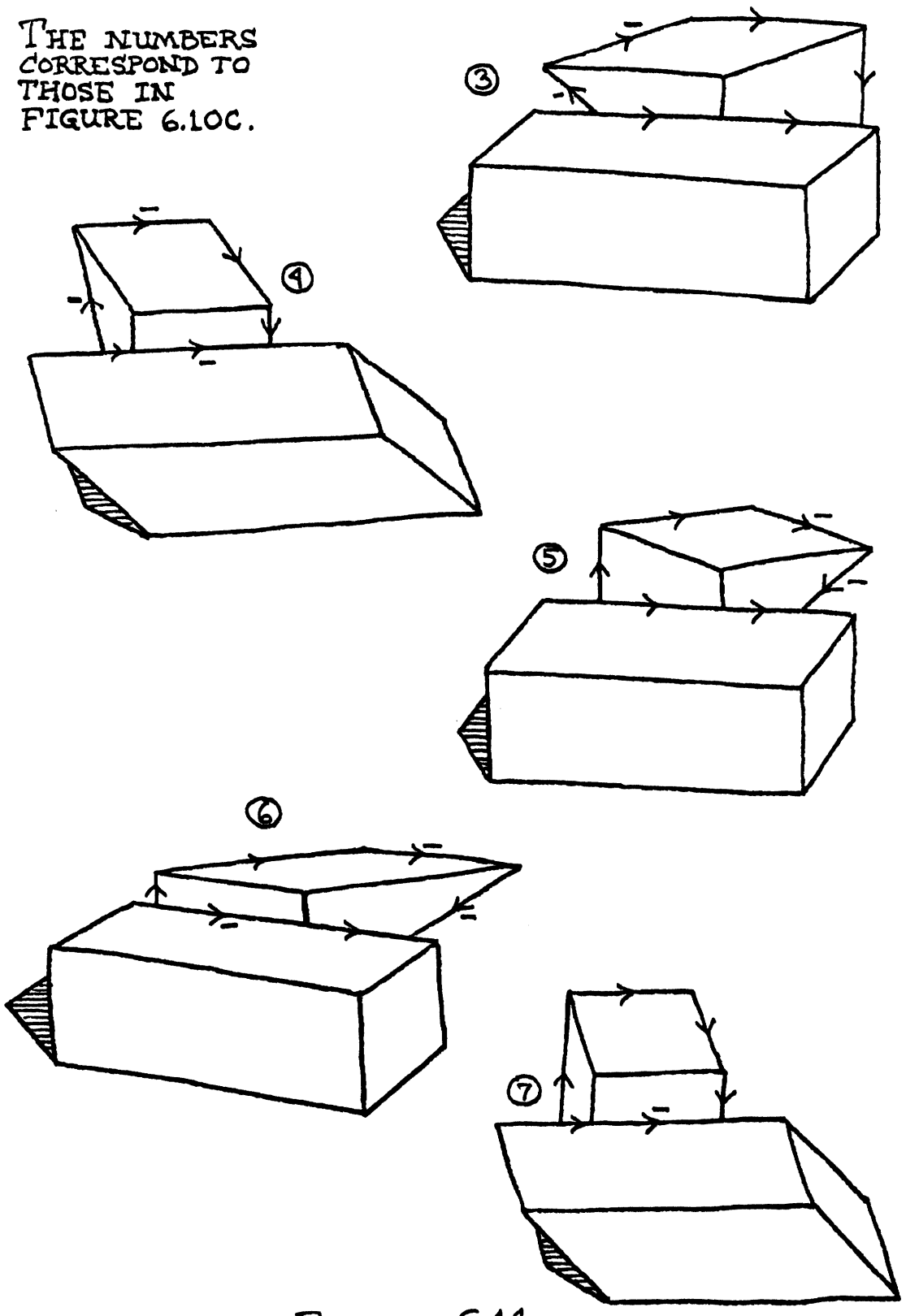
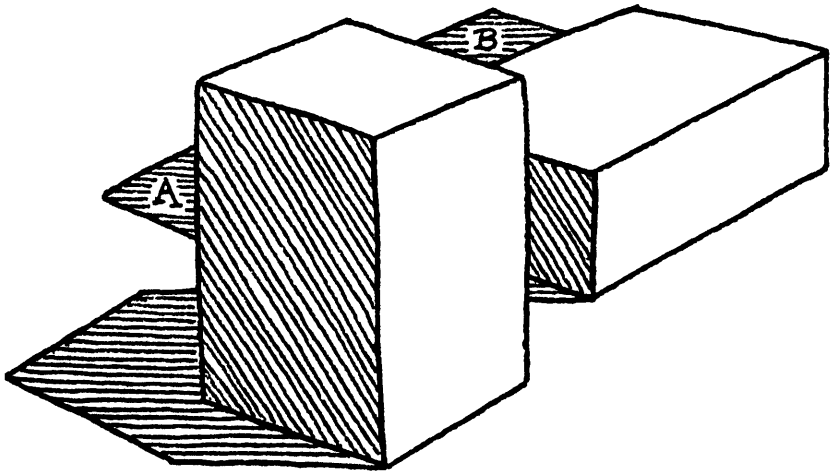


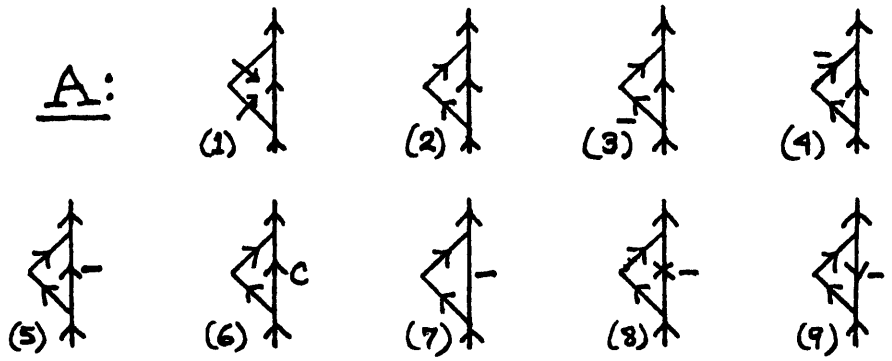
FIGURE 6.11

FIGURE 6.12

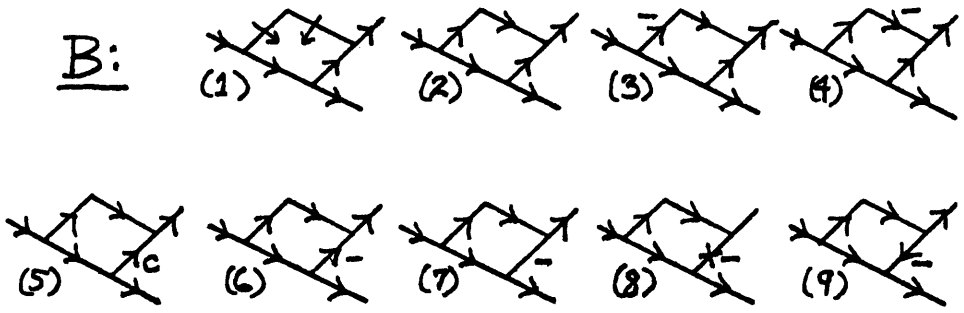


ACCORDING TO THE PROGRAM A AND B CAN BE LABELED AS FOLLOWS:

A:



B:



In figure 6.12, do you think that both A and B should really be labeled as shadow regions? In fact neither A nor B can be shadows! You can prove this for yourself by finding the characteristic light source slope for the scene, using the front object and its shadow. Then note that there can be no hidden objects which could project A or B. Figure 6.13 shows this construction. It is this type of distinction for which the light source slope information could be useful.

I will not go through the process again of showing how each of the labelings could arise. Clearly the interpretation of A and B as shadows is reasonable for this scene, since I can easily find a topologically equivalent line drawing where some obscured objects could cause the shadows. The program needs to know about gravity, support and line segment directions in order to eliminate some of the interpretations of region A. Every one of the interpretations is possible for B.

A closely related ambiguity is illustrated in figure 6.14A. Again difficulties arise because a shadow-causing junction is hidden. The fact that the program does not know at this point about gravity can be visualized as meaning that the objects which form both sides of a crack can appear

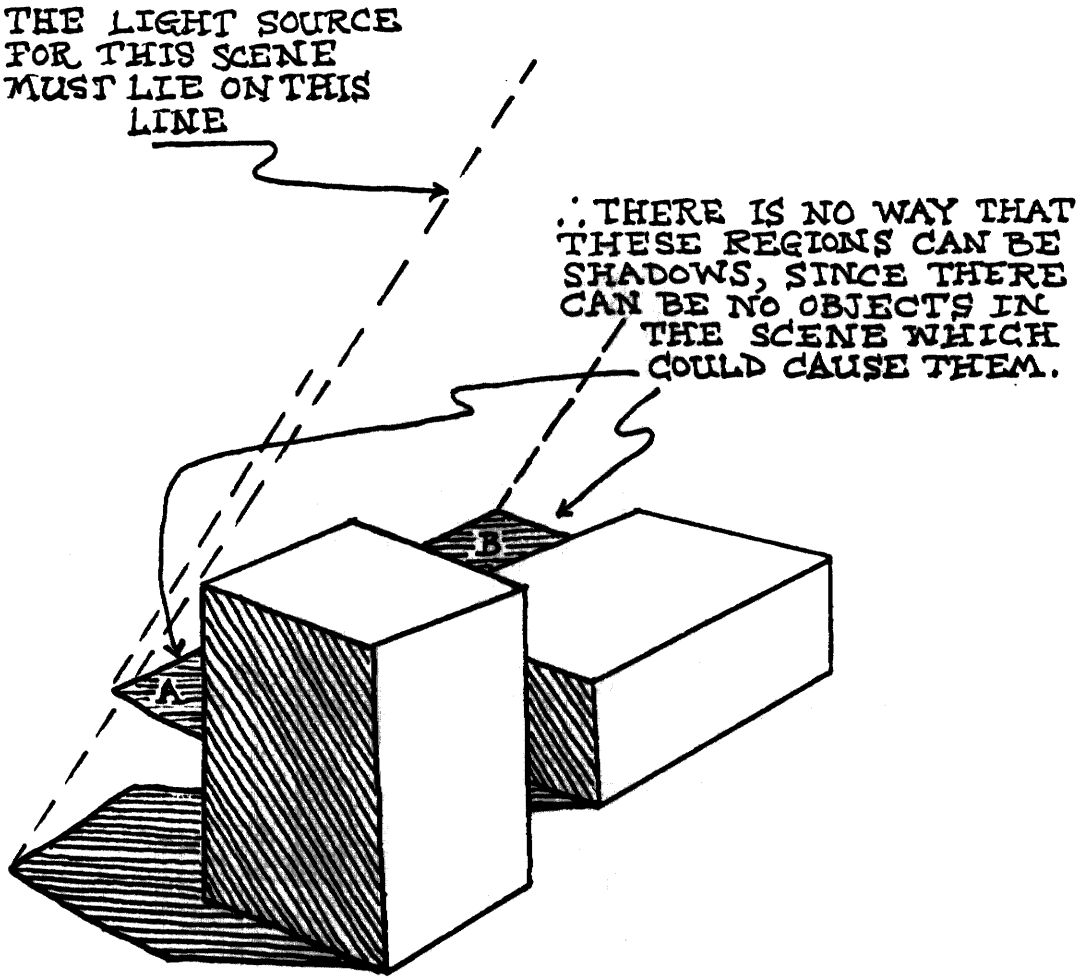
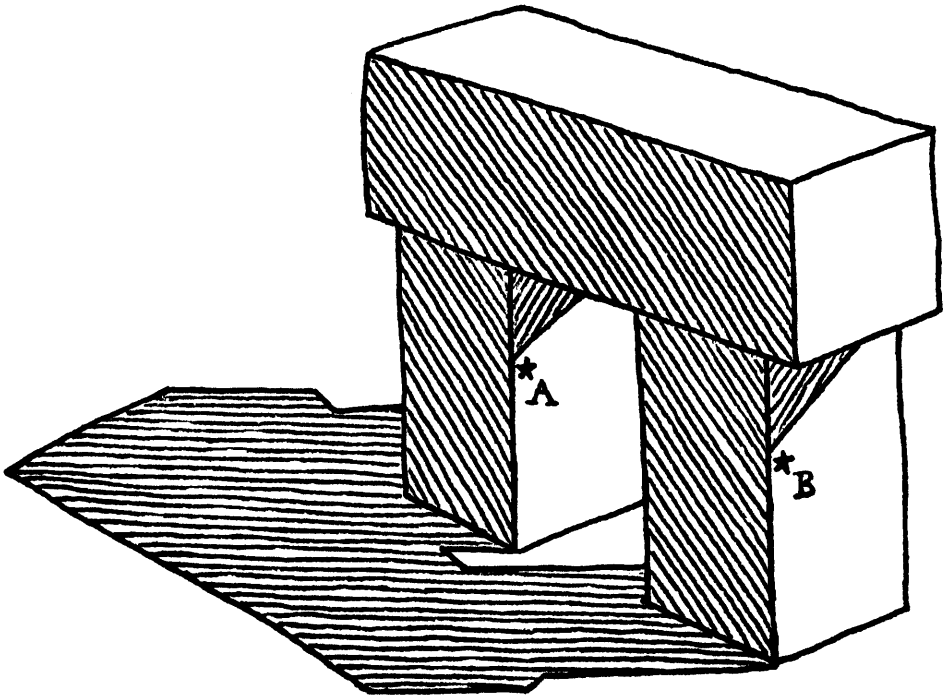

THE LIGHT SOURCE FOR THIS SCENE MUST LIE ON THIS LINE
∴ THERE IS NO WAY THAT THESE REGIONS CAN BE SHADOWS, SINCE THERE CAN BE NO OBJECTS IN THE SCENE WHICH COULD CAUSE THEM.
A
B

FIGURE 6.13



*EACH OF THESE IS AMBIGUOUS:

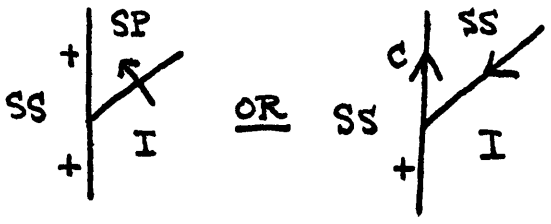
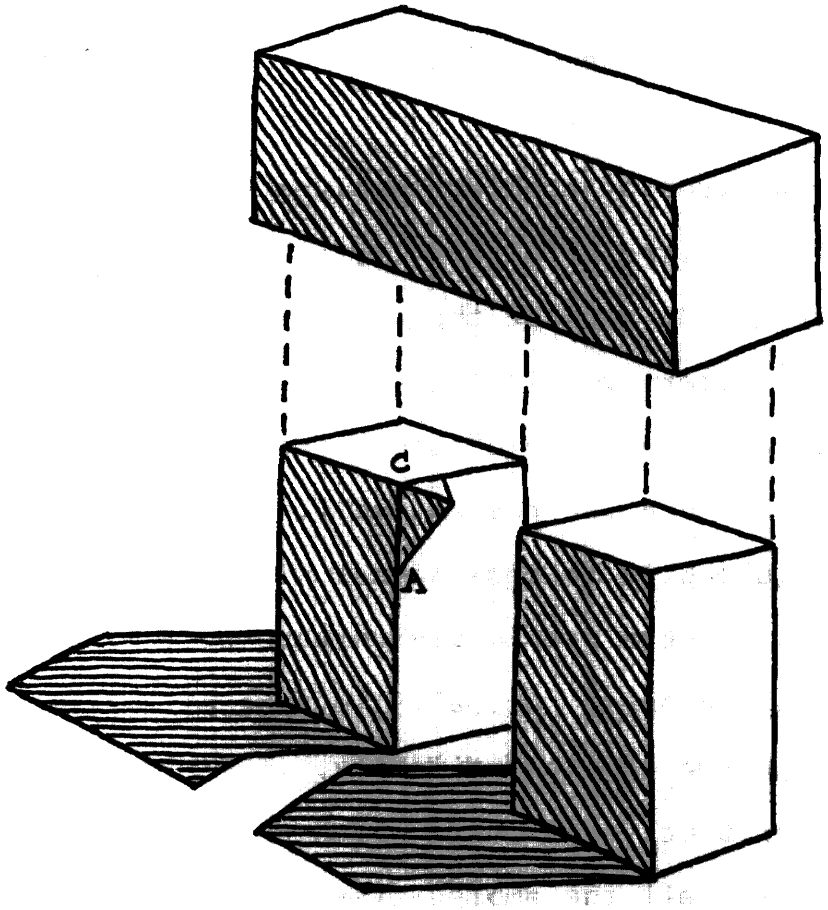


FIGURE 6.14A



SINCE THE PROGRAM DOES NOT KNOW ABOUT DIRECTIONS, IT FINDS THAT L-A-C CAN POSSIBLY BE A CRACK – WITHOUT DIRECTIONS IT CANNOT KNOW ABOUT GRAVITY.

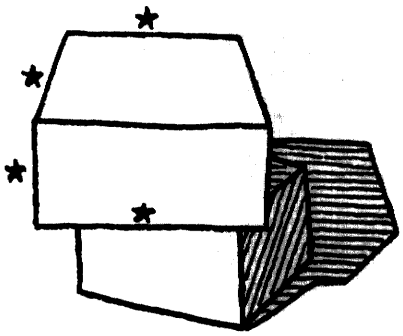
FIGURE 6.14B

anywhere, just as if the two objects were glued together. Figure 6.14B shows such a case.

The next type of problem involves support directly. An example of this type of difficulty is shown in figure 6.15. As in figure 6.10, each of the edges which is ambiguous is marked with a star (★) in figure 6.15A, and the possible labelings, both "reasonable" and "unreasonable" are shown in figures 6.15B and 6.15C respectively. I have redrawn figure 6.15C in figure 6.16 to show scenes with the same topology which have what were previously unreasonable labelings as their reasonable ones. Actually in some of the cases I have had to change the topology slightly. This happened because I wanted to construct an example which contained shadows and which exhibited all the ambiguities I show in figure 6.15; while I was not able to easily find a scene which satisfied these criteria and also did not require changes in topology, there probably are such scenes. I do not believe that any general rules can be derived from the needed modifications.

One final type of ambiguity is interesting and also serves to emphasize one of the findings of the work reported in this chapter. In figure 6.17 I show the two types of interpretations my program returns for holes. One of these

A:



B:

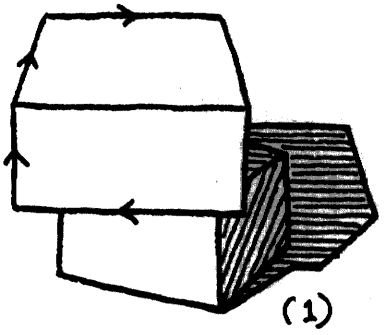


FIGURE 6.15

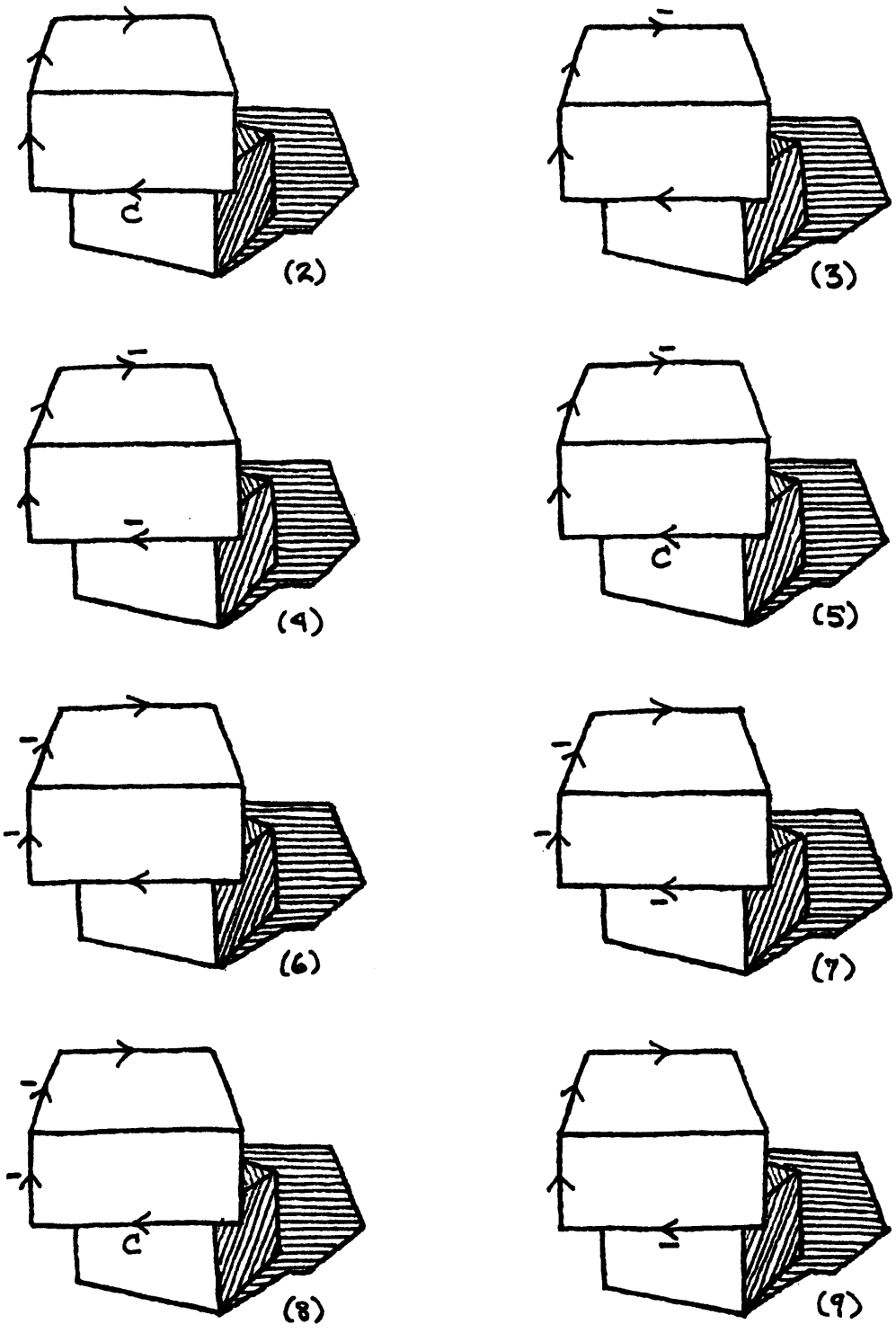


FIGURE 6.15C

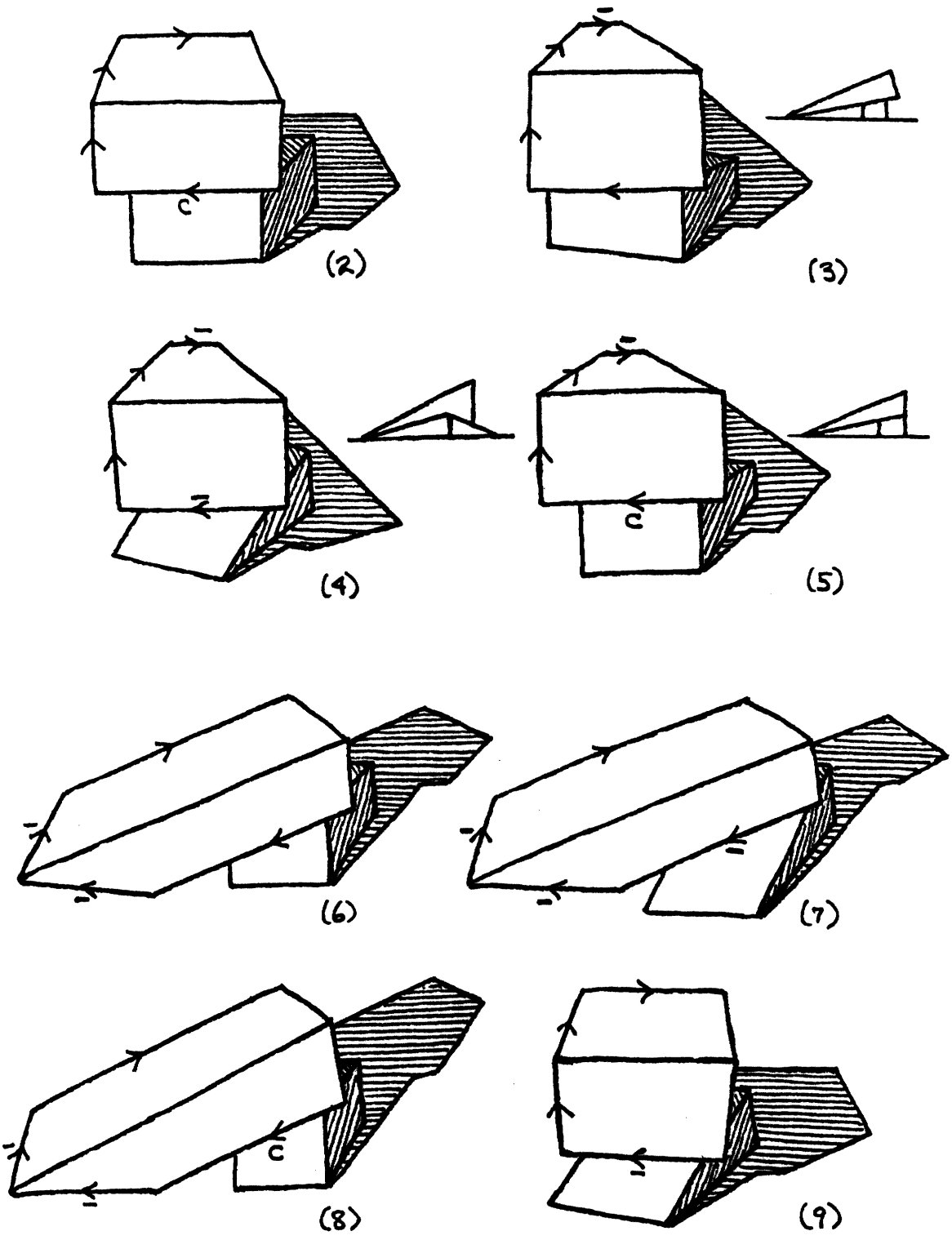
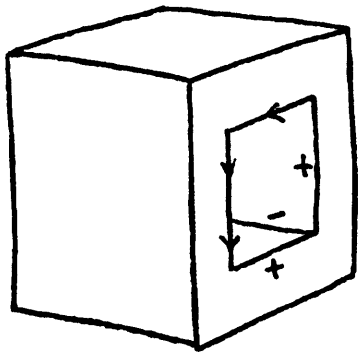
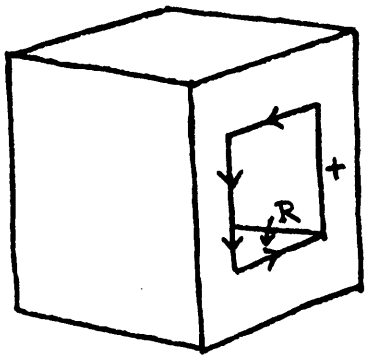


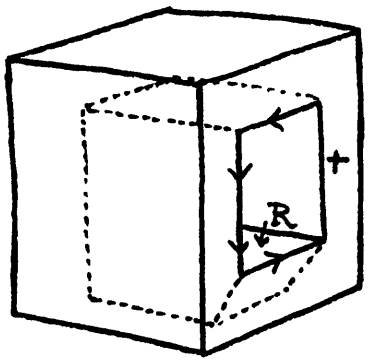
FIGURE 6.16



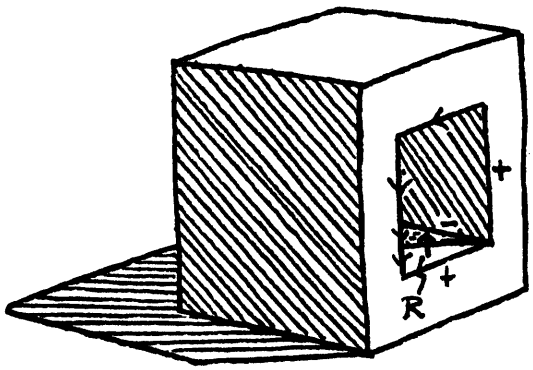
THE EXPECTED INTERPRETATION



REASONABLE BUT UNEXPECTED INTERPRETATION; THE HOLE GETS LARGER THE DEEPER ONE GOES INTO THE BLOCK.



DOTTED LINES SHOW A POSSIBLE HOLE FOR THE SECOND INTERPRETATION



WITH A SHADOW OF THIS TYPE INSIDE THE HOLE IS NO LONGER AMBIGUOUS.

FIGURE 6.17

interpretations is the one I expected; I was surprised that the hole was ambiguous, but even more surprised to find that I had missed an obvious alternate interpretation of the same geometry. The alternate interpretation shown in figure 6.17B does not even need to be drawn with different line segment directions in order to appear reasonable.

The labelings which the program finds must be made up of local features, each one of which is physically possible, but it is not obvious that the features which remain should each be part of a total labeling of the scene which is physically possible. After all, the only conditions I impose are that each of these features must agree with at least one other feature at each neighboring junction. On the basis of the fact that the main labeling program does not leave extraneous labels on junctions, it seems clear that topology provides a major portion of the cues necessary to understand a scene.

In the next chapter I show some heuristic rules which can be used to eliminate some of the labelings which people usually consider unlikely. In fact the true case is that these labelings are not unlikely, but the scenes which have these labelings as reasonable ones (to our eyes) do not often arise in our experience. Unfortuntely, heuristics sometimes

reject real interpretations, and indeed would reject each of the interpretations shown in figures 6.11 and 6.16 in favor of the ones in figures 6.10B and 6.15B. Nonetheless, in the absence of solid rules, these heuristics can be useful. In the chapter on region orientations I deal with the types of techniques which would enable a program to find the labelings which we would assign to these line drawings without resort to heuristics.

7.0 NON-TRIHEDRAL VERTICES & RELATED PROBLEMS

So far I have assumed that all the junctions I am given are normal trihedral junctions and essentially that the line drawing which I am given is "perfect". When a program has to be able to accept data from real line finders and from arbitrarily arranged scenes, these criteria are rather unrealistic.

In this chapter, I show how to correct some of these problems in a passive manner. By passive I mean that the program is unable to ask a line finding program to look more carefully or to use alternative predicates at a suspicious junction, and similarly that it cannot move its eye or camera, or direct a hand to rearrange part of a scene in order to resolve ambiguities (Gaschnig 1971).

Instead I handle these types of problems by including labels for a number of the most common of these junctions in the regular data base. In cases where the program confuses these junction labelings with the regular labelings and where I want a single parsing, I can easily remove these new types of junction labels first, since I have included special markers for each labeling of this type. Moreover, depending

on the reliability of the program which generates the line drawing, I may wish to remove labels in different orders. For example, if a line finding program rarely misses edges, missing edge interpretations can be removed first; if a line finding program tends to miss short line segments, then accidental alignments are probably being generated by the program, and these interpretations can be retained until last. Therefore the labels for each type of problem are marked with different indicators in the data base.

7.1 NON-TRIHEDRAL VERTICES

Some non-trihedral vertices must be included in the data base; indeed some are much more common than many of the trihedral vertices. I will limit the number by including only those non-trihedral vertices which can be formed by convex trihedral objects.

The first type of vertex is formed by the alignment of a vertex with a convex edge as shown in figure 7.1 and in figure 7.2. In figure 7.3 a similar set of junctions is shown for objects which MARRY (i.e. have coplanar faces separated by a crack edge; see Winston 1970) along one edge, but which have difference face angles.

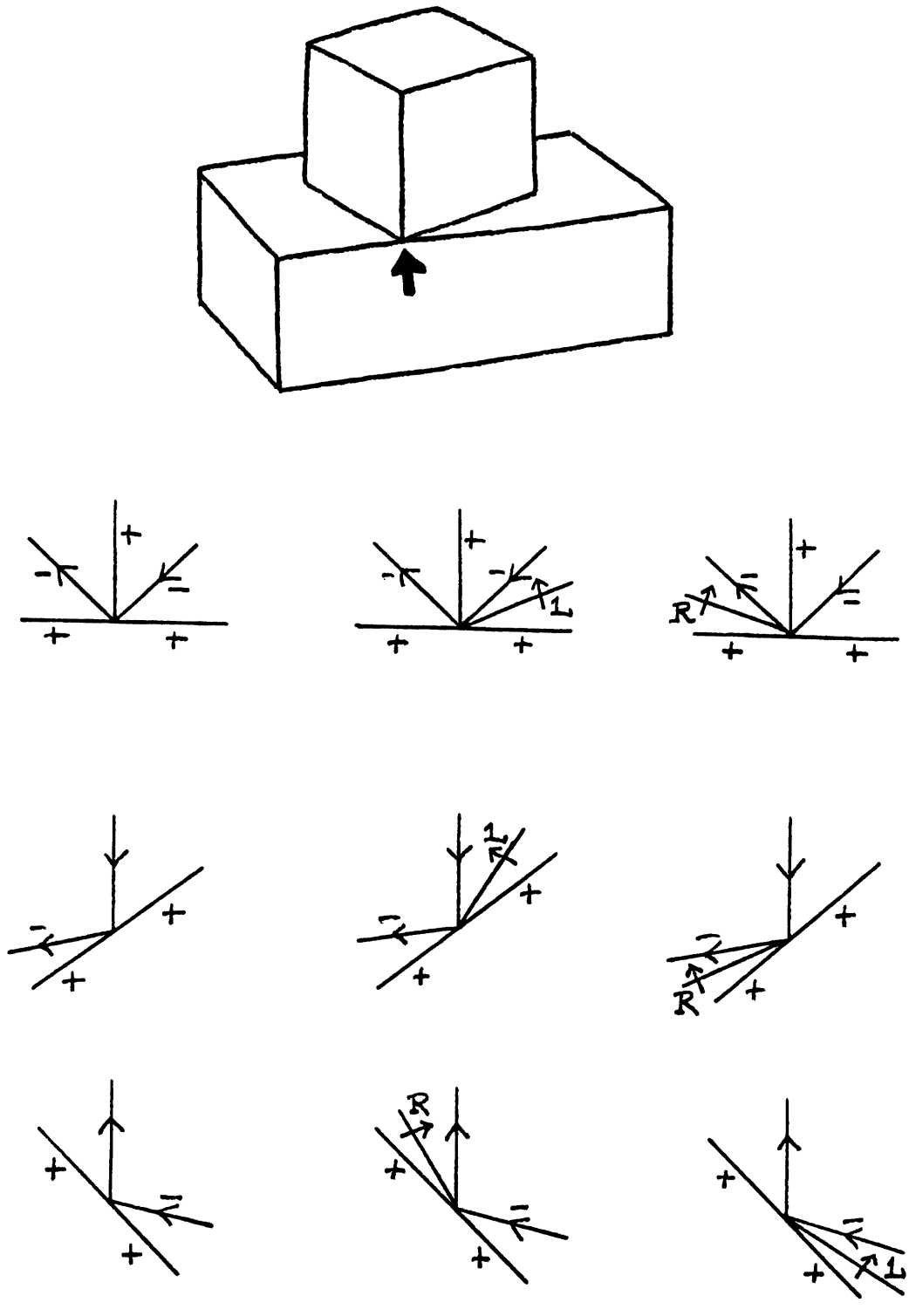


FIGURE 7.1

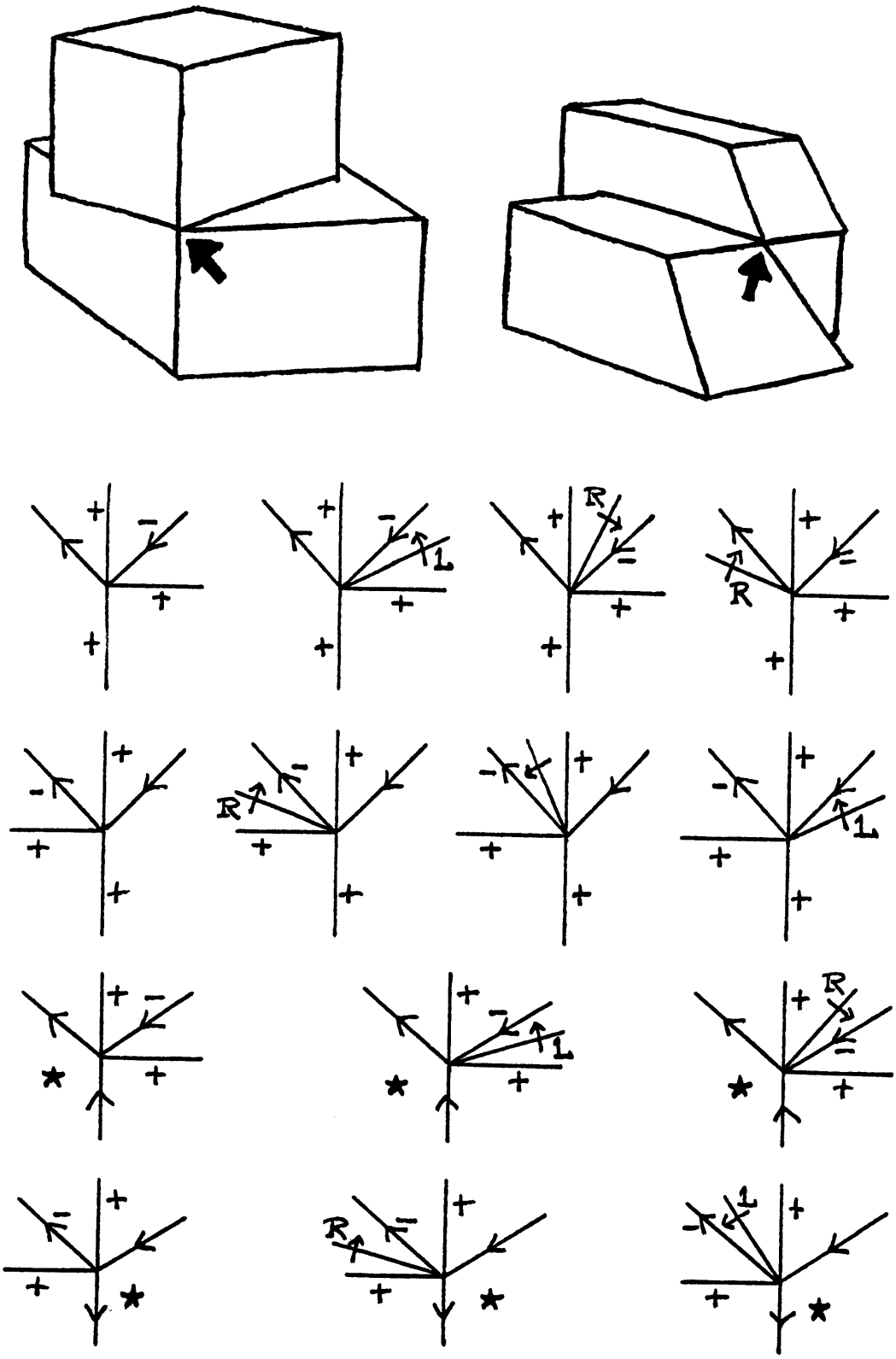


FIGURE 7.2

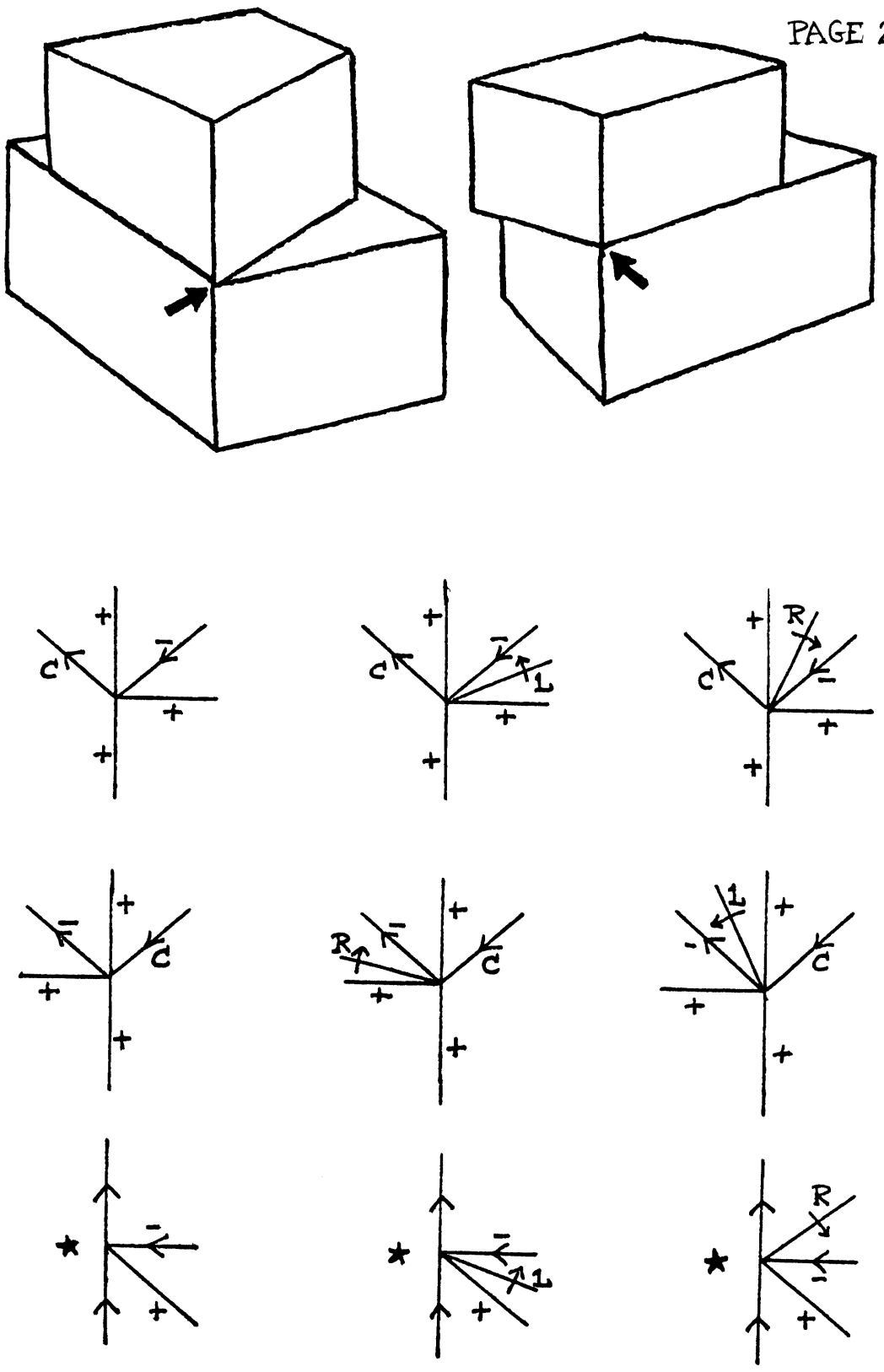


FIGURE 7.3

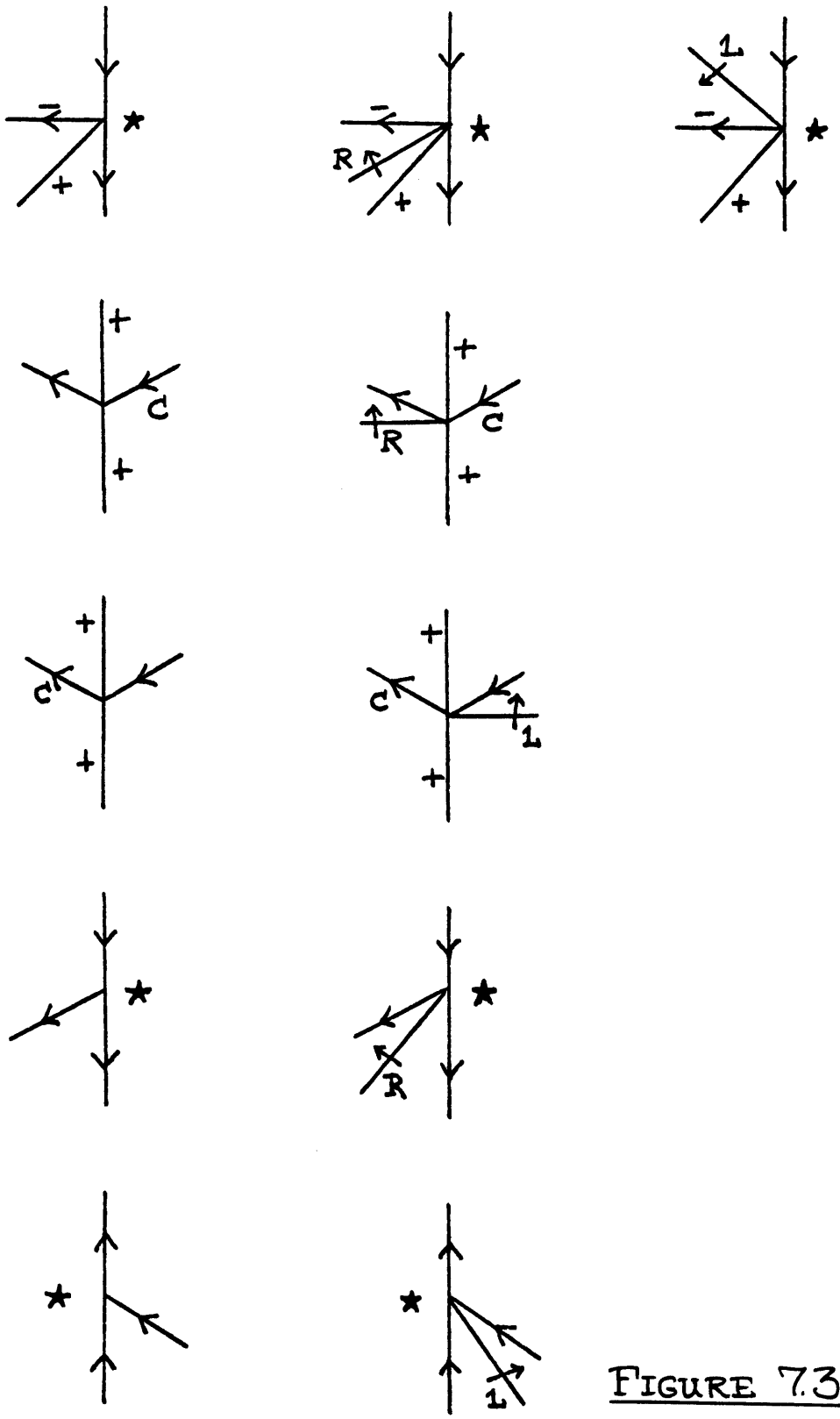
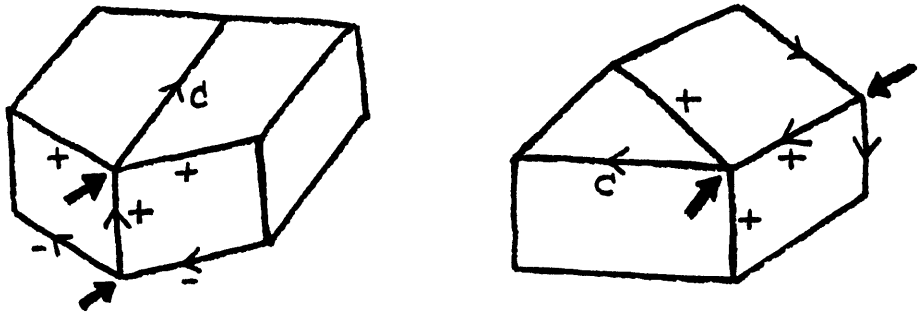


FIGURE 7.3

Figure 7.4 illustrates another common non-trihedral vertex which results again from objects with dissimilar face angles. This time I need a new type of edge, a separable convex edge, labeled as shown in figure 7.4.

Figure 7.5 illustrates the types of non-trihedral vertices which can occur when one block leans on another. In order to keep these cases from being confused with other trihedral junctions, I have introduced three new edge types. These types only can occur in a very limited number of contexts. Figure 7.6 shows some of the ways in which these edges can appear.

In the data base each of the labelings shown in figures 7.1, 7.2, 7.3, 7.4, and 7.5, and any other junction labels involving the leaning edges or the separable convex edges, are marked as non-trihedral. Later, if I wish to find a single parsing for a scene where there are still ambiguous labels, removing these non-trihedral junctions, if possible, may be a good heuristic.



NEW LABEL:

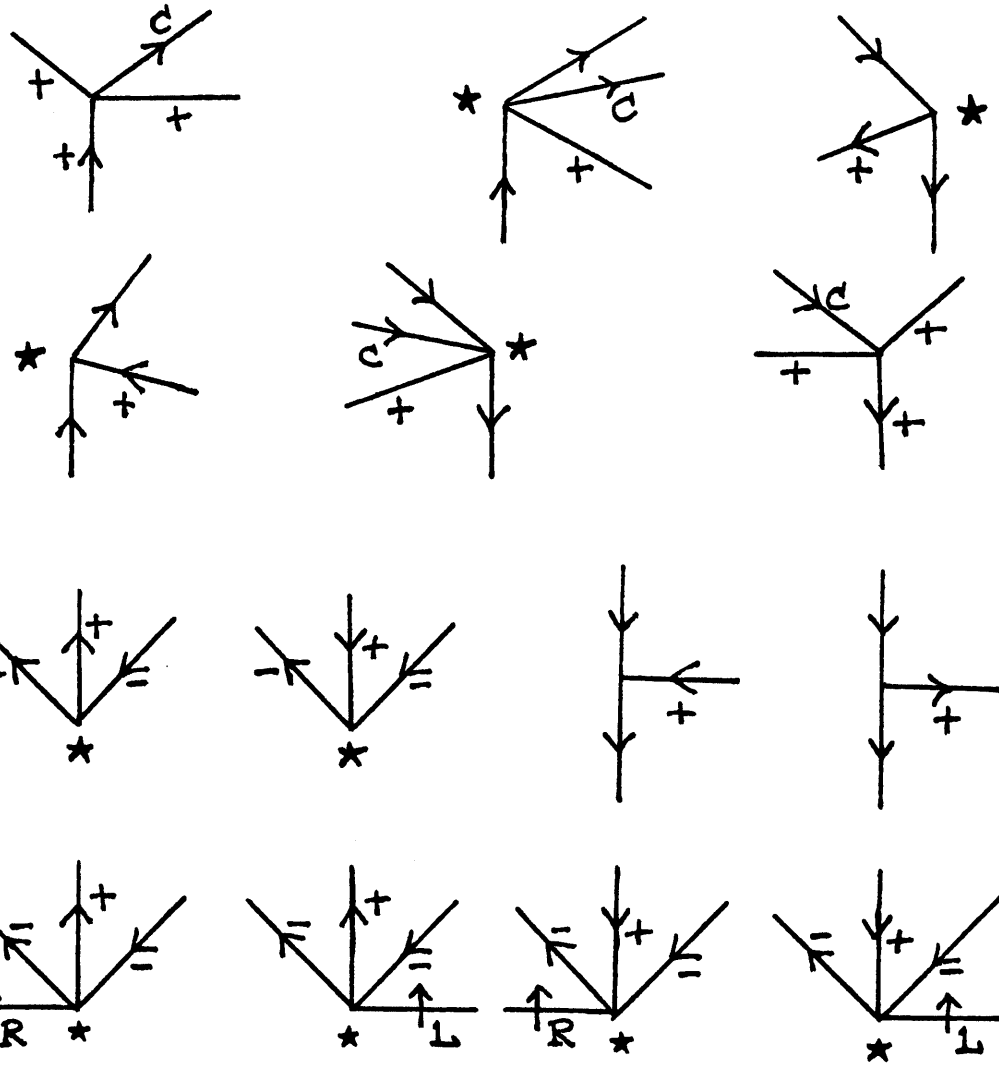
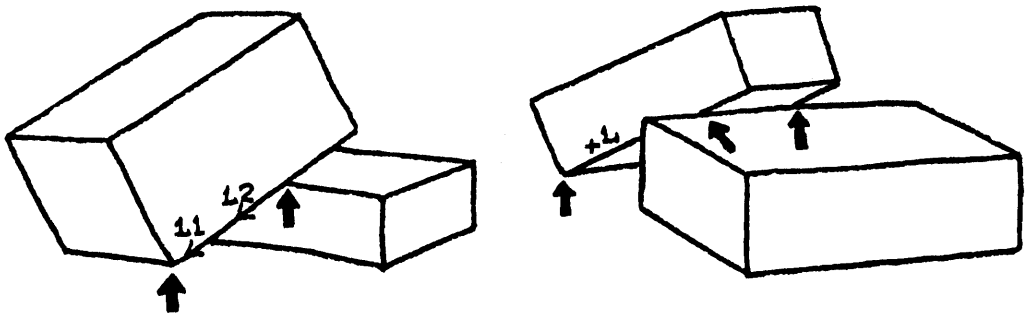


FIGURE 7.4



NEW LABELS: L1 L2 +L

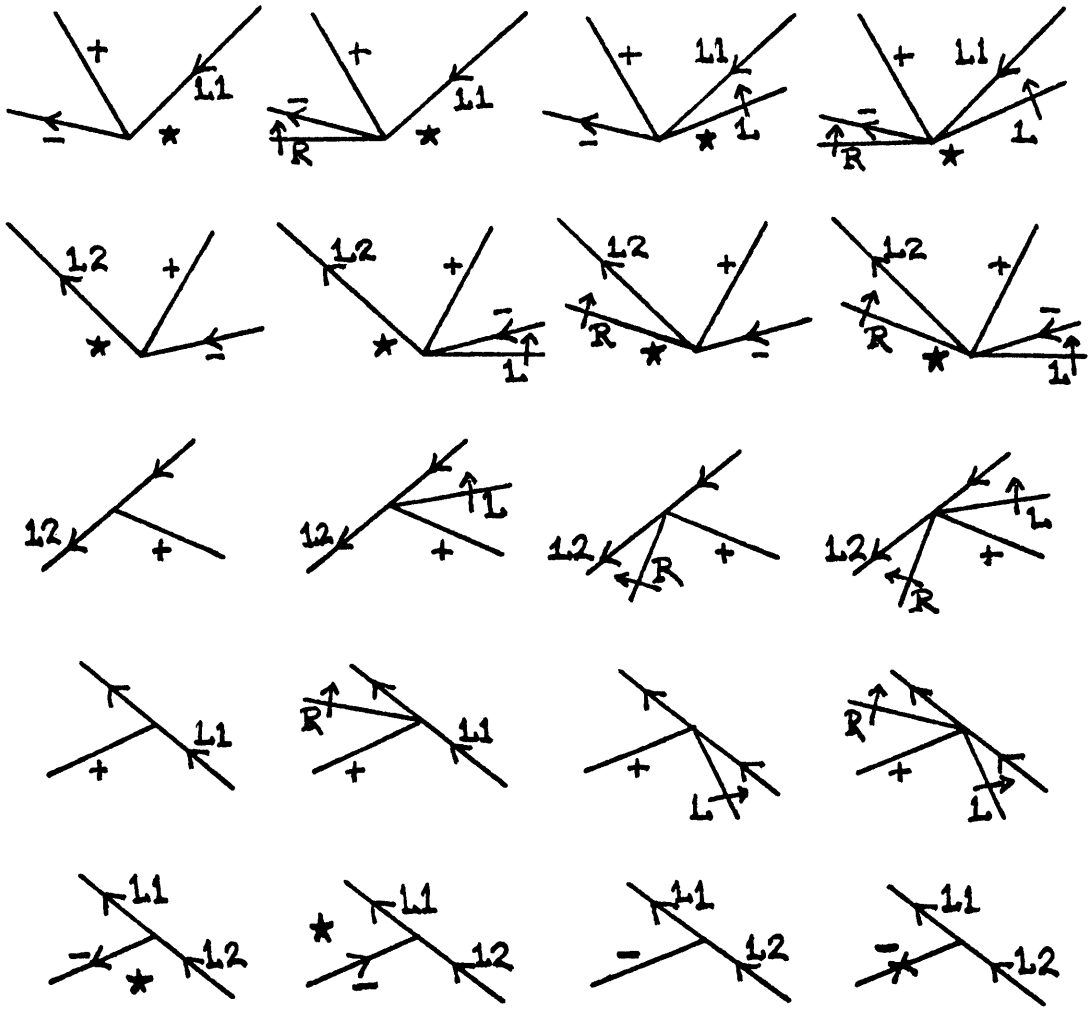
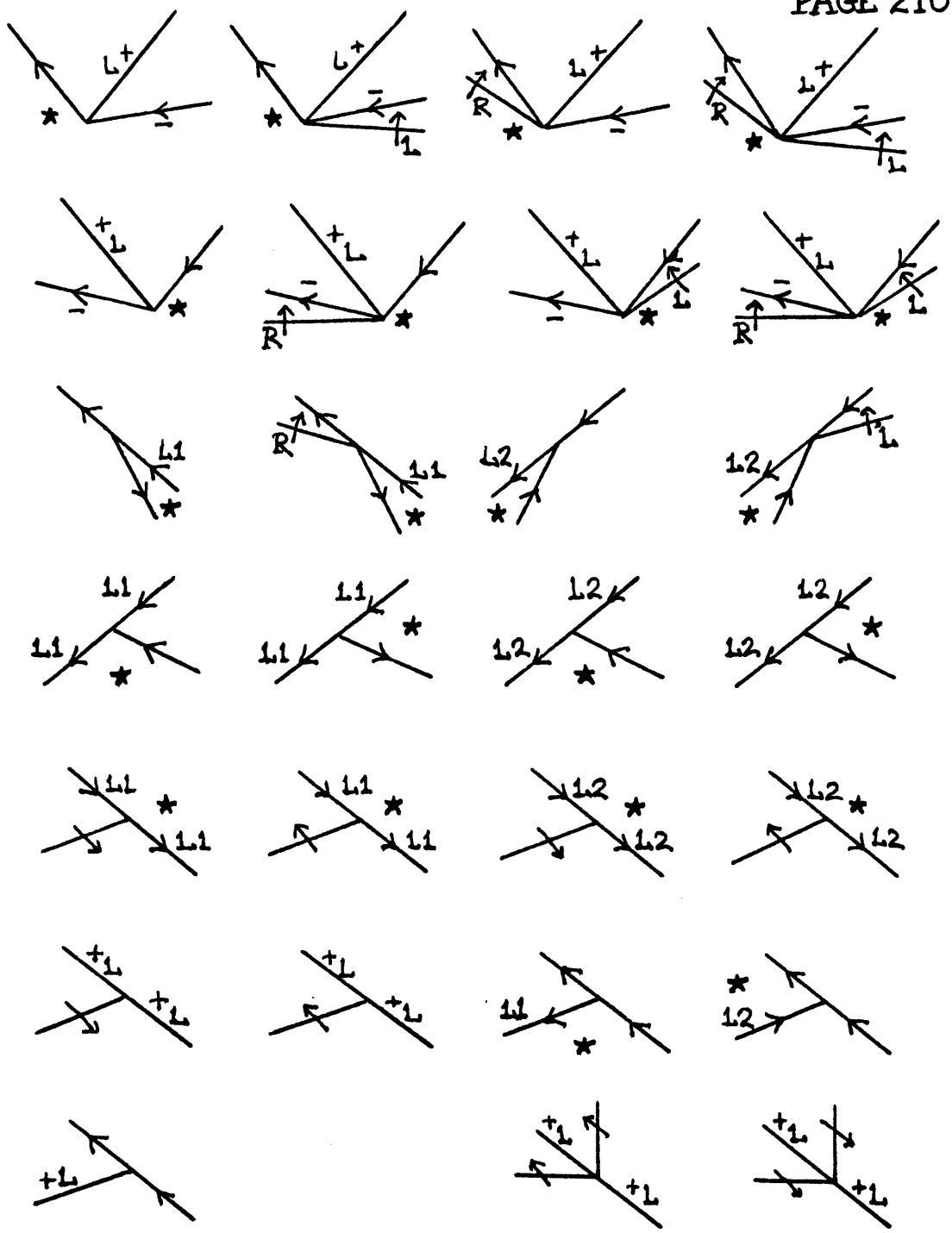


FIGURE 7.5



(THE OBSCURING L1 & L2 EDGE CAN ALSO FORM OBSCURING Ts WITH OTHER EDGES AS WELL BUT THESE ARE LESS LIKELY.)

FIGURE 7.5

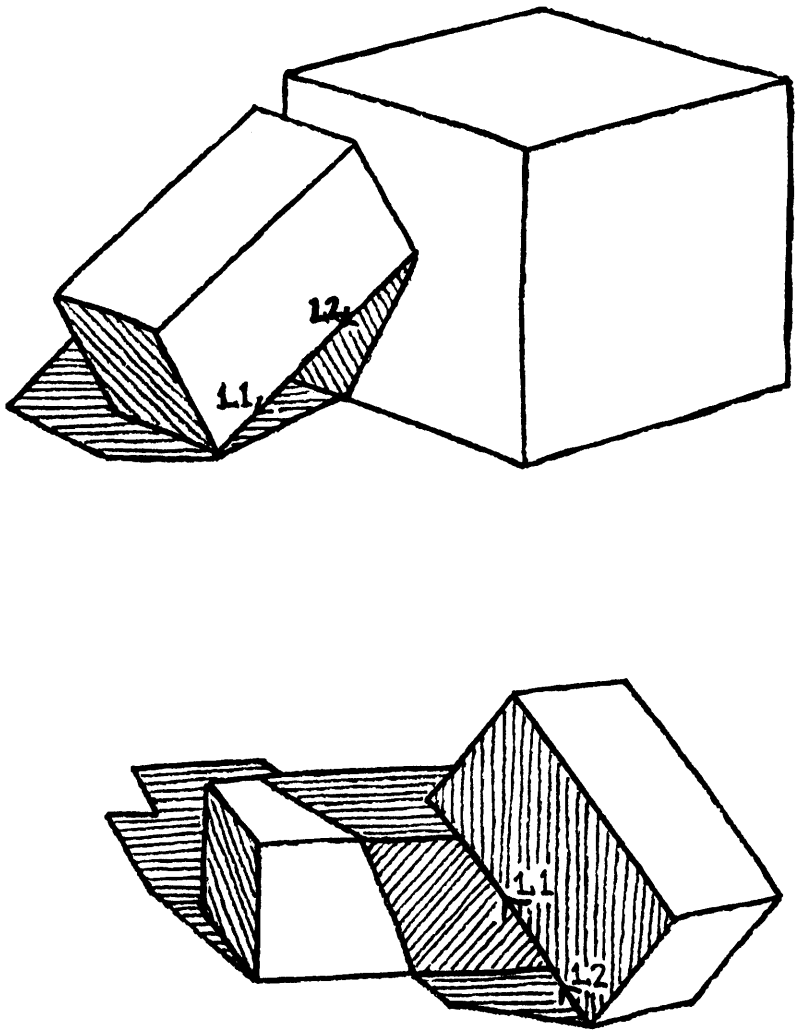


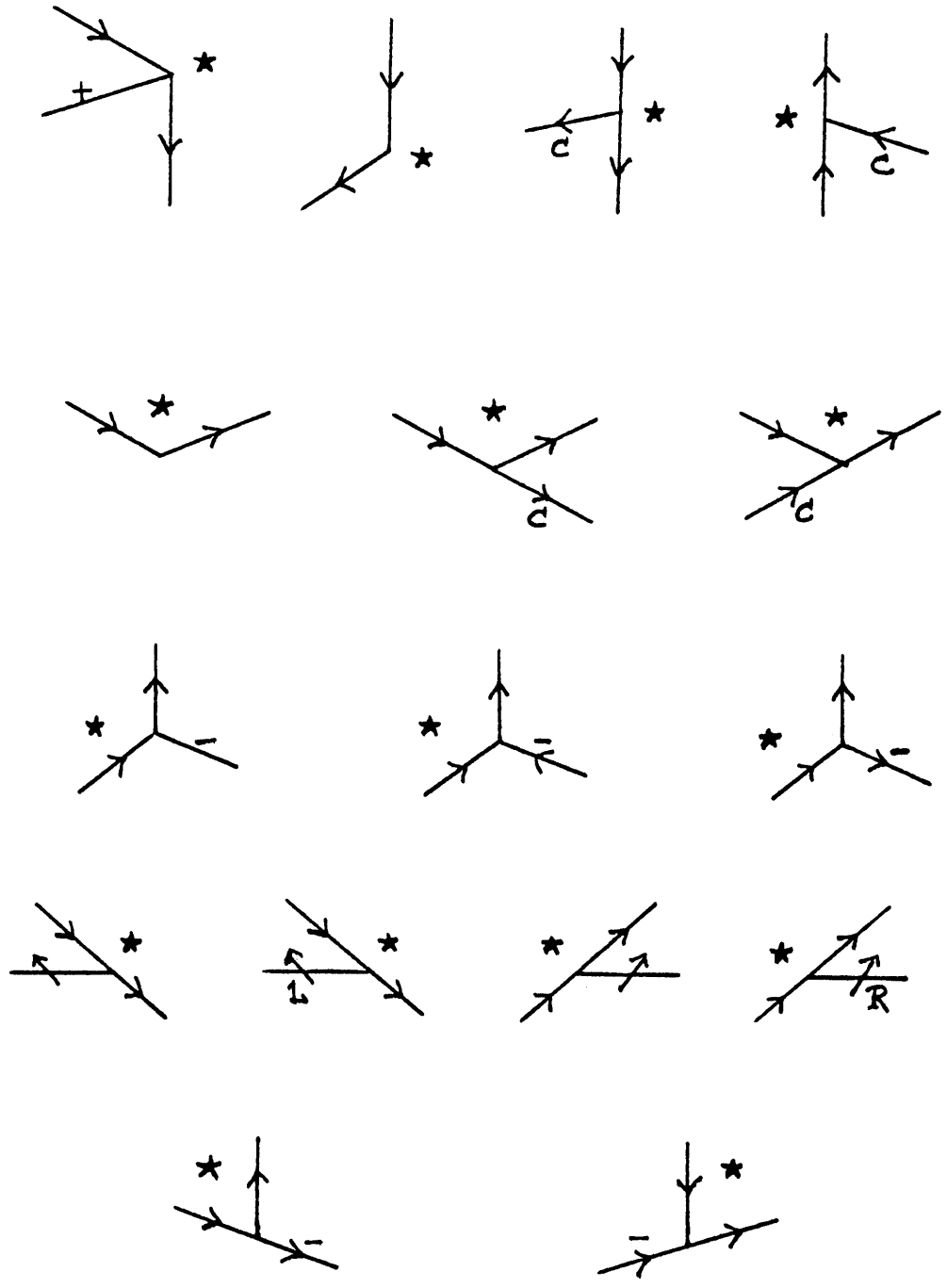
FIGURE 7.6

7.2 ACCIDENTAL ALIGNMENTS; FIRST TYPE

In this section I have not attempted to exhaustively list every possible junction labeling which results from accidental alignment, but have concentrated on including only the most common cases. There is some justification for this, in that ambiguities caused by accidental alignments can be resolved by simply moving with respect to the scene.

Figure 7.7 lists all the junctions which can take part in the first type of accidental alignment I will consider. This type of alignment occurs when a vertex is closer to the eye than an edge which appears to be but is not part of the vertex. Thus the set of vertices in figure 7.7 are exactly that subset of the scene/background boundary junctions (Appendix 4) which contain only obscuring edges on the scene/background boundary. Figure 7.7 shows only those junctions which I include as sufficiently common. The rest are excluded because they involve unusual concave geometries like those found in SOMA cube pieces (SOMA cubes are three-dimensional puzzles manufactured by Parker Bros. Inc., Salem, Mass.) or because they involve three-object edges or because the resulting junction would have enough line segments to require a designation of "SPECIAL" or because the

JUNCTIONS WHICH ARE USED TO MAKE UP ACCIDENTAL ALIGNMENT LIST



(EXTRA EDGES ARE IN STARRED REGIONS)

FIGURE 7.7

junction would require the alignment of the eye with three points in space.

There is no regular junction which could be confused with any of the ARROW or K junctions generated by the the alignment of the junctions shown in figure 7.7 with edges behind them. To see why this is so, consider figures 7.8 and 7.9. Figure 7.8 gives names for the distinguishable region segments for each type of junction. Figure 7.9 shows all the K and ARROW junctions that can result from accidental alignment with each each of the junctions shown in figure 7.7. Notice that the background region can only appear in segments ARROW1, ARROW2, K1, K2 and K3 in these accidentally aligned cases, whereas for all trihedral ARROW and K junctions which can appear on the scene/background boundary, only segments ARROW0 and K0 (the segments of these junctions which are greater than 180 and equal to 180 degrees, respectively) can be part of the background. Of course for the junctions where no segments are distinguishable (e.g. FORKs) or where the junction appears on the interior of the scene, these accidental alignment cases cannot be directly distinguished from the regular cases.

NAMES OF DISTINGUISHABLE JUNCTION SEGMENTS

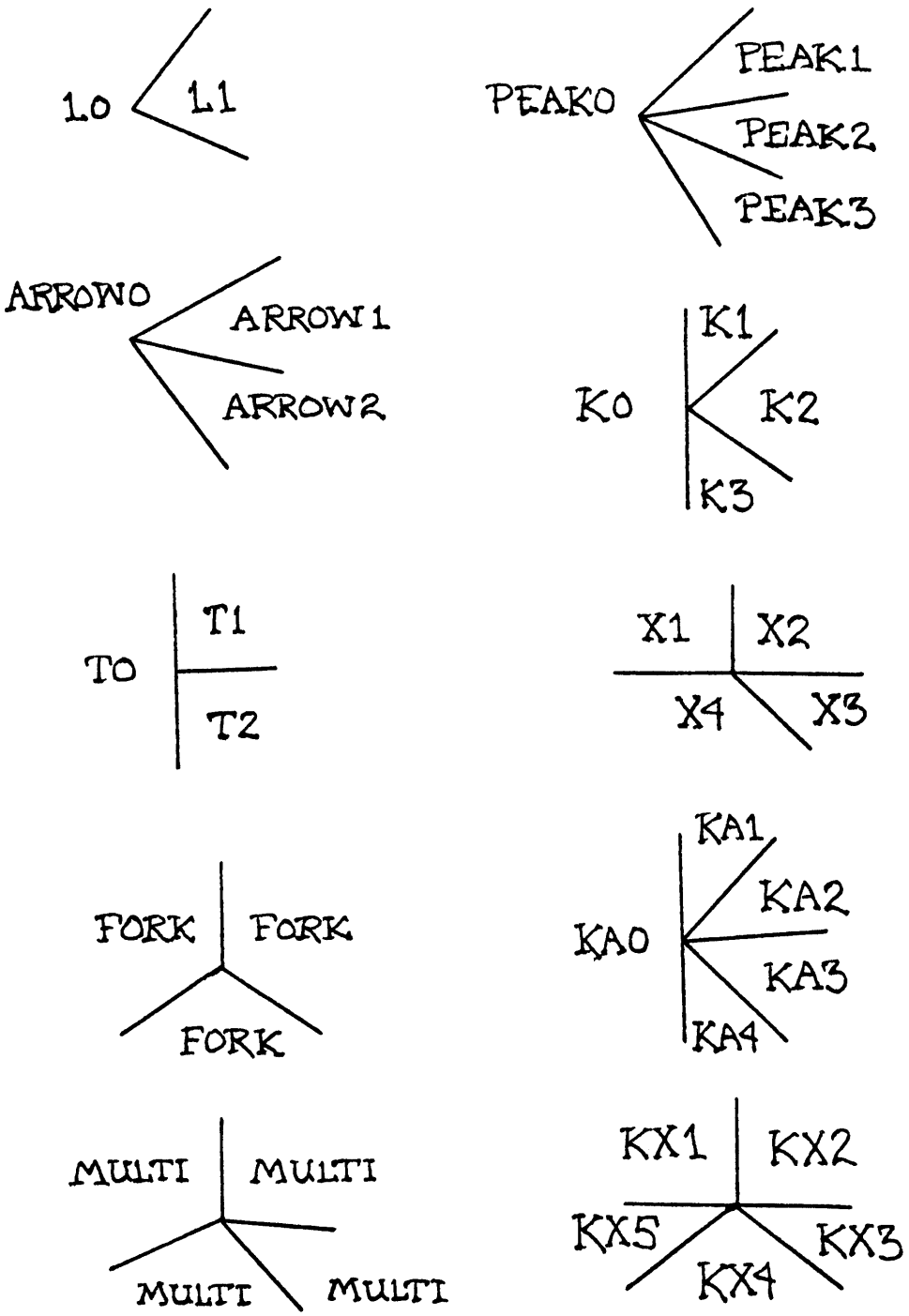


FIGURE 7.8

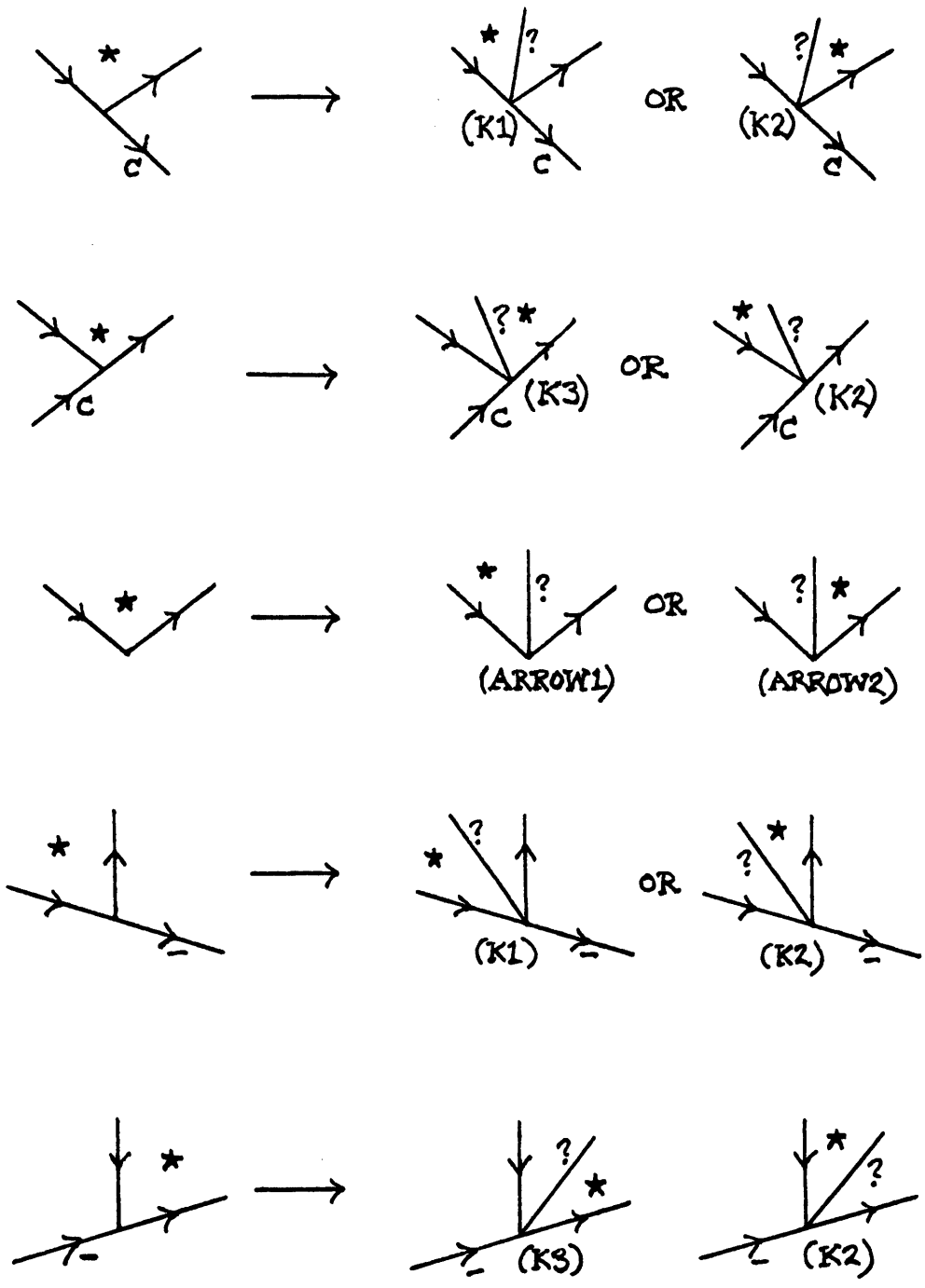


FIGURE 79

At this writing, I have not included all these accidental alignment types in the program's data base, but I have included most of the scene/background boundary cases and a number of the interior cases. In general, I have assumed that no non-trihedral edges or three-object edges will be among those obscured since both the alignment itself and the edge types are relatively unlikely, so their coincidence at a single junction is extremely unlikely.

7.3 ACCIDENTAL ALIGNMENT WITHOUT OBSCURING EDGES

Figure 7.10 shows some alignments which have shown up frequently in scenes I have worked with. These junctions have occurred because (1) our line finding program misses short line segments (and therefore tends to include more lines than it should in a single junction), (2) our line finding program has a tolerance angle within which it will call edges collinear, so some edges are called collinear even when they are not, and (3) edges which lie in a plane parallel to the surface on which they cast shadows are parallel to the shadows they cast, so that alignments become particularly likely when we use bricks, cubes, and prisms.

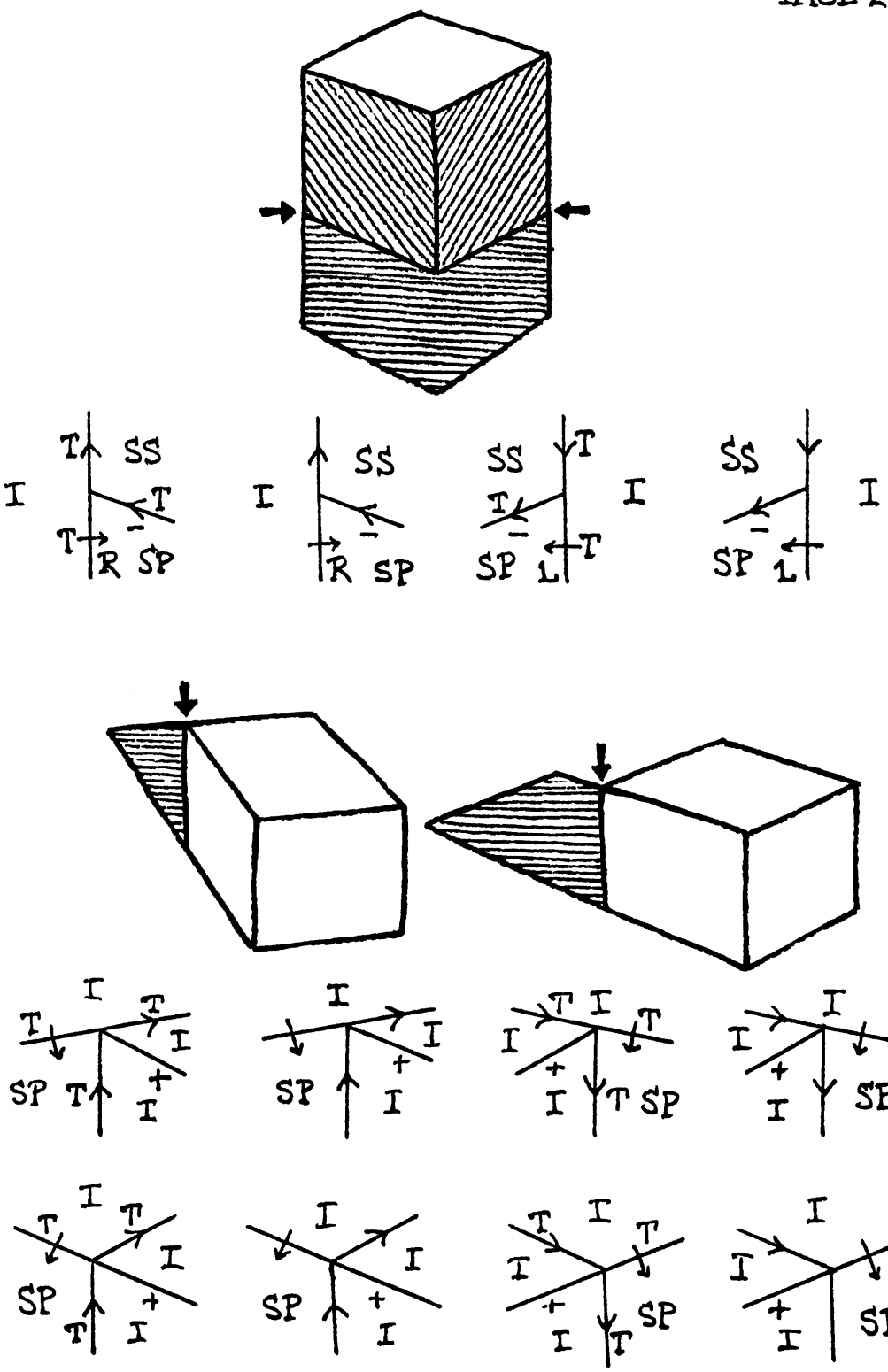
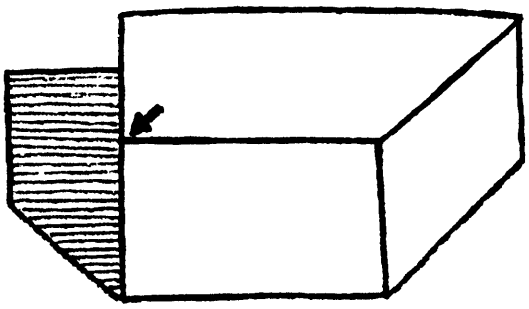
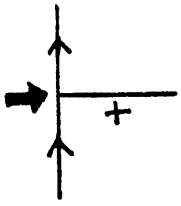

T
SS
I
T
T
R
SP
SS
I
R
SP
SS
T
I
T
SP
L
T
SS
I
SP
L
I
T
T
I
SP
T
I
I
I
SP
I
T
I
T
I
T
SP
I
I
SP
I
T
T
I
SP
T
I
I
I
SP
I
T
I
T
I
SP
I
T
I
I
SP
I

FIGURE 7.10





[THERE ARE 35 DIFFERENT COMBINATIONS OF REGION ILLUMINATIONS AND TABLE OR INTERIOR LABELS FOR THIS JUNCTION]

FIGURE 7.10

Figure 7.11 shows some other types of accidental shadow edge alignment which our group's line finding program frequently yields; these junctions are relatively common because of the tendency of the program to miss short line segments, but each of these types of alignment can occur naturally as well. (For information on our line finding program see Horn 1971 and Shirai 1972.)

7.4 ACCIDENTAL ALIGNMENTS; FINAL TYPE

The worst type of accidental alignment, in terms of the number of new junctions it can introduce, occurs when an edge between the eye and a vertex appears to be part of the vertex. Fortunately, all of the types of junctions which these alignments introduce are either Ks, KAs or SPECIALs. To see why this is so, look at figure 7.12. All these labelings can be quite easily generated by a program which operates on the regular data base. Notice that for each obscured vertex labeling, there are three new labelings generated, since the near region can have any of the three illumination values.

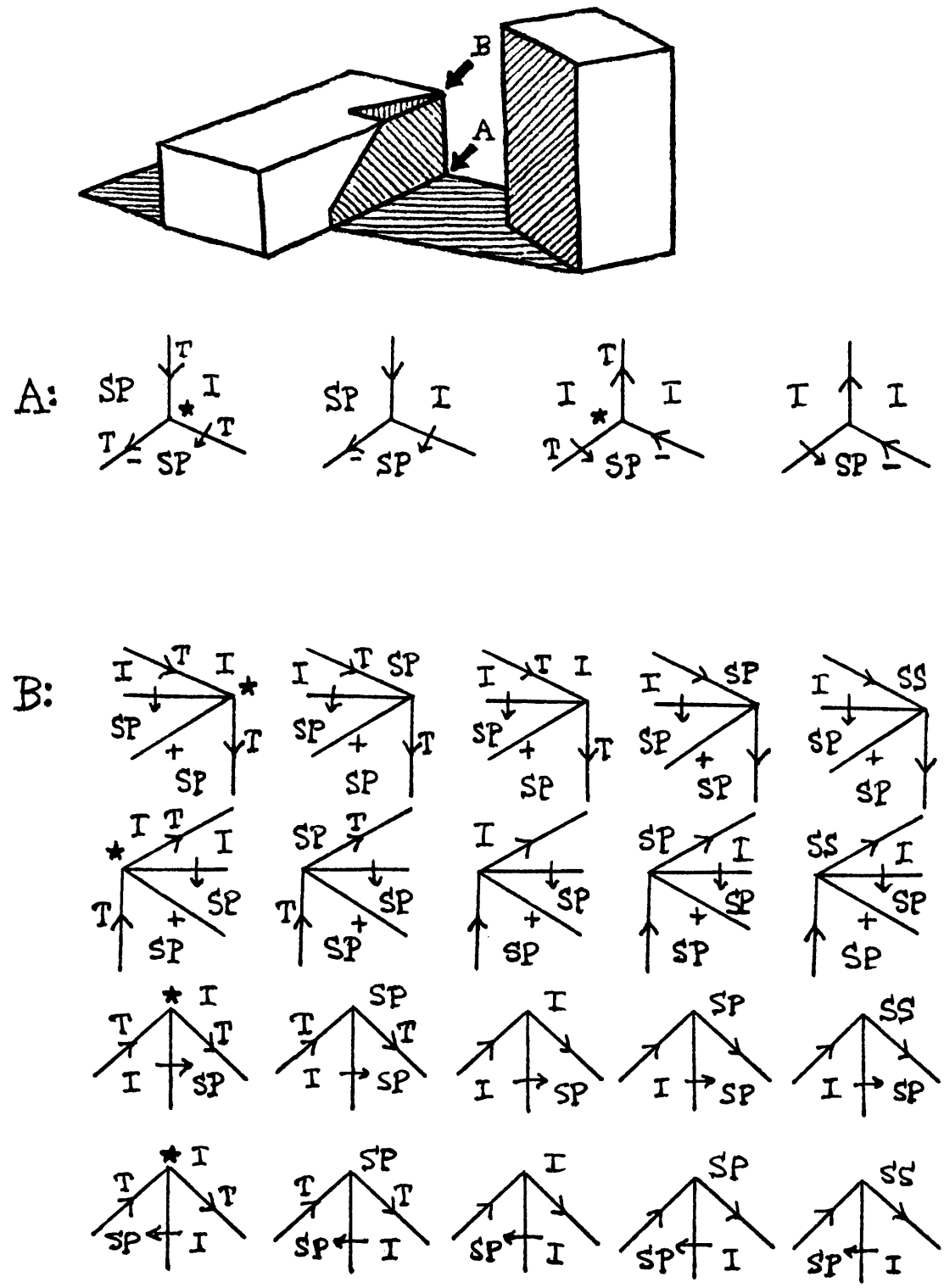


FIGURE 7.11

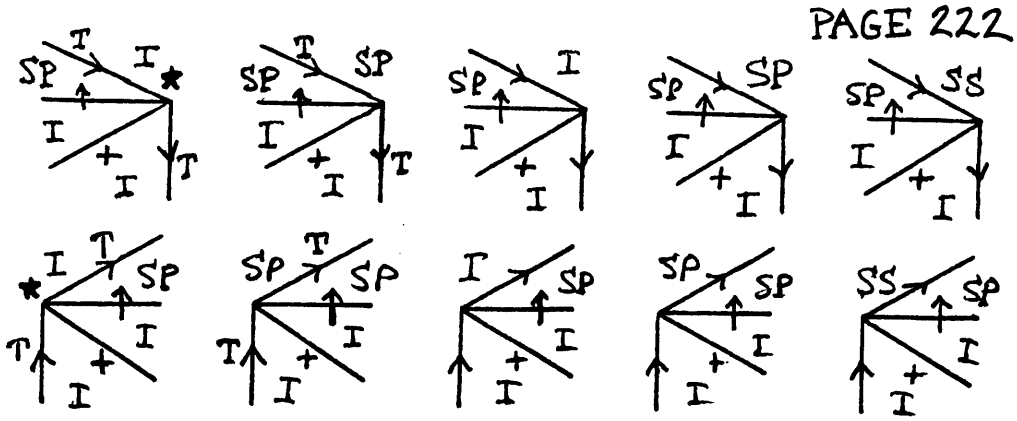


FIGURE 7.11

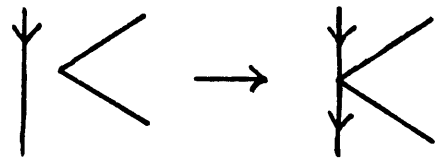
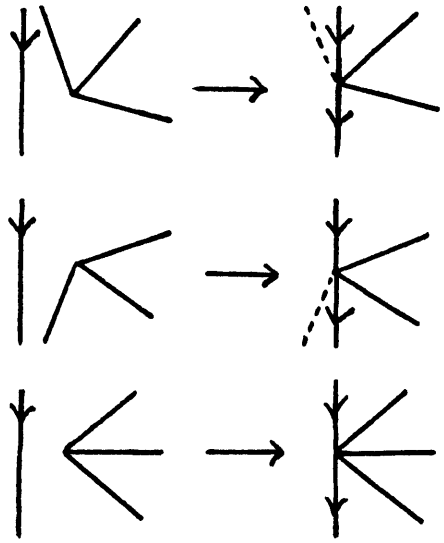
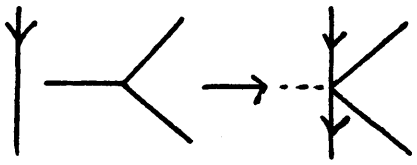
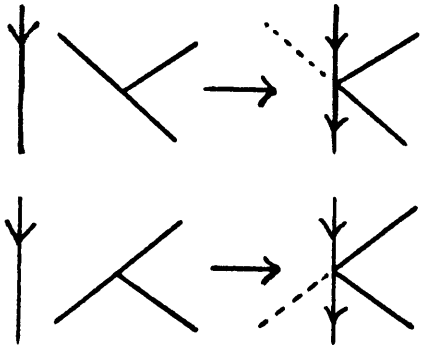
IF THE VERTEX IS:	THEN THE POSSIBLE ACCIDENTAL ALIGNMENTS WITH AN EDGE ARE:
L	
ARROW	
FORK	
T	

FIGURE 7.12

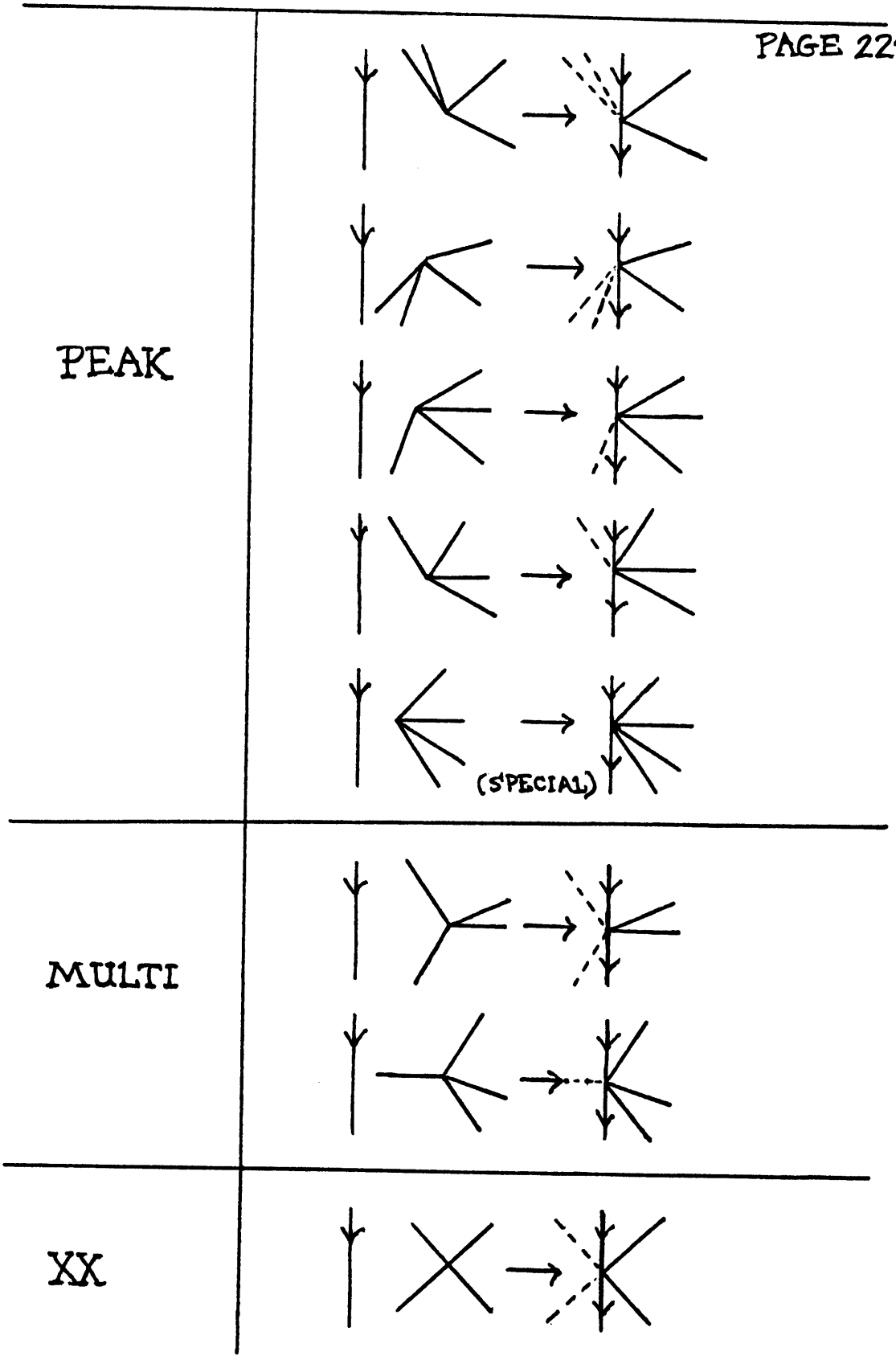


FIGURE 7.12

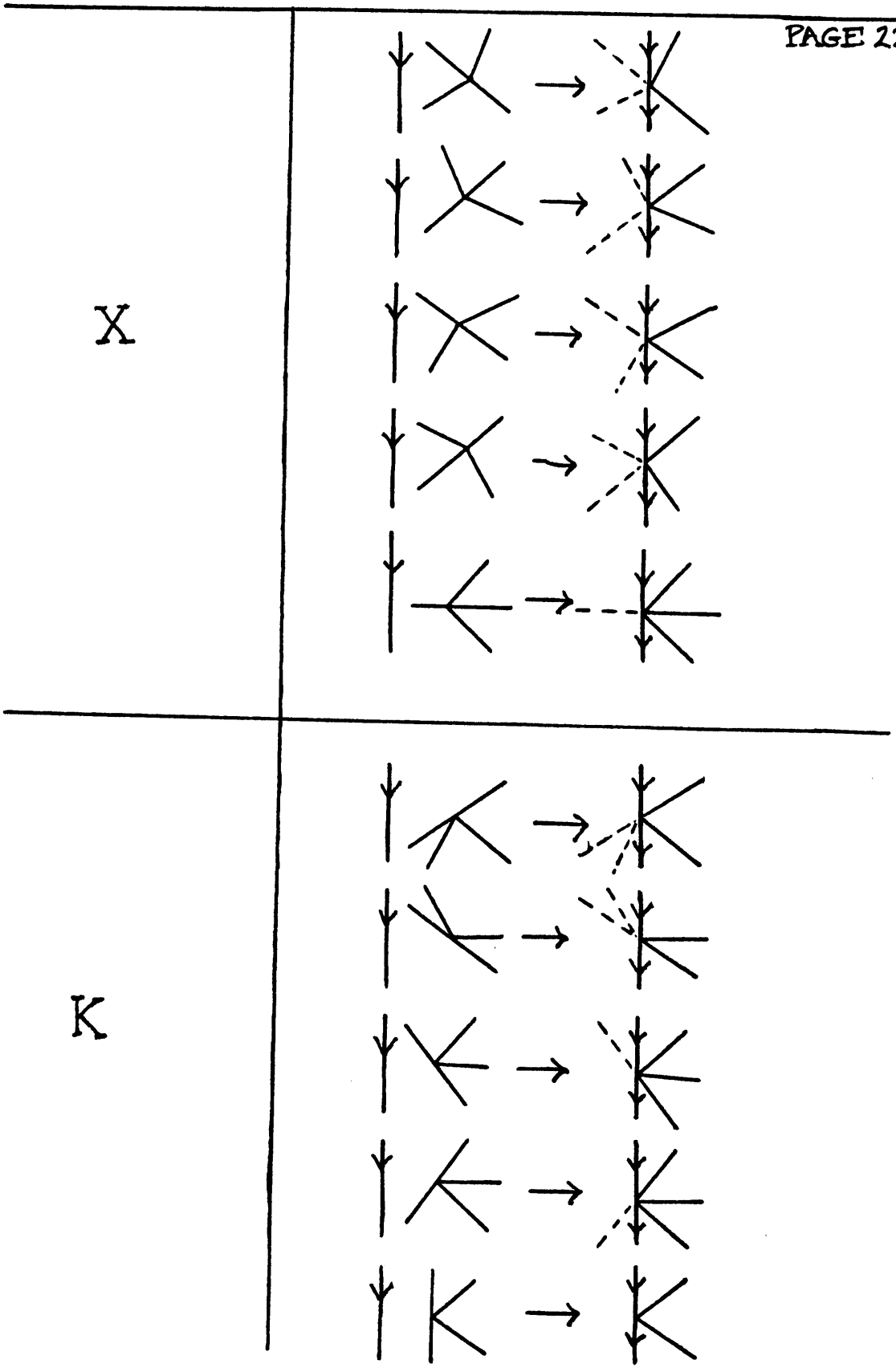


FIGURE 7.12

Also notice that any of these junctions which appear on the scene/background boundary can only be oriented with the background in a junction segment type K1, K2, K3, KA1, KA2, KA3, or KA4 (see figure 7.8). Therefore it is not difficult to recognize the cases where accidental alignments of this type occur on the scene/background boundary since none of the regular trihedral junctions can ever appear on the scene/background boundary in any of these orientations. (The background can only appear normally in segments of type K0 of KA0.)

The number of K junctions of this type which can occur is limited by the fact that two of the line segments (the collinear ones) must always be obscuring edges and so can be labeled in a total of 108 different ways (including region illuminations); the other two line segments can each be labeled in 81 ways, so there can be no more than 81x81x108 = 708,588 possible K labelings. In fact, as usual, there are not nearly this many labelings. To find the limit on the number of these junctions, use figure 7.12 and Table 5.3 together, as shown in Table 7.1. The numbers in Table 7.1 are obtained by taking the total number of interior labelings for a type of junction (remember that this number includes TABLE labels as well), multiplying this number by the number

TYPE OF SOURCE JUNCTION	NUMBER OF K LABELINGS THESE JUNCTIONS CAN GENERATE	NUMBER OF KA LABELINGS THESE JUNCTIONS CAN GENERATE
L	118 x 1 x 3 = 354	NONE
ARROW	109 x 2 x 3 = 654	109 x 1 x 3 = 327
T	809 x 2 x 3 = 4854	NONE
FORK	1012 x 1 x 3 = 3036	NONE
PEAK	10 x 2 x 3 = 60	10 x 2 x 3 = 60
K	213 x 2 x 3 = 1278	213 x 2 x 3 = 1278
X	563 x 4 x 3 = 6757	563 x 1 x 3 = 1689
XX	152 x 1 x 3 = 456	NONE
MULTI	224 x 1 x 3 = 672	224 x 1 x 3 = 672
TOTALS	18,221	4,026

TABLE 7.1

of ways in which it can form a K junction, and multiplying this number by three (since the obscuring region can have three types of illumination, independent of what the other labels are). Thus, for example, there are 109 ARROW labelings, and each can be used two ways to make a K label of this type (see figure 7.12), so the total number of K junctions due to obscured ARROWs is 109x2x3 = 654. Each ARROW labeling can be used in only one way to form a KA junction, so the total number of these is 109x1x3 = 327.

While I could include these labelings directly in the data base, their number is clearly unwieldy. In any event, I managed to find a way to include the labelings exactly but in a manner somewhat different than those I have been dealing with so far. In order to show this method, I first have to fill in some gaps I left earlier.

7.5 MORE CONTROL STRUCTURE

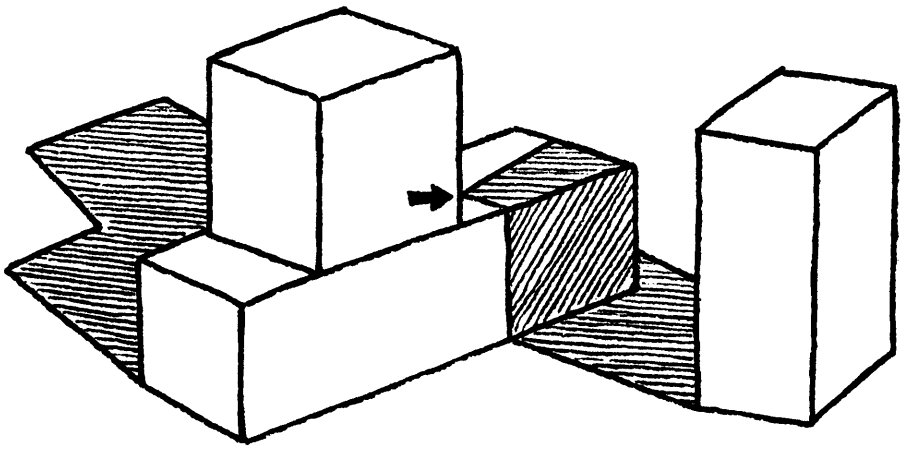
In this section I return again to the main labeling program and describe what happens when the program is unable to label a scene consistently, using the set of labels with which it has been equipped.

The program is written in MICRO-PLANNER, a programming language with automatic back-up facilities (Sussman et al 1971). Before the program begins labeling a junction J, it saves the context of the junction (i.e. the labeling which existed before the program assigned any labels to J). As the program iteratively eliminates the labels which can now be removed because of the new constraints which J adds, it checks at each step to make sure that at least one label remains possible for each line segment. If this number ever goes to zero for any line segment, the program assumes that J is the source of the problem, i.e. that J needed a label that was not in the list assigned to it by the selection rules. When this happens, the program restores the context to the state that existed before it began labeling J, and it marks J to indicate that J cannot be labeled from the normal label lists. Once J has been marked in this manner, it appears to neighboring junctions to be just like a junction which has not been labeled yet, and therefore J imposes no conditions at all on the possible line labels for its neighbors. The program can then continue and as long as two adjacent junctions are not left unlabeled at the end of the program's operation, every line segment can be assigned a value or set of values, just as if every junction had been labeled.

The problem with this arrangement is this: suppose that the program is given a line drawing which has one junction that cannot be labeled from the regular set of junction labelings. Clearly if the program labels this junction last, it will be unable to label the junction and will give the correct result. However, if this junction is labeled before any of its neighbors, then it is, of course, automatically assigned labels from the normal set, for none of the surrounding junctions impose any constraints on it. In this case, one or more perfectly normal junctions in the scene will eventually be marked as unlabelable, and the resulting total labeling for the scene will be invalid. In general, if the bad junction is labeled toward the end of the program's operation, then the total scene labeling is correct, and if the junction is labeled early in the program's operation, the total scene labeling is incorrect.

My first attempt at solving this problem was to label all Ks and KAs last. In many cases the Ks and KAs were then indeed correctly identified as unlabelable from the normal set. However, I managed to come up with a much neater solution which enables the program to generate labels for these otherwise unlabelable junctions.

As before, I have the program label all Ks and KAs last, but this time I modified the labeling procedure. If a junction cannot be labeled from the normal set, instead of marking it unlabelable I generate possible labelings by modifying the line drawing so that it contains equivalent junctions which are not accidentally aligned, and then I label these junctions in the normal manner. Thus, as shown in figure 7.13, if the normal set of junctions is inadequate to label a K, the most reasonable alternative is that the junction is actually an obscured L vertex. Therefore I change the line drawing (saving the original of course) and try to label the new line drawing. This change is equivalent to moving the eye slightly to see what type of junction is obscured, except that since the program is unable to move its eye and therefore does not know what the real vertex type is, it keeps trying various alternatives until one works, or until it hits a default case. In the example shown, the program finds a reasonable interpretation on the first try. If it had not, then the program would next have tried to label the junction as an obscured ARROW, since ARROWs are the next most common type of junction after Ls.



THIS K JUNCTION CANNOT BE LABELED FROM THE NORMAL LABELINGS LIST FOR Ks. THEREFORE THE PROGRAM MODIFIES THE LINE DRAWING, ASSUMING THAT THE K IS REALLY AN OBSCURED L, AND NOW THE LINE DRAWING CAN BE LABELED.

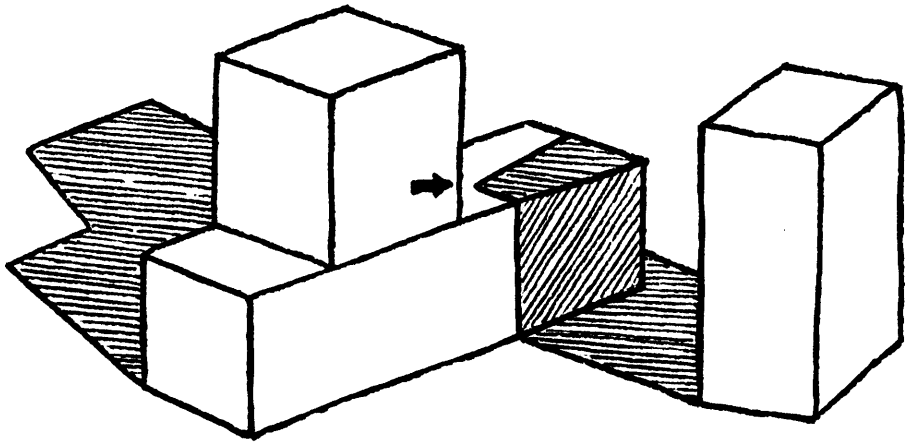


FIGURE 7.13

Notice that the condition for a modification to be reasonable is not as simple as the old condition for a single junction, as illustrated in figure 7.14. The condition for figure 7.14A is that J, J1, and J4 must all be labelable. Before there was no condition joining J1 and J4; if they do not match now, it does not matter whether J0 can be labeled or not because a total labeling would be impossible. This means that the program has to be able to save the context until it has finished checking the labeling of several junctions, and that it should only finalize the modifications when it has proved that every portion of the new line drawing is reasonable. To illustrate further, in figure 7.14B I show the modifications necessary to interpret J0 as an obscured ARROW junction. These modifications create a new junction, and the two junctions, J0 and JJ0 must both be checked; unless both can be labeled consistently this interpretation is impossible.

In fact, I can carry this idea even further. Suppose that a K junction, J0, is actually an accidental alignment, but that since other K and KA junctions in the line drawing have not yet been labeled J0 can be labeled from the normal set of labelings. Later another K, which should be labelable from the normal set cannot be labeled, since the wrong choice

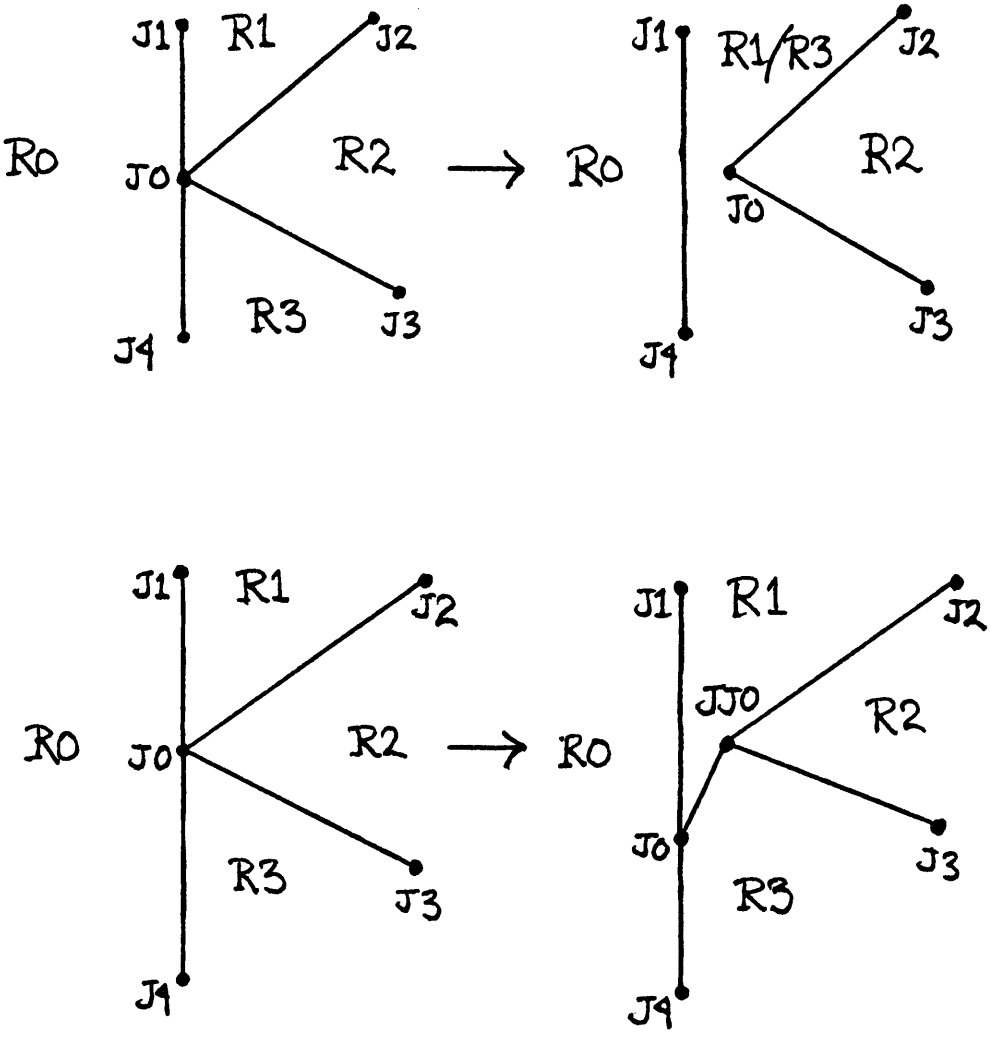

J1
R1
J2
R0
J0
R2
J4
R3
J3
J1
R1/R3
J2
R0
R2
J0
J4
J3
J1
R1
J2
R0
J0
R2
R3
J3
J4
J1
R1
J2
JJ0
R2
J0
J3
R3
J4

FIGURE 7.14

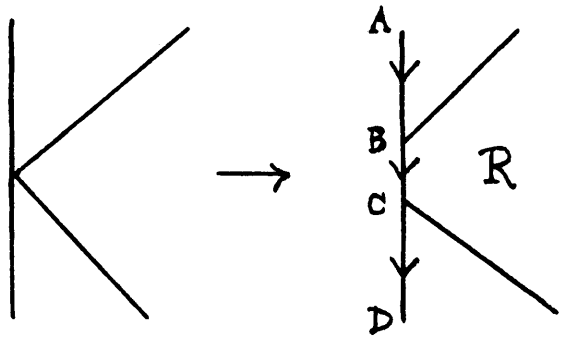
was made for J0. To eliminate this type of difficulty, I require all K and KA junctions to agree, and if they do not agree, the program can back up to any of the K and KA junctions until it has actually tried every combination of interpretations for the junctions. Thus the program should not finalize any of the labels for K or KA junctions until all of them agree.

This solution is still not guaranteed to contain the correct one; the program will be satisfied with the first set of modifications for the K and KA junctions which gives a complete labeling. To be certain of including the correct solution, the program would have to try every combination of interpretations for every K and KA and save all the ones which give complete labelings. Eventually I hope to include this ability when I modify the program to run in the CONNIVER language (McDermott & Sussman 1972); this language has better facilities for developing and saving parallel contexts, whereas MICRO-PLANNER does not. MICRO-PLANNER is oriented toward a tree search model of problem solving where the branches of a solution tree are explored until a correct solution is found. In my case, the problem is that there may be more than one correct solution.

In any case, when I programmed this ability, I lumped a number of junction types together into a default case for two reasons: this lessened the possibility of stopping before getting the desired ("correct") solution, and itenabled the program to run much faster and required a much smaller program than would have been needed if I had included separate machinery for each type of junction. The program tries the possibilities for a K in the following order:

(1) try to label the K from the normal label lists.
(2) try to label the K as an obscured L vertex.
(3) try to label the K as an obscured ARROW vertex.
(4) if all these fail, label the K as two T junctions (see figure 7.15).

The default condition represents the exact opposite of the previous conditions. The two Ts result if instead of moving the eye (by imagination) to see what vertex is behind the obscuring edge, the program moves its eye (by imagination) to completely cover the vertex and eliminate the accidental alignment. Notice that the default condition gives much weaker constraints than could be obtained by trying all the rest of the junction types explicitly. The only relation that must hold for the two T uprights is that



DEFAULT CONDITION

FIGURE 7.15

the region between them (marked R in figure 7.15) have an illumination value which matches both uprights. Nonetheless this is a much stronger condition than is imposed by leaving the junction totally unlabeled and, in addition, the collinear segments (L-A-B, L-B-C, L-C-D in figure 7.15) can all be labeled unambiguously as occluding edges. The information I throw away requires that the two uprights be adjacent segments of the same vertex, where this vertex can presumably be labeled from the normal label lists.

7.6 MISSING EDGES

Missing edges usually occur when the brightness of adjacent regions is nearly the same, since most line finding programs depend heavily on steps in brightness to define edges. I have made no attempt to treat missing edges systematically, but have only included a few of the most common cases in the data base. Clearly missing edge junction labels could be systematically generated by a program merely by listing all possibilities for eliminating one edge from each junction label. This procedure would generate (n-1)x(old number of regular labels) for each junction type (where n is the number of line segments which make up the junction), and clearly this would be a rather unmanageable

number of new labels. The number of new labels could be lessened somewhat by noting that certain types of edges such as cracks are likely to be missed whereas certain other edges such as shadows are relatively unlikely to be missed.

Even if a program such as mine can recognize that a junction must be labeled as having a missing edge, problems still remain about exactly how the line drawing should be completed. This difficulty is illustrated in figure 7.16. Depending on the line segment directions and lengths, the missing edge junction D can be connected to vertex A, vertex B or vertex C, even though the topology of all the line drawings is identical.

The missing edge junctions which are included in the program's data base are all L junctions which result from deleting one of the branches of a FORK junction with three convex edges. Incidentally, in the examples shown in figure 7.16, my program finds each of the given interpretations, but finds no other interpretations, i.e. it finds no interpretations which do not involve missing edges.

TOPOLOGICALLY EQUIVALENT LINE DRAWINGS WITH MISSING LINES:

MOST "REASONABLE" COMPLETIONS:

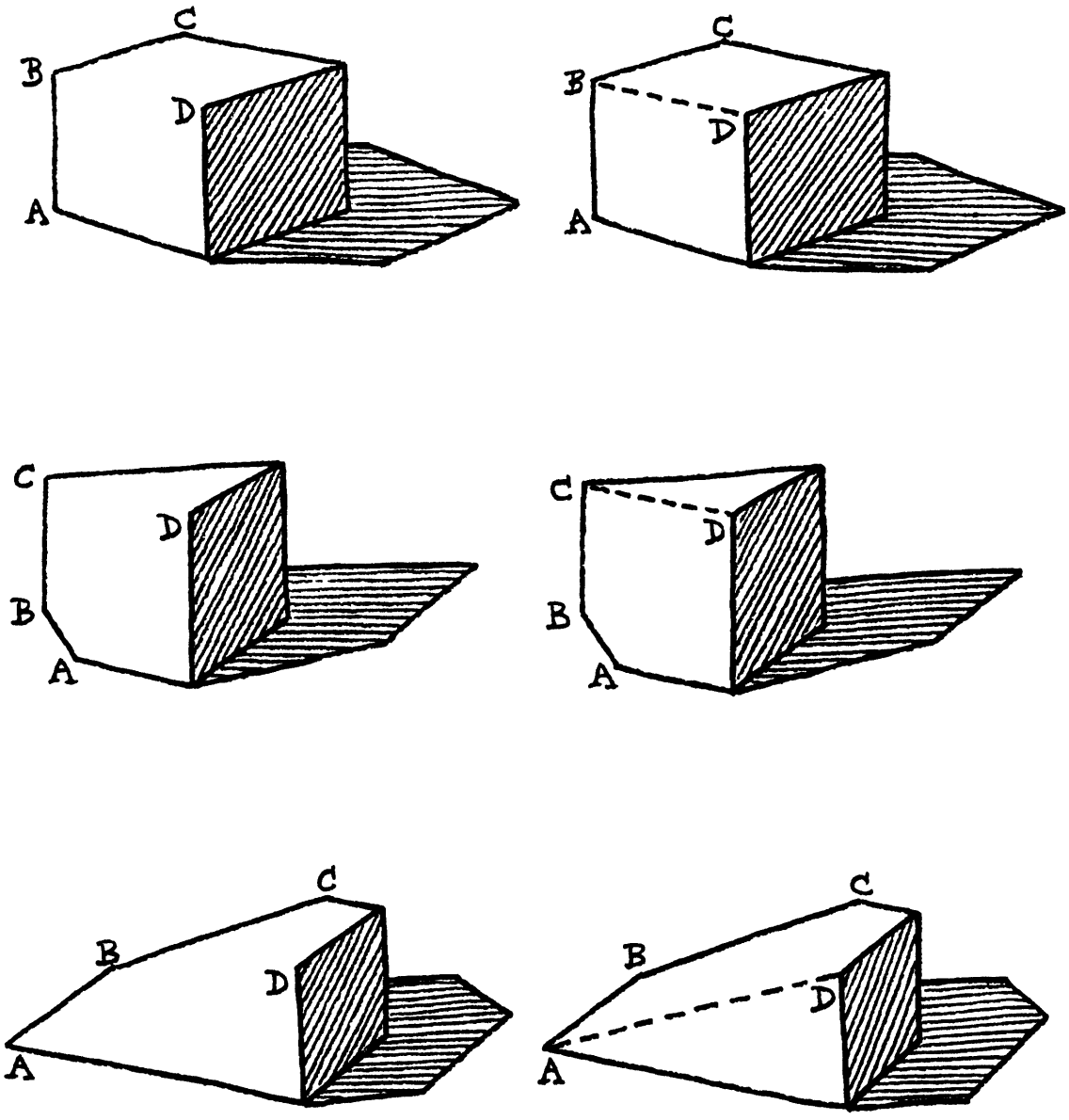


FIGURE 7.16

A rule which can be helpful in removing impossible missing edge interpretations is that if a region is bounded by only one junction which can be interpreted as having a missing edge in that region, then that missing edge interpretation is impossible. (There must be another junction to connect with the missing edge.) A similar rule depends on including the label that the missing edge would have had in each missing edge labeling. In this case, the rule is that not only must there be a pair of missing edge junctions around a region in order for either of them to be possible, but this pair must also match in the label that each gives to the missing edge. One final rule is that the previous rules only hold if the pair of missing edge junctions are not adjacent to one another (i.e. each pair of junctions can be connected by only one straight line).

If more than one edge is missing, then a program requires greater constructive understanding than my program has, although I believe that there are reasonably simple rules which allow a program to solve scenes even if they are as bad as the one shown in figure 7.17. For example, Shirai has demonstrated that the silhouette of a scene contains a great deal of information about where interior lines and junctions can appear (Shirai 1972). Although he does not

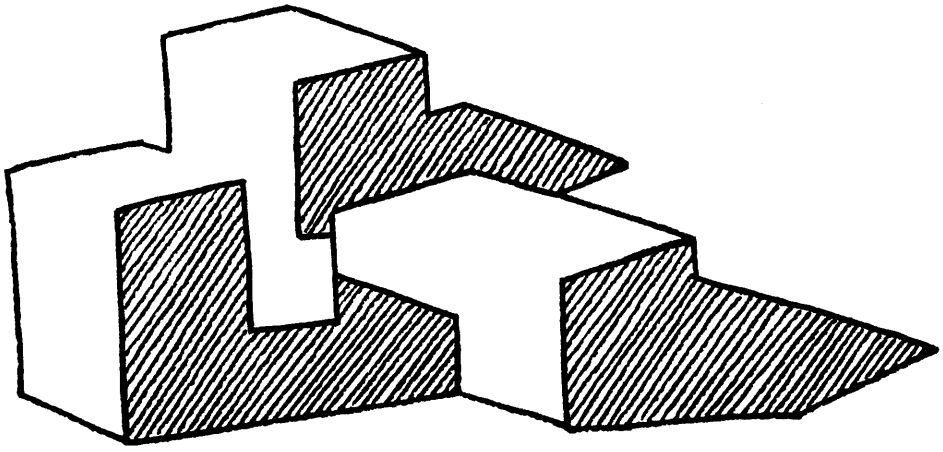


FIGURE 7.17

THREE BLOCKS & SHADOWS
(THIRTEEN EDGES MISSING)

consider scenes with shadows, I believe that the same principles which he uses are applicable for shadowed scenes. Freuder has also written a sophisticated heuristic program which fairly reliably fills in edges missed by our group's line finding programs (Freuder 1971a, 1971b).

7.7 HEURISTICS

As I have mentioned earlier in several places, the program is able to remove junction labels selectively according to a crude probability measure of the relative likelihood of various individual feature interpretations. These heuristics are a poor substitute for foolproof rules; in essence I view the heuristics as an expedient method for handling problems I have not yet been able to solve properly. As I explained in Section 5.4, these heuristics may nonetheless be of considerable value in guiding programs which find sound solutions.

There is not much to say about the heuristics themselves. The ones I am using currently lump all the "unlikely" junction labels into one class, the "likely" ones into another, and simply eliminate all the "unlikely" labels as long as there are "likely" alternatives.

However there are some interesting cases where I have found that I can usually eliminate the unwanted the problem scenes in Section 6.5. Obviously, to solve these cases exactly would require a great deal more programming effort.

Heuristic 1: Try to minimize the number of objects in a scene interpretation.

Implementations:

(1) Make shadow L junction labels (see figure 7.18A) more likely than any other type of L junction.

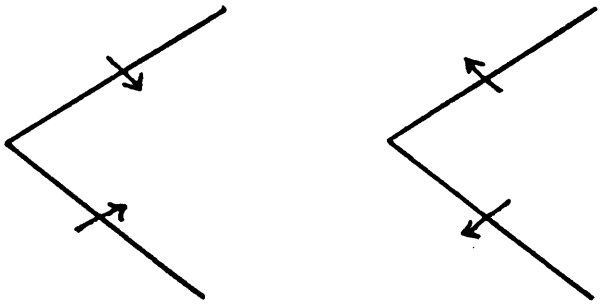
(2) Make labels representing interior TABLE regions more likely than the equivalent labels that do not involve TABLE regions.

(3) If regions can be interpreted either as shadows or as objects, make shadow interpretations more likely.

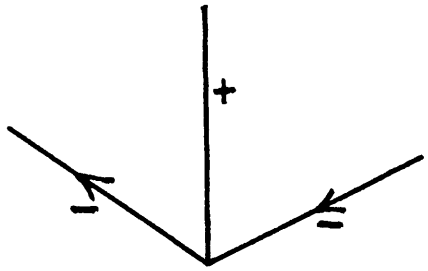
Heuristic 2: Eliminate interpretations that have points of contact between objects or between objects and the TABLE unless there is solid evidence of contact.

FIGURE 7.18

A: SHADOW L JUNCTIONS:



B: CONTACT ARROW JUNCTION:



Implementation: Make ARROW junction labels which have two concave edges and one convex edge (see figure 7.18B) less likely than ARROW labels of other types.

These heuristics select interpretations (1), (2), and (7) from figure 6.10, interpretations A(1) and B(2) from figure 6.12, interpretation (1) from figure 6.14, and interpretations (1), (2), (3), (4), (5), and (9) from figure 6.15.

8.0 REGION ORIENTATIONS

What has obviously been missing from all that I have shown so far is a connection between line segment directions on the retina and possible labelings for these lines. Such a connection is extremely useful if the program is to understand gravity and support. In this chapter I describe approaches to this problem which I have not yet included in my program. There is probably as much work required to properly add the ability to handle direction information as I have already invested in my program. Nonetheless, I believe that this chapter provides a good idea of the work that needs to be done as well as the physical knowledge that these additions will allow one to include in the program.

8.1 LINE LABEL ADDITIONS

To begin with, I investigate the partitioning of each edge type into three subtypes, a technique analogous to the ones I used earlier to divide concave edges into four classes and all edges into types according to their region illumination values. As in the case of occluding edges, the line values are only defined with respect to a reference point and direction, where the usual reference points are

junctions. The three values are:

(1) U (Up) - an edge directed up, away from the TABLE. The reference end is closer to the TABLE than any other points along the edge in the reference direction.

(2) D (Down) - directed downward toward the TABLE. This is the opposite of U, the same edge but with the other end of the line and the opposite direction on the line as references.

(3) P (Parallel) - parallel to the TABLE or in the plane of the TABLE.

Notice that there are some immediate limitations that can now be set on the set of junction labelings:

(1) Any shadow edge or concave edge marked with a "T", i.e. which is in the plane of the table, automatically can have only one direction, P, in this partitioning.

(2) Any junction which has one or more shadow and concave edges labeled "T" must have its edges of other types in the U direction, since the edges at such junctions must

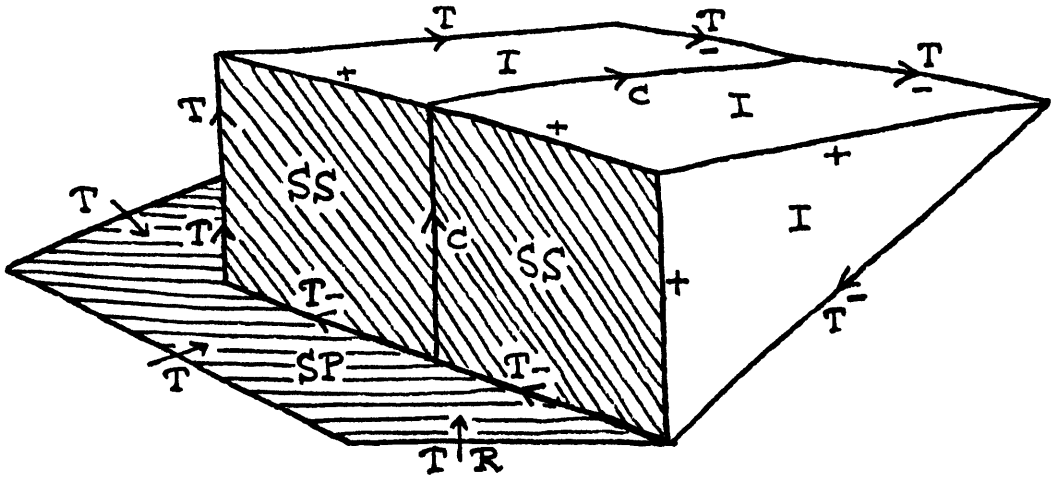
either be in the plane of the TABLE or above this plane.

(3) Two edges which bound the same region and which are parallel or collinear must both have the same direction value, U, D, or P. This fact can be chained through several regions.

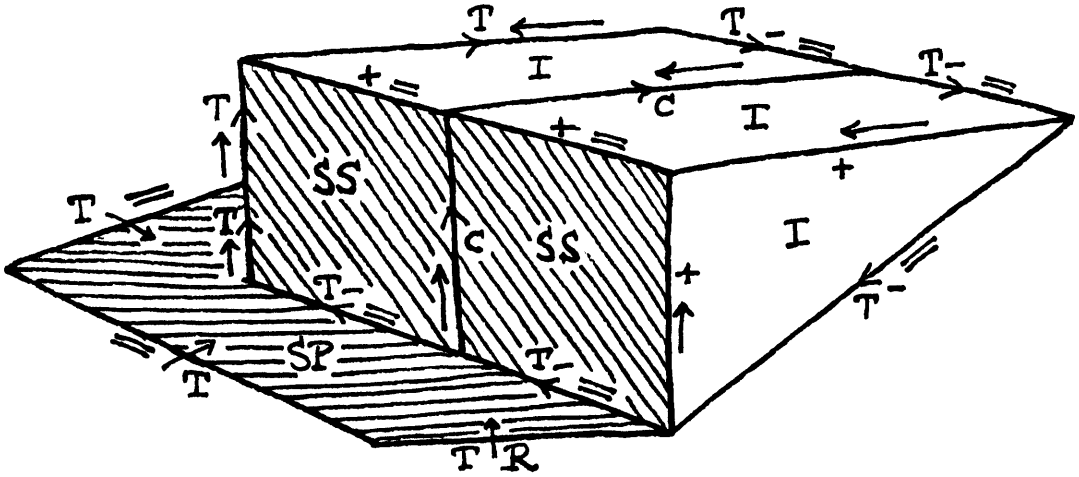
Figure 8.1 illustrates these facts; U is indicated by placing an arrow along the side of a line segment pointing in the Up direction.

Notice that these rules also allow a program to find horizontal surfaces, an important part of the notion of support. A horizontal surface can be defined in this system of notation as any region bounded by two or more edges which are both marked P (||) and which are not parallel to each other or collinear. Moreover, any edges which are in the plane of a horizontal surface can then also be marked as parallel to the TABLE, regardless of the directions of these lines on the retina. Finally, any junctions which bear a relationship to a horizontal surface, analogous to the one that I mentioned earlier for junctions which had segments in the plane of the TABLE, can similarly have their other segments labeled U. Figure 8.2 illustrates these points.

WITHOUT EDGE DIRECTION INFORMATION THE FOLLOWING LINE DRAWING IS LABELED AS SHOWN:



IT ALSO HAS A UNIQUE LABELING IF LINE DIRECTION INFORMATION IS INCLUDED:



↑ = Up ↓ = Down ||| = Parallel

FIGURE 8.1

USING THE RULES DESCRIBED SO FAR EXCEPT THE HORIZONTAL SURFACE RULE, THIS SCENE CAN BE LABELED:

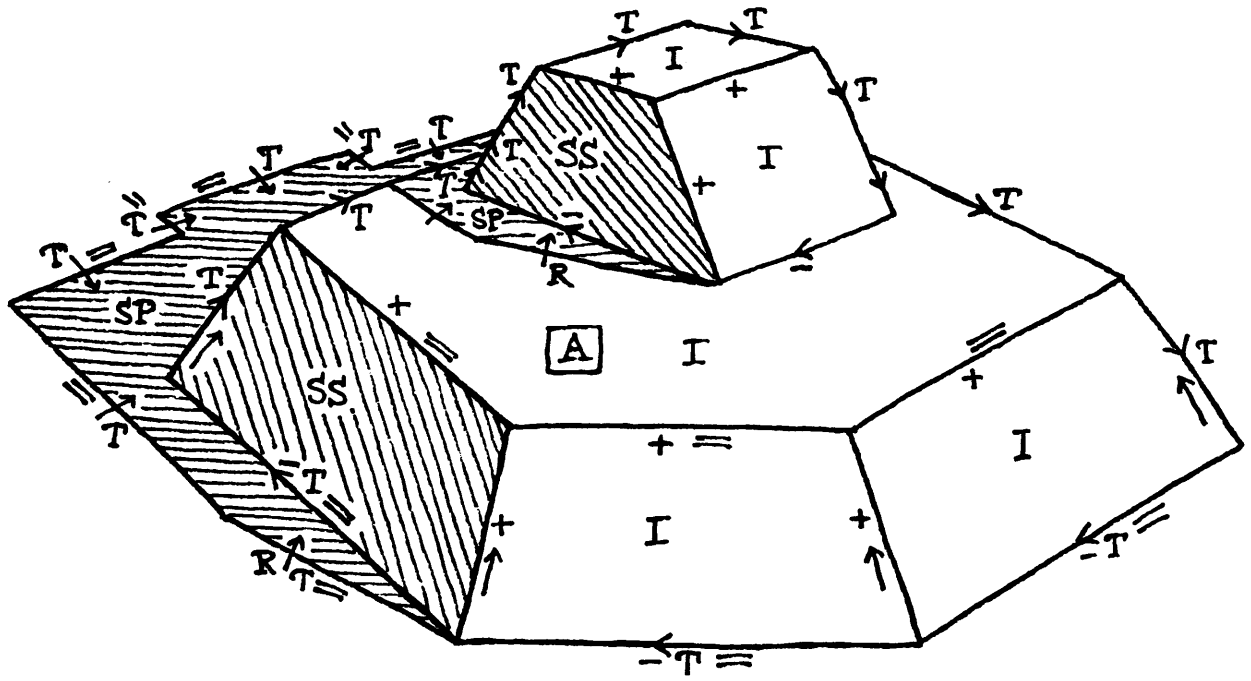


FIGURE 8.2

REGION [A] CAN BE NOW FOUND TO BE HORIZONTAL, SO THAT ALL EDGES WHICH ARE PART OF [A] CAN BE LABELED PARALLEL; ONLY ADDED LABELS ARE SHOWN:

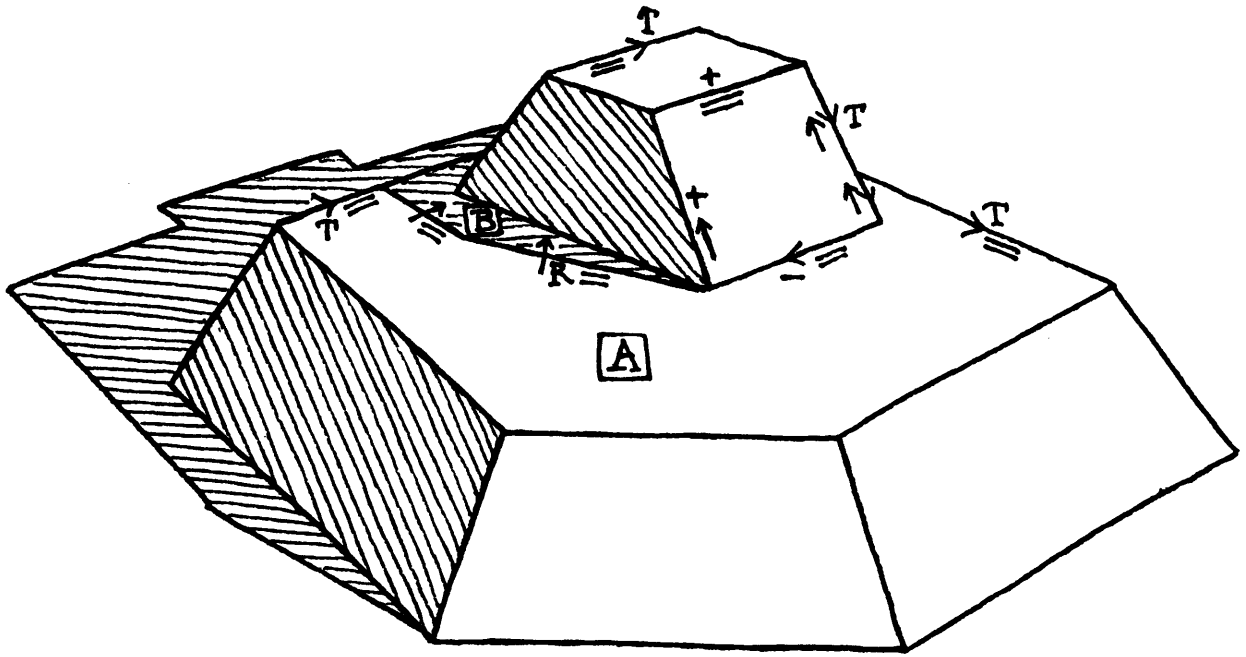


FIGURE 8.2

[B] CAN NOW BE MARKED AS HORIZONTAL, AND AS SHOWN BELOW, [C] CAN BE FOUND TO BE HORIZONAL ALSO:

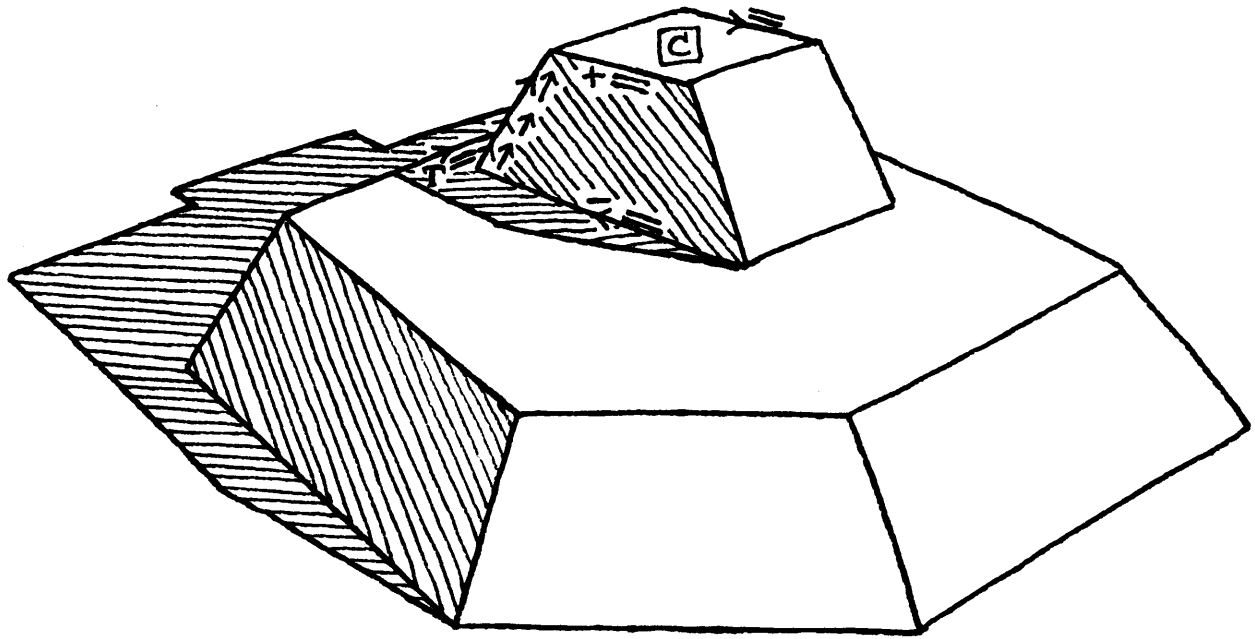


FIGURE 8.2

These rules are not particularly helpful when there are no parallel edges; it is possible to chain some values in the absence of parallel edges and horizontal surfaces, but generally such chaining cannot be carried very far. Depending on the way that edges deviate from parallel, it is sometimes possible to assign an Up direction. (In some of the figures which follow I have not marked the lines with their normal labels, but have only included the direction labels for clarity.) See figure 8.3, and note that since edge L-A-B is not parallel to L-C-D, I can mark L-C-D with an Up direction as shown. This means that since L-E-F is parallel to L-C-D, it can also be marked with an Up direction.

8.2 AN EXAMPLE

Using the methods I have already discussed plus one other piece of new information, I can show how to eliminate some labelings for a line drawing if I know the line segment directions. To see how these can help, consider again the example I showed in figures 5.10 and 5.11, as illustrated now in figure 8.4A. Because L-A-B is parallel to L-C-D and L-B-E is parallel to L-D-F, R1 must be horizontal, assuming that

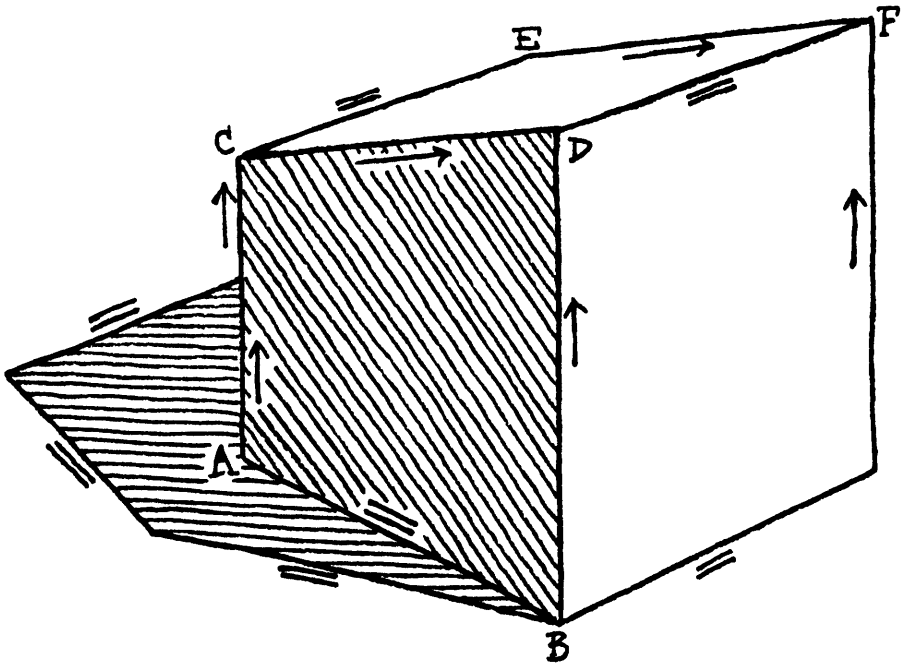
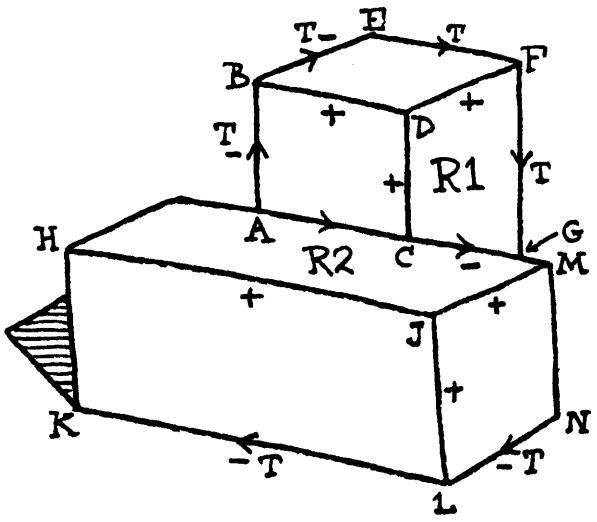


FIGURE 8.3

A:



B:

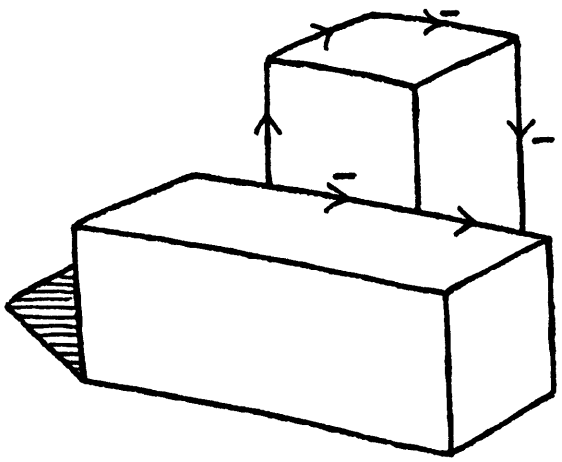


FIGURE 8.4

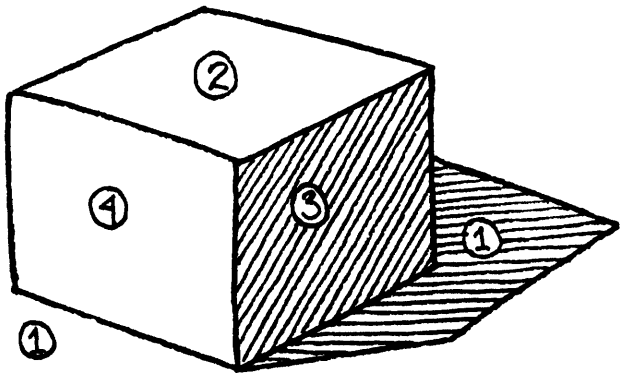
the labeling shown is true. By the same kind of reasoning R2 must be horizontal. Now the additional rule is: two horizontal regions can only be separated by crack, shadow or obscuring edges. Therefore, the labeling shown is impossible, since L-C-G is a concave edge, and consequently cannot separate two horizontal regions. Similarly I can eliminate the labeling shown in figure 8.4B.

8.3 GENERAL REGION ORIENTATIONS

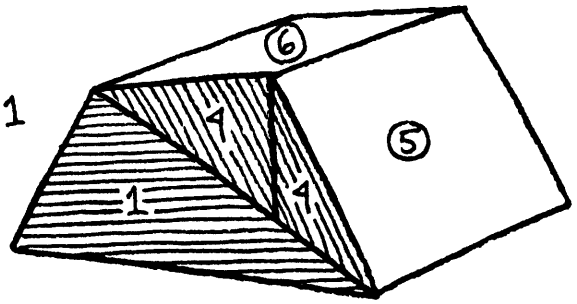
In this section I define a quantization scheme which assigns to each visible region one of sixteen values. The regions are named in as sensible and simple a manner as I could devise, and are defined with respect to a coordinate system which is itself defined by the TABLE surface and the position of the eye viewing the scene. The region orientation values are each shown in figure 8.5; I assume that this figure will serve as an adequate specification for the meaning of the different orientation values. If the scene is moved with respect to the eye or vice-versa, then the region values (except Table and Horizontal) may change, and regions previously invisible may become visible. Thus the region orientation values are not inherent properties of the surfaces, but are only defined with respect to a

[ASSUME THAT EDGES WHICH ARE VERTICAL ON PAPER ARE VERTICAL IN THE SCENES.]

① TABLE
② HORIZONTAL
③ FRV
(FRONT RIGHT VERTICAL)
④ FLV
(FRONT LEFT VERTICAL)



1 TA
4 FLV
⑤ FRU
(FRONT RIGHT UP)
⑥ BLU
(BACK LEFT UP)



1 TA
3 FRV
⑦ FLU
(FRONT LEFT UP)
⑧ BRU
(BACK RIGHT UP)

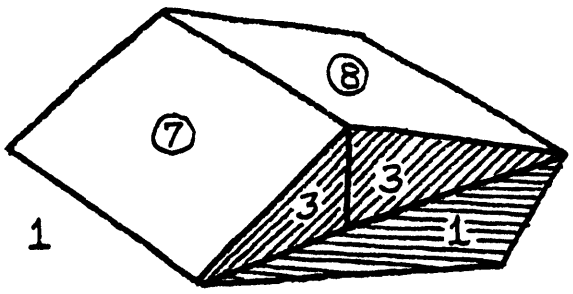
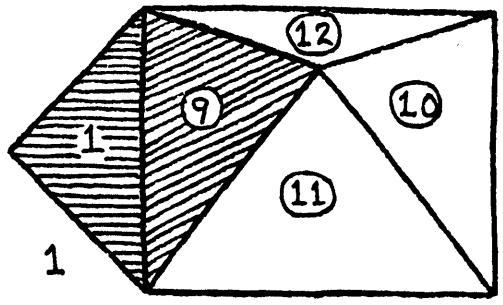
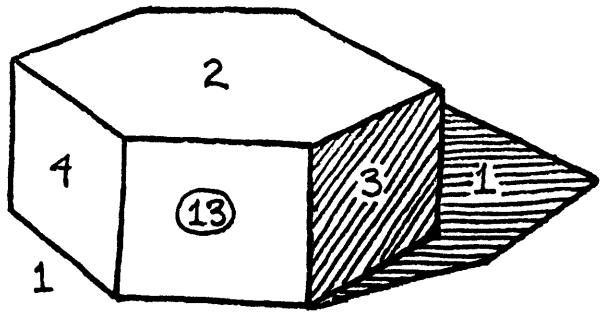


FIGURE 85

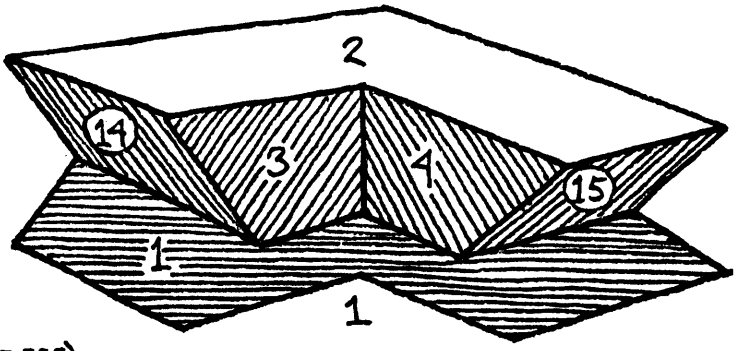
1 TA
(9) LU (LEFT UP)
(10) RU (RIGHT UP)
(11) FU (FRONT UP)
(12) BU (BACK UP)



1 TA
2 H
3 FRV
4 FLV
(13) FV
(FRONT VERTICAL)



1 TA
2 H
3 FRV
4 FLV
(14) FLD
(FRONT LEFT DOWN)
(15) FRD
(FRONT RIGHT DOWN)



1 TA
2 H
3 FRV
4 FLV
(16) FD (FRONT DOWN)

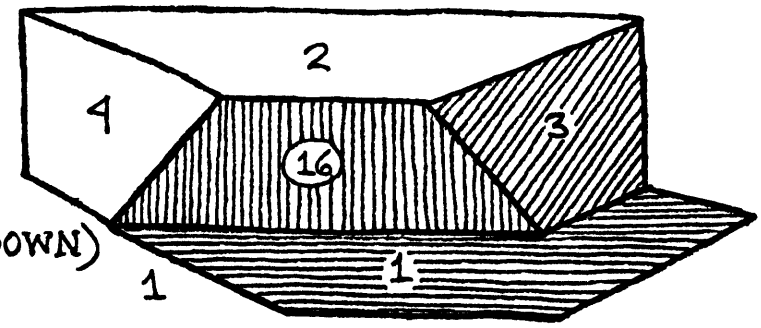


FIGURE 8.5

particular eye-table arrangement.

If a region R1 is type FRV (Front Right Vertical) and an edge separating this region from region R2 is a shadow edge, then region R2 must also be type FRV (see figure 8.6A). The problem is not quite so simple when the other edge types are involved. To give the flavor of what I would like to be able to do in general, note that if an edge separating R1 and R2 is vertical on the retina, and R1 appears to the right of R2 on the retina, then R2 can only be type FLV or type FV or type FRV (see figure 8.6B).

8.4 GENERAL LINE DIRECTIONS

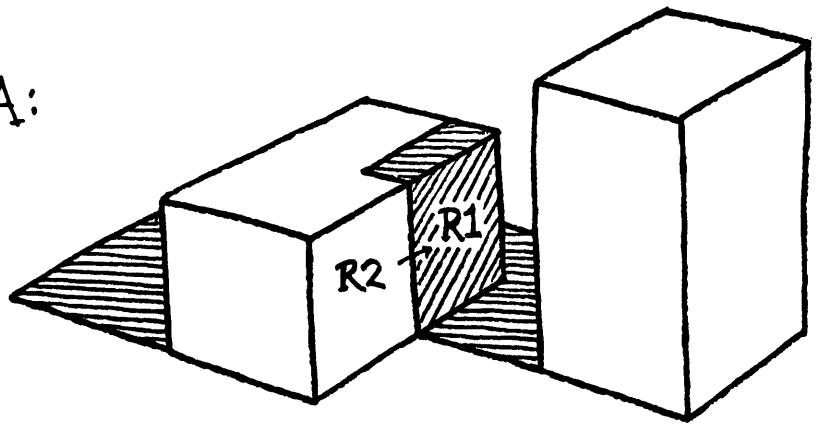
Before I can carry out this type of association in general, I must

(1) define line directions on the retina and

(2) define line directions in the scene domain with greater precision, and

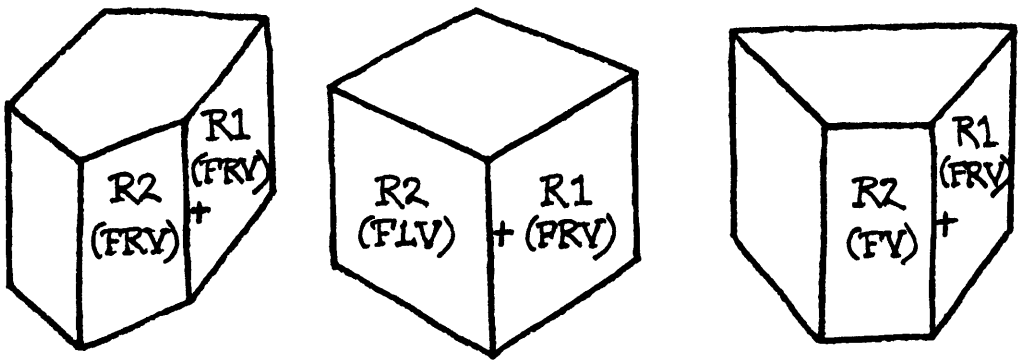
(3) show how to find the scene direction values, given a labeled line drawing and the retinal line directions.

A:



R1 AND R2 ARE BOTH FRV

B:



THREE POSSIBILITIES FOR NORTH/SOUTH CONVEX EDGE BOUNDING AN FRV REGION (R1) ON ITS LEFT: FRV, FLV, OR FV.

FIGURE 8.6

Throughout this chapter I assume that the eye is far enough away from the scene so that vertical edges in the scene project into North/South lines on the retina. Since the definition of North/South edges includes a tolerance angle ϵ, the eye does not need to be at infinity for this condition to hold. By the same reasoning I assume that parallel edges can be recognized without resort to perspective or vanishing point considerations.

First I define the retinal line directions in terms of compass points as shown in figure 8.7.

Next , in figure 8.8, I define the names for the directions of lines in the scene by showing examples for each type possible direction. These names resemble the names for region orientations, but I will always use lower case letters in referring to the line names and will use upper case letters when I refer to the region names.

Now to make the connections between the retinal and scene line directions, note that I can catalog all the possible edge directions in the scene domain which can map into each of the direction values on the retina. As an example of how to do this, in figure 8.9 I show all the edge

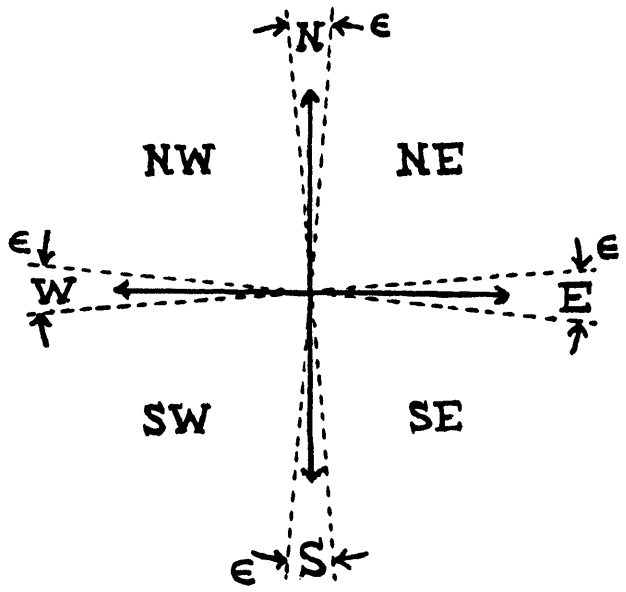


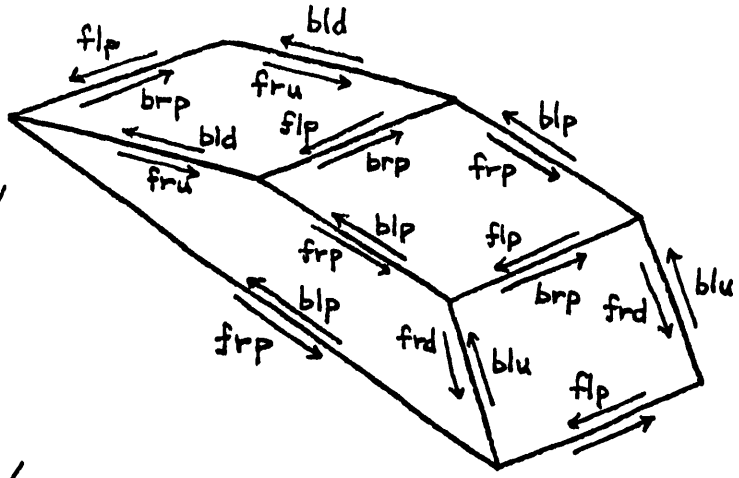
FIGURE 8.7

flp/brp
(front left parallel/
back right parallel)

frp/blp
(front right parallel/
back left parallel)

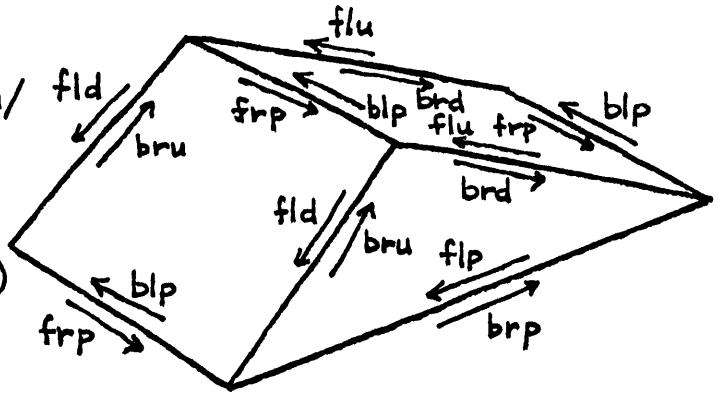
fru/bld
(front right up/
back left down)

frd/blu
(front right down/
back left up)



fld/bru
(front left down/
back right up)

flu/brd
(front left up/
back right down)



lp/rp
(left parallel/
right parallel)

fp/bp
(front parallel/
back parallel)

vu/vd
(vertical up/
vertical down)

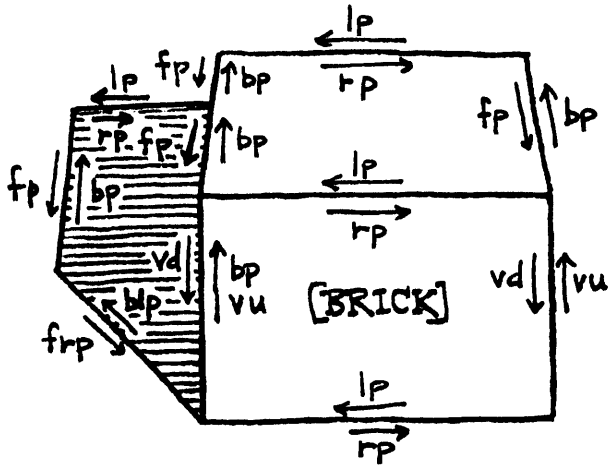
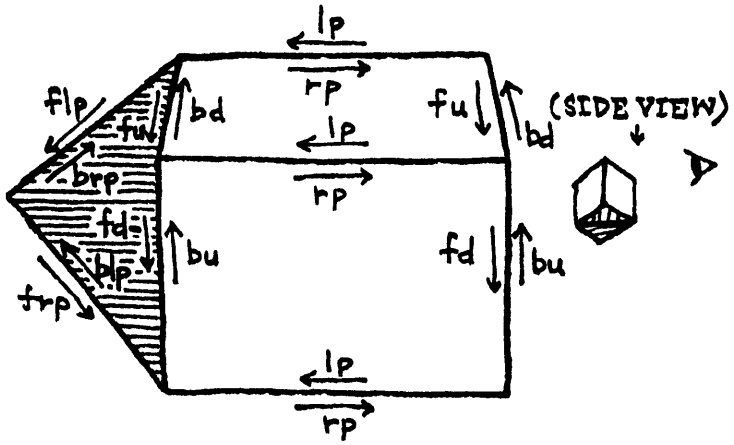


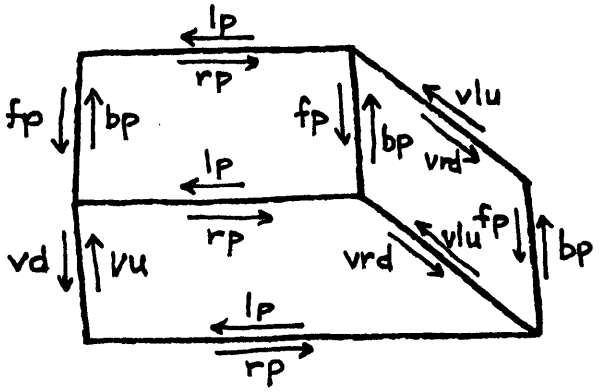
FIGURE 8.8

fu/bd
(front up/
back down)

fd/bu
(front down
back up)



vlu/vrd
(vertical left up/
vertical right down)



vld/vru
(vertical left down/
vertical right up)

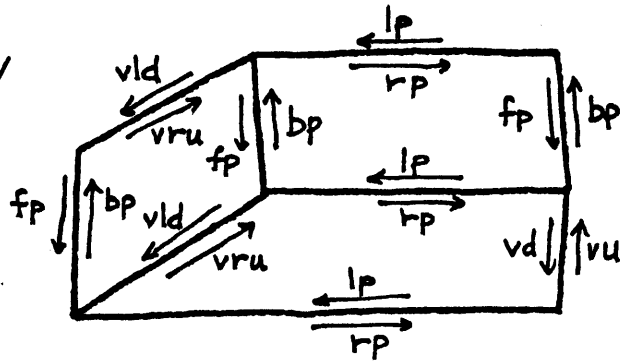


FIGURE 8.8

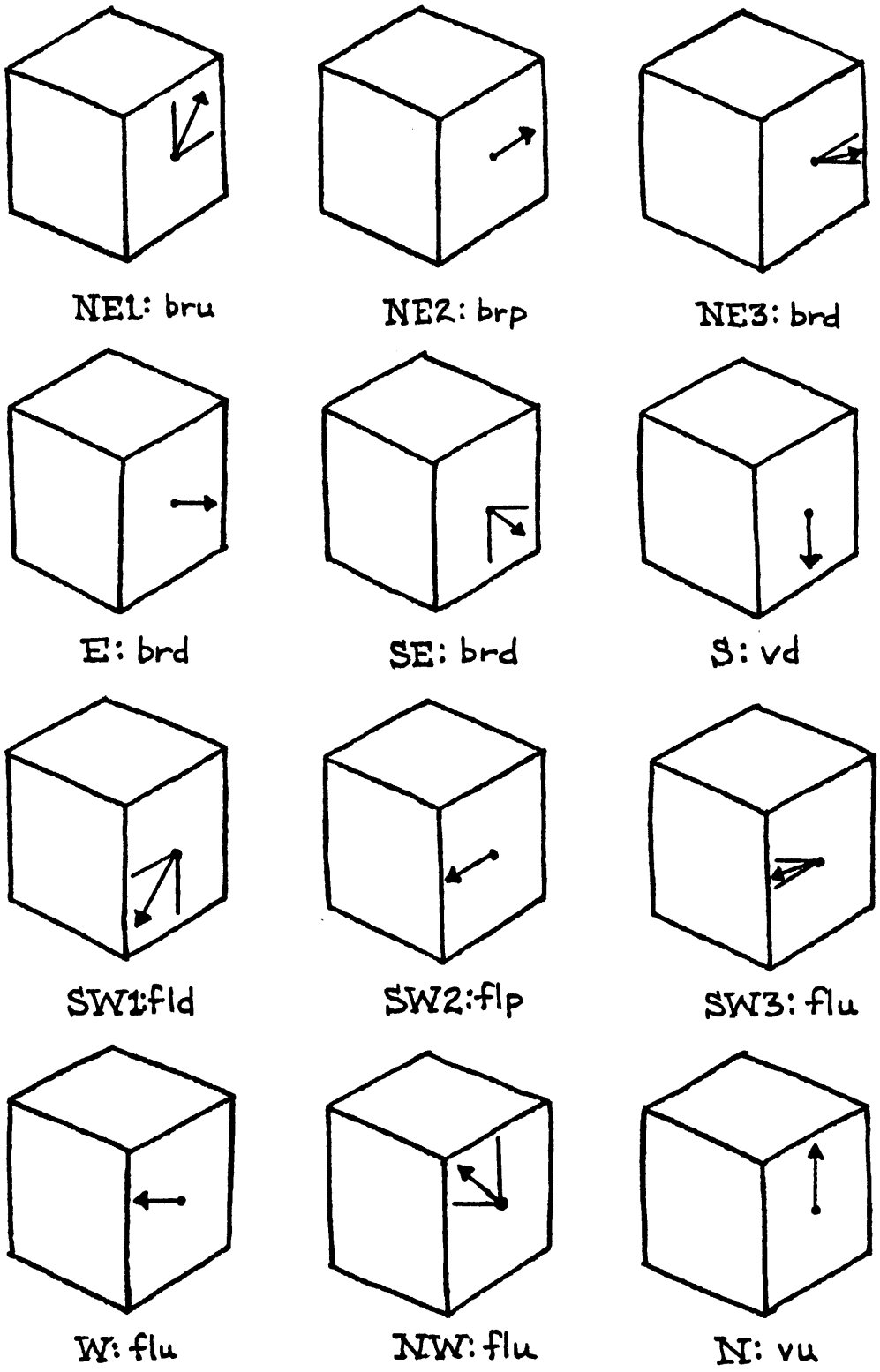

NE1: bru
NE2: brp
NE3: brd
E: brd
SE: brd
S: vd
SW1:fld
SW2:flp
SW3: flu
W: flu
NW: flu
N: vu

FIGURE 8.9

directions possible for an edge which bounds a type FRV region. The diagrams in this figure show that an NE (Northeast) line on the retina which bounds a type FRV region can be an edge of types bru, brp, or brd, that an E (east) line on the retina which bounds a type FRV region can only be caused by a type brd edge, etc. Table 8.1 is a summary of the types of scene edges which can cause lines of each type on the retina, arranged according to the types of regions that each edge can bound.

Now to tie everything together, notice that an edge can only separate two regions if the edge could have the same direction in both regions bounding the edge. Therefore, to find all the region pairs that an N (North) edge (as seen on the retina) could separate, look down the N column in Table 8.1 and find all the pairs of regions which can share an edge which points in a particular direction. A north pointing edge can thus separate any of the following pairs of region types (this is not a complete list):

((TA TA) (H H) (TA LU) (H LU) (H RU)
(RU TA) (RU H)
(FRV FRV) (FRV FLV) (FRV FV)
(FLV FRV) (FLV FV) (FLV FLV)

TABLE 8.1	DIRECTION OF LINE ON RETINA							PAGE 268
REGION TYPE ↓	N	NE	E	SE	S	SW	W	NW
H or TA	bp	brp	rp	frp	fp	flp	lp	blp
FU	bu	bru	rp	frd	fd	fld	lp	blu
FV	vu	vru	rp	vrd	vd	vld	lp	vlu
FD	fu	fru	rp	brd	bd	bld	lp	flu
FRU	bu	bru brp brd	brd	brd vrd frd	fd	fld flp flu	flu	flu vlu blu
FRV	vu	bru brp brd	brd	brd	vd	fld flp flu	flu	flu
FRD	fu	fru, vru brp, bru brd	brd	brd	bd	bld, vld fld, flp flu	flu	flu
RU	bp	brd	brd	brd vrd frd	fp	flu	flu	flu vlu blu
BRU	bd	brd	brd	brd, vrd frd, frp fru	fu	flu	flu	flu, vlu blu, blp bld
BU	bd	brd	rp	fru	fu	flu	lp	bld
BLU	bd	brd, brp bru, vru fru	fru	fru	fu	flu, flp fld, vld bld	bld	bld
LU	bp	bru vru fru	fru	fru	fp	fld vld bld	bld	bld
FLU	bu	bru vru fru	fru	fru frp frd	fd	fld vld bld	bld	bld blp blu
FLV	vu	fru	fru	fru frp frd	vd	bld	bld	bld blp blu
FLD	fu	fru	fru	fru, frp frd, vrd brd	bd	bld	bld	bld, blp blu, vlu flu

(FV FV) (FV FLV) (FV FRV)
(LU H) (LU RU) (LU LU)
(RU H) (RU LU) (RU RU)
(BLU BRU) (BRU BLU))

Not all these pairs can be separated bythe same types of edges; shadows and cracks can only separate regions with the same orientation values, and convex edge pairs become concave edge pairs if the order of the pairs is reversed. For example, a North line separating regions with orientation values (FLV FRV) represents a convex edge (where the ordering of the regions is in a clockwise direction), but if the orientation values are (FRV FLV) for a North line, this must represent a concave edge. This fact is illustrated in figure 8.10.

If the Up/Down/Parallel designations are also included in the regular labeling program, then it is possible to make even finer distinctions. Table 8.2 shows some of the lists of region orientation pairs which can be assigned to lines having the indicated labels and directions.

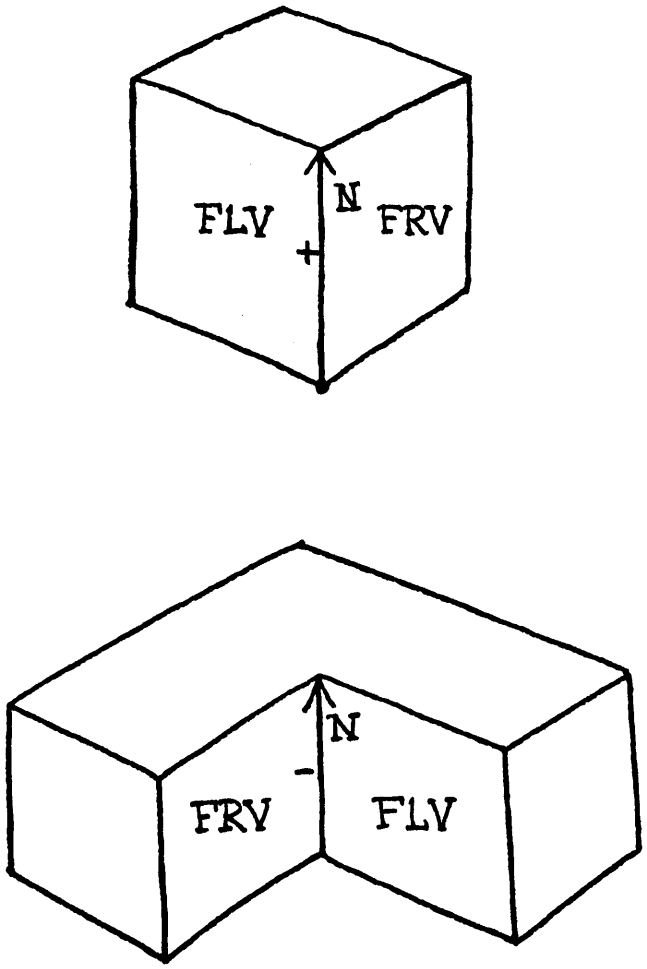


FIGURE 8.10

LINE DIRECTION	LABEL	POSSIBLE REGION PAIRS
N	PLUS-U	((FU FRU) (FRU FRU)(FLU FLU) (FLU FU)(FLU FRU) (FV FRV) (FRV FRV)(FLV FLV) (FLV FV)(FLV FRV) (FD FRD) (FRD FRD)(FLD FLD) (FLD FD)(FLD FRD))
N	PLUS-P	((H RU) (RU RU)(LU LU) (LU H)(LU RU))
N	PLUS-D	((BRU BRU)(BLU BLU) (BU BRU) (BLU BU)(BLU BRU))
N	(concave) M-D or M3-D or OCRM-D or OCLM-D	((BRU BU)(BRU BLU) (BU BLU) (BLU BLU)(BRU BRU))

TABLE 8.2

A program can use this information in the following ways:

(1) If there are ambiguities remaining after the regular labeling program has finished, pick a single labeling, assign region values using the lists shown in part in Table 8.2, and see whether this labeling can represent a possible interpretation; if the interpretation is not possible, then the program will be unable to assign orientation values to every region, very much like the case earlier in this chapter where a concave edge could not separate two horizontal surfaces.

(2) Region illumination values can be tied in with the region orientation values. For example, if a scene is lit from the left, and the light-eye angle is less than 90 degrees (see figure 8.11; the light-eye angle is the angle between the projections of the eye and the light onto the plane of the TABLE, as measured from the center of the scene), then a region cannot be labeled simultaneously as orientation type FLV and illumination type SS (Self-Shadowed).

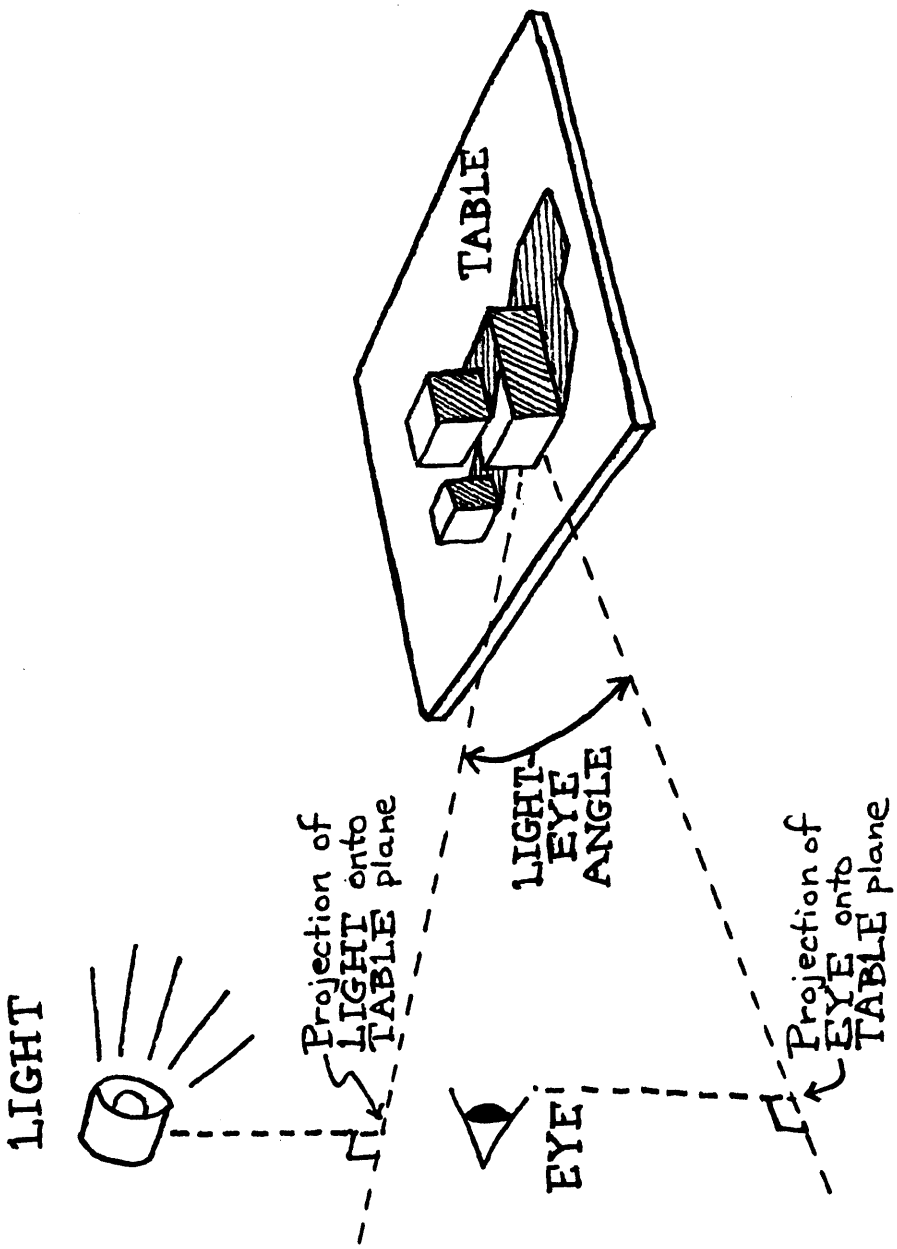


TABLE
LIGHT
Projection of LIGHT onto TABLE plane
LIGHT EYE ANGLE
EYE
Projection of EYE onto TABLE plane

FIGURE 8.11

(3) All these facts provide a neat way to integrate stereo information into a scene description. For example, as shown in figure 8.12, if an edge is truly vertical (type vu) then it must appear as N (North) in any retinal projection of a stereo system. However an edge which is of type bp (back parallel) can appear to be N on the retina because of the particular placement of the eye with respect to the scene. If the eye is shifted slightly to the right, this edge will now appear to point NE (Northeast) and if the eye is shifted to the left, the edge will appear to point NW (Northwest). Clearly this knowledge would enable a program to much more severely restrict the region orientation pairs, and consequently the labelings, that can be assigned to a line drawing of a scene. Without the region and edge orientation formalisms (or other similar formalisms) it is not possible for a program to understand this stereo information, although one could undoubtedly find ad hoc ways of using the information.

(4) All the possibilities for region orientations can be generated by the function I called ILLUMINE in Section 4.1. For each labeling which the program finds, ILLUMINE can select region pairs according to the line directions and line labels, and build up a set of region orientation values in

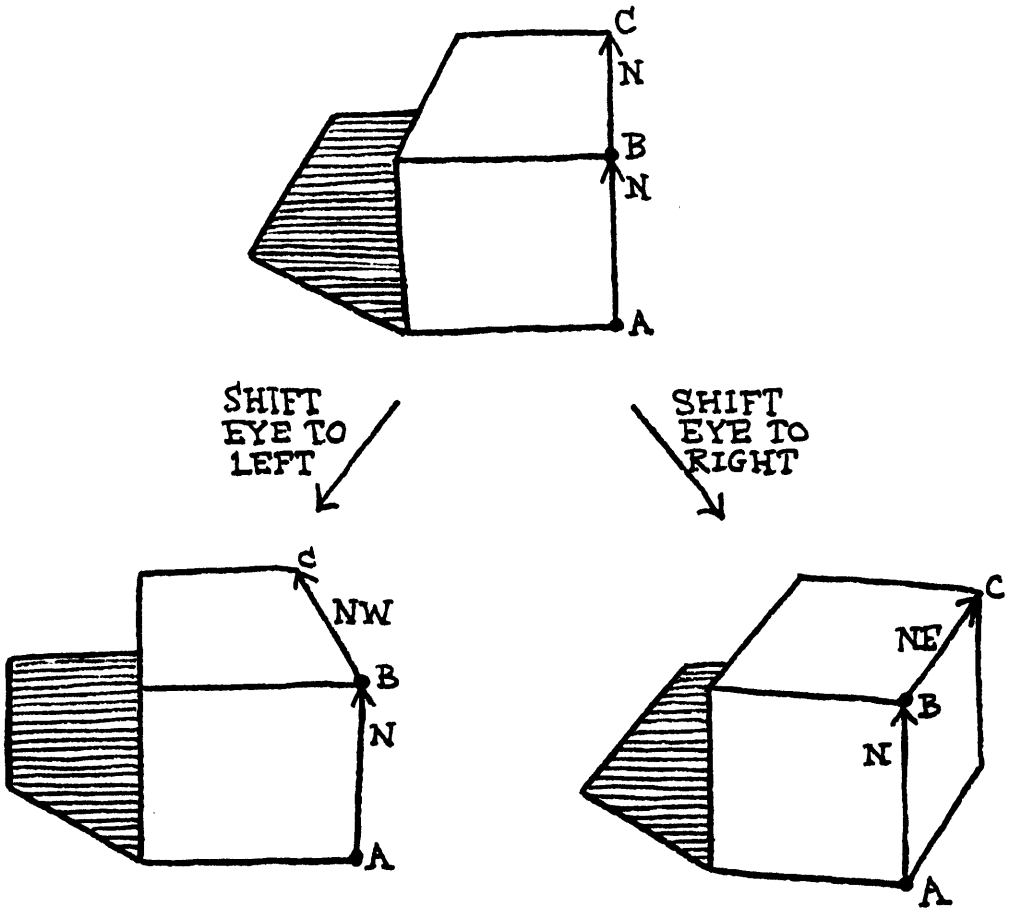


FIGURE 8.12

exactly the same manner that ILLUMINE builds up sets of region illumination values. The difference is that there are far too many region orientation values in general to possibly include them in precompiled form; the values must be generated from the greatly reduced set of possibilities that remain after the regular labeling program has completed its work. The reason why there are so many possibilities is that there are so many possible region orientations. Each edge can potentially have 16x16 = 256 region orientation pairs as opposed to the nine possible region illumination pairs.

8.5 SUPPORT

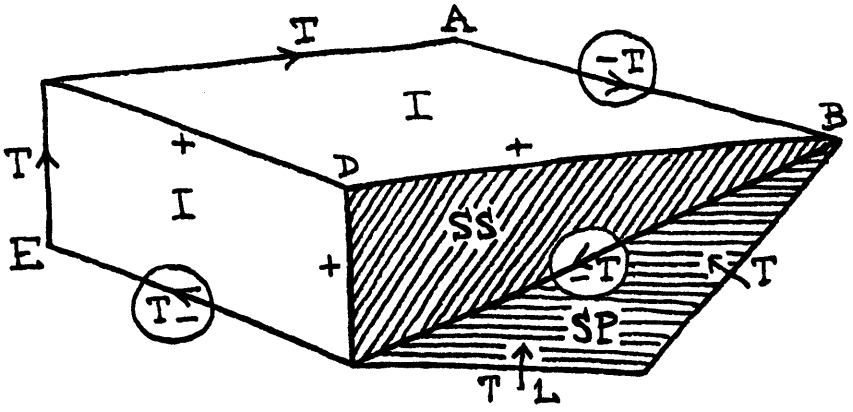
Using the region orientation values, I can now define the set of edges along which support must hold, the set of edges along which support can hold, and the set of edges along which support cannot hold. By support I mean what is commonly termed either resting on or leaning on.

To start with, I can eliminate from consideration any edges which are shadows, convex edges, obscuring edges, or concave edges made up of one object or of three objects, and I can say for certain that support is exhibited along any concave edge which has the TABLE as a bounding region. In

addition edges labeled as "leaning" (see Section 7.1) point to places where support relations must hold, although support does not hold along the leaning edges themselves, since these are either obscuring or convex edges.

The important fact is that these edges exhibit support regardless of their directions on the retina, so that there is no problem with edges such as L-A-B in figure 8.13. The best previous rules to find where support holds in a scene (see Winston 1970) are not able to handle cases like this; Winston's rules were biased toward finding ARROWs, Ks and Xs which have vertical (or at least upward pointing) lines as do all of the cases in figure 8.14 (this figure is a copy of figure 2-41 from Winston 1970). In addition, Winston's rules failed to find one support relation for the leaning block; his rules assumed that objects would be supported by face contact only.

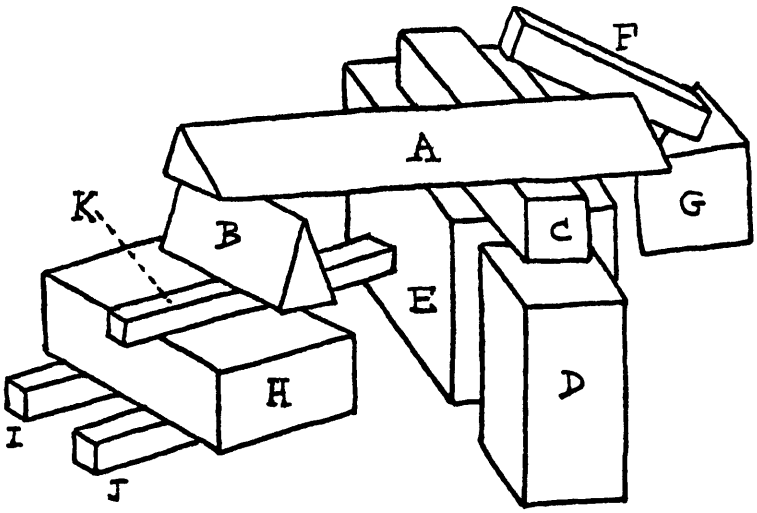
Although my program can find support in cases like figure 8.13, it is important to note that, in general, it is not possible to use my regular labelings and line directions alone to find which edges exhibit support and which do not. Suppose that on the basis of the frequency of crack edges like the ones shown in figure 8.15A I decided to label as



(SUPPORT RELATIONS ARE CIRCLED)

FIGURE 8.13

FIGURE 8.14



OBJECT	SUPPORTED-BY	IN-FRONT-OF
A	B,C	F,G
B	K	—
C	D,E	—
D	—	E
E	—	—
F*	E	—
G	—	—
H	I,J	—
I	—	—
J	—	—
K	H	E

*PROGRAM MISSES G AS SUPPORT FOR F

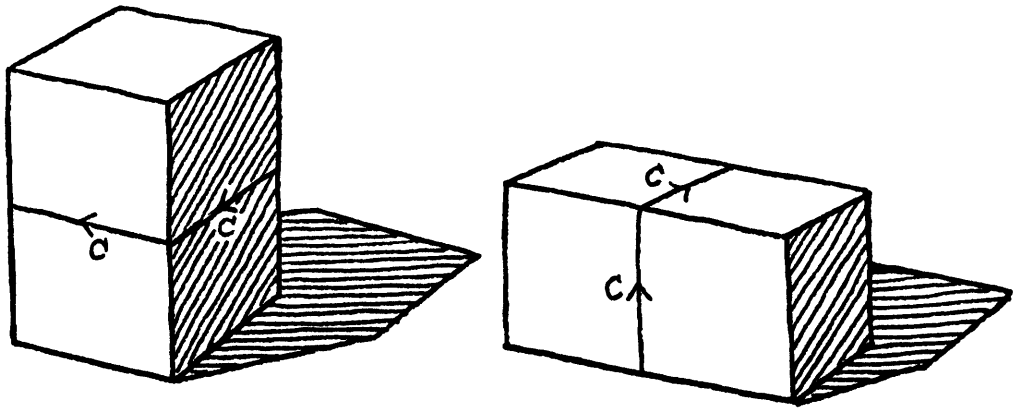


FIGURE 8.15A

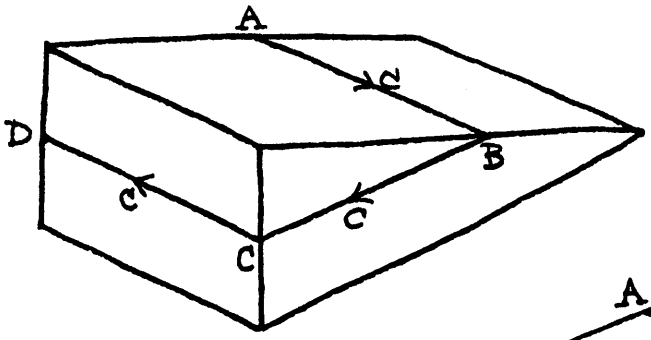


FIGURE 8.15B

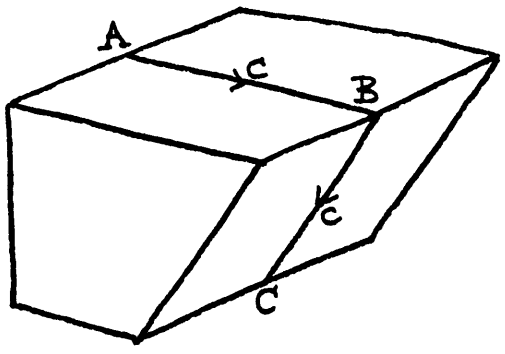


FIGURE 8.15C

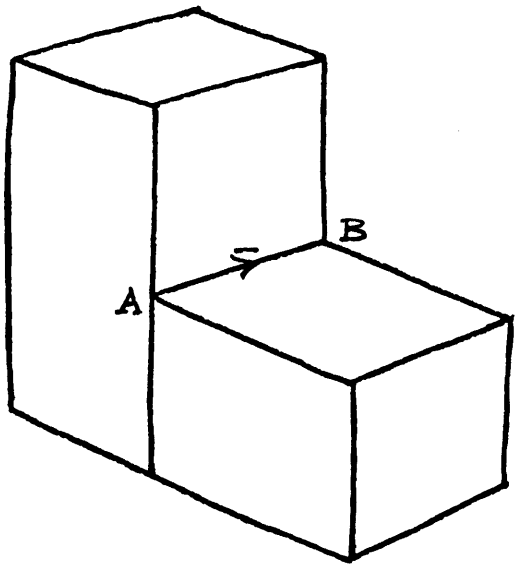


FIGURE 8.15D

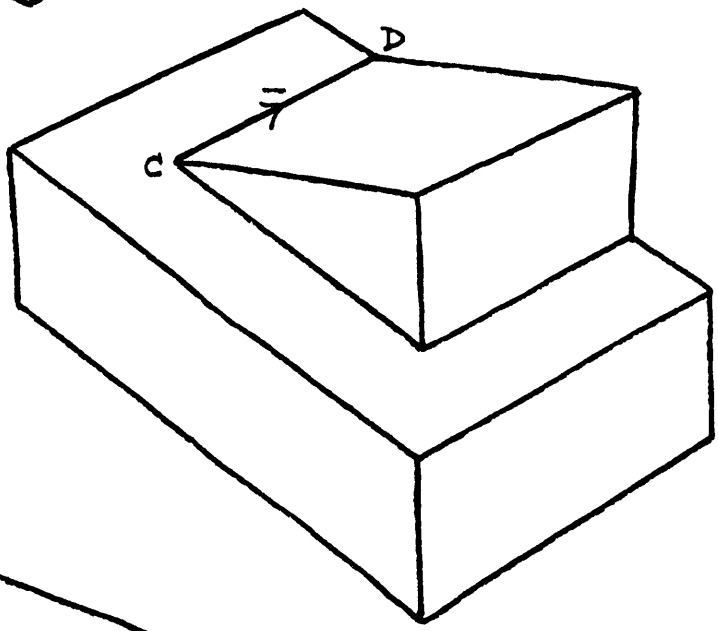


FIGURE 8.15E

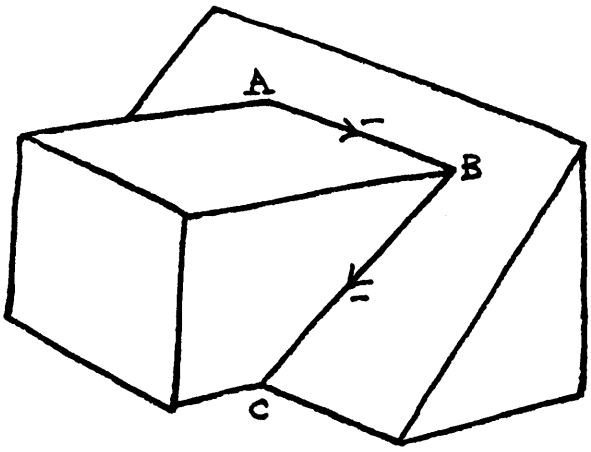


FIGURE 8.15F

Supporting/crack edges ones in which the arrow of the crack label points SW, W, or NW, and to class all the others together as being crack edges without support relations. Then in figure 8.15B edges L-B-C and L-C-D would be correctly marked but L-A-B would not. I could patch up the rule by saying that if support holds for one non-collinear line in an X junction it must hold for the other non-collinear line of the X as well. Unfortunately this rule causes the program to assert that support holds between the two objects in figure 8.15C, since support would be transferred by the rule from L-B-C to L-A-B.

Similarly, for concave edges I cannot use line directions and the direction of the arrow on the label to define support. As an example, observe that while L-A-B in figure 8.15D does not exhibit support, L-C-D in figure 8.15E does.

Region orientation values can help to avoid these problems, at least for some cases. (There are some, cases such as the one in figure 8.15F , where I do not know whether to say that support holds along L-A-B and L-B-C or not.) Interestingly enough, with region orientations specified, I do not necessarily need line directions, although I certainly

need line directions to find the region orientation values to begin with.

An example of an edge where support must hold is any concave edge which has a horizontal surface on its left when one looks along the edge in the direction of its "arrow", as does L-C-D in figure 8.15E.

Some examples of edges where support cannot hold are concave edges which have vertical surfaces (FRV, FV, or FLV) or downward pointing surfaces (FRD, FD, or FLD) on the left of the edges when looking along the direction of the "arrow"; line L-A-B in figure 8.15D is an edge of this type.

While I do not show how to do so here, I believe that the best way to add the understanding of support to the framework of my program is to:

(1) add support labels to lines in junction labelings where support can hold, and add these labelings to the regular set of labels,

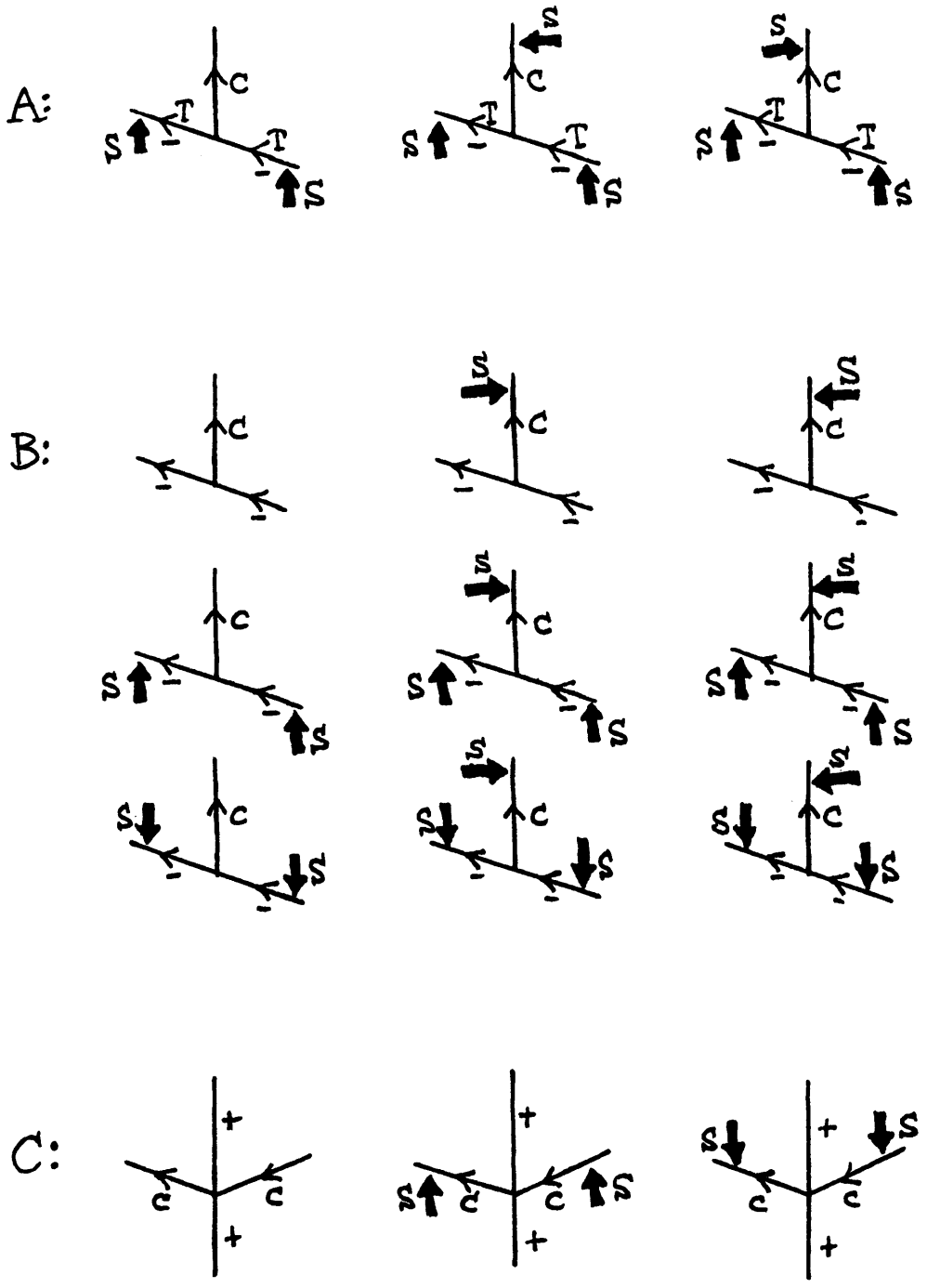
(2) as usual, do nothing when a line represents unambiguously a support edge or an edge without support, and

(3) when there is ambiguity, use the region orientation values to help decide the issue. To do this, note that since there is a connection between the edges which can have support, line directions, and region orientations, I can use the function ILLUMINE again to eliminate impossible combinations and hopefuly decide where support can and cannot hold.

I have no great confidence that such a system will show where support must hold for certain, but the knowledge about where support can hold combined with the knowledge that every object must be supported somehow, should allow the program to do quite well. I suspect that the program will be quite good at finding places where support cannot possibly hold. To solve these problems fully a program needs considerable knowledge about stability, gravity, and friction. These problems are outside the scope of this paper; for a discussion of some of the issues involved see Blum et al 1970.

To give a feeling for the number of new junctions which would be required in the data base, I have shown some junctions in figure 8.16 which can involve support. Figure 8.16A illustrates the fact that any concave edge which touches the TABLE must be a support edge. In figure 8.16B, if the crossbar of the T (the collinear lines) exhibits support on one of its halves, then it must exhibit support on the other half as well and the support direction must be the same for both of these edges. Similarly, in figure 8.16C both non-collinear edges must have the same support or lack of support values. If each of the branches which can potentially exhibit support relations were labeled independently, then the cases in figures 8.16A and 8.16B would each have 27 possible support assignments instead of three and nine respectively, and the case in figure 8.16C would have 9 assignments instead of the actual three. Thus the same kinds of techniques which I have shown earlier for other descriptions would almost certainly work well for support cases too. Finally, obscuring edges, which have up to now accounted for the biggest increases in the numbers of new labels when the old labels were split into subtypes do not even take part in this partitioning, so that the increase in the total number of labelings should be well within bounds.

FIGURE 8.16



A
B S ← MEANS THAT OBJECT OF WHICH B IS A PART SUPPORTS OBJECT OF WHICH A IS A PART.

NO MARK MEANS THAT NEITHER OBJECT SUPPORTS THE OTHER ALONG THIS EDGE.

Of course my present program can already list lines where support may hold (i.e. all crack and two-object concave edges), and as before, simple heuristics would allow the program to say with some confidence where support could or could not hold. Clearly, it would also be quite natural to call some of the support assignments in figure 8.16 "likely" and certain others "extremely unlikely".

9.0 HISTORICAL PERSPECTIVE

It is instructive to reexamine earlier vision work which dealt with similar problems in the light of the formalisms I have presented in this paper. In this chapter I review the work of Guzman (Guzman 1968), Rattner (Rattner 1970), Orban (Orban 1970), Freuder (Freuder 1971a, 1971b), Dowson (Dowson and Waltz 1971, Dowson 1971a, Dowson 1971b), Huffman (Huffman 1971), and Clowes (Clowes 1971, Clowes et al 1971).

In what follows I hope to give you some appreciation for the real advances in thinking about vision which were brought about by these authors. Ten years ago the whole area of computer vision was uncharted territory, and it was certainly far from obvious where one should begin. Today, while there are innumerable questions still unanswered, we have some definite ideas about how vision systems could be organized and about the reasons why many appealing systems such as perceptrons and template matching schemes are inadequate models for vision systems (Minsky & Papert 1970).

9.1 GUZMAN'S SEE PROGRAM

Guzman's work is probably the most famous of the earlier vision work, and indeed his approach was a dramatic departure from what had been done before him. His formalisms were designed to group regions together into bodies. Basically his program did this by identifying each line in a line drawing as linking or not linking, where linking means that the regions on both sides of the line belong to the same body and not linking means that there is no evidence about the line; it may be either linking or the regions on either side of the line may belong to different bodies. Guzman used a set of junction types exactly as my program does (L, ARROW, T, etc.) but he included only one labeling for each type of junction. Guzman's junction set is shown in figure 9.1.

There can be two conditions for any line in a line drawing after the labelings have been assigned to each junction:

(1) the labels at either end of a line agree, in which case the labels are assumed to be correct, or

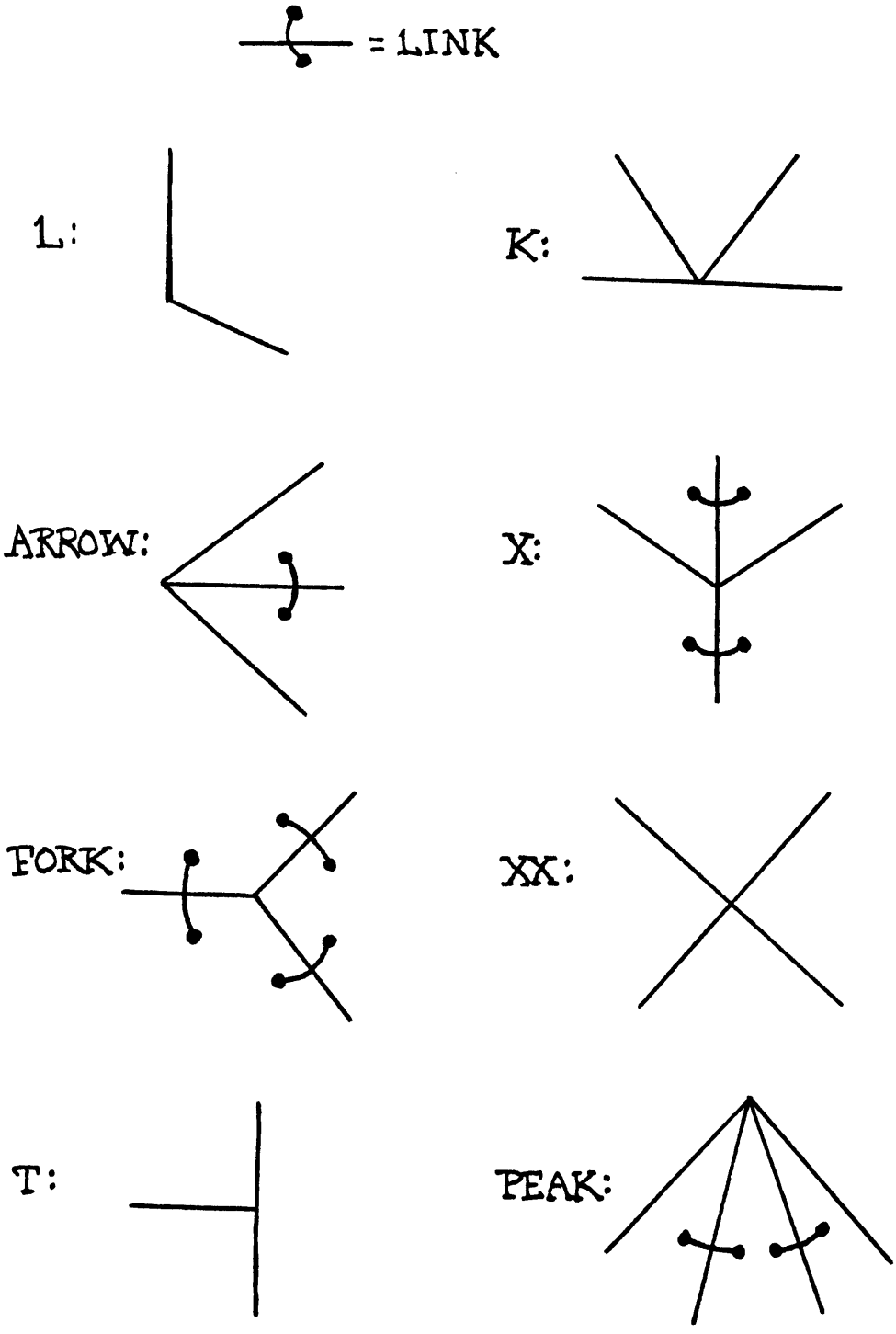


FIGURE 9.1

(2) the labels on a line do not agree; in this case heuristics are invoked to settle the issue in favor of one or the other of the labels.

As examples of these heuristics, Guzman originally linked regions if either end of a line were marked with a linking label. Later he added a system using "weak" and "strong" links to allow more subtle weighting of possibilities and, still later, he added a link inhibition feature which provided evidence against linking certain regions. Rattner (Rattner 1970) worked out various extensions to Guzman's work along these lines.

As it turned out, the link inhibition feature proved to be a much more powerful method than the previous methods he had tried. Basically this is because the link inhibition technique was less local than the previous links had been. The assignment of a link inhibition between two regions has consequences for every line which separates the two regions, unlike the linking mark which only serves as one piece of evidence in favor of linking two regions. In terms of my program, the program using links only is very much like what my program would be if I divided my labels up into those which had PLUS (convex) marks and all the rest (assume that

there are no shadows). The link inhibition labels would be those which have an arrow on the line segment, such as occluding edges, cracks, etc. The only strong evidence for linking regions comes from ARROW and FORK junctions, and of these the ARROW junctions are the more important, since (ignoring shadows and separable PLUS edges) every ARROW labeling links the two regions which bound the shaft of the ARROW. In contrast, there are a number of FORK junctions which have non-linking lines (see figure 9.2).

But if link inhibitions are used there is considerable evidence in ARROW, T, X, and K junctions; in fact Freuder has shown that if only link inhibitions are used, the program works just about as well as Guzman's full program.

There are numerous problems with Guzman's approach. First, his system simply does not work very well; for carefully chosen scenes it will find the correct results, but the program is very easy to fool. As Winston showed (Winston 1968) Guzman's program fails badly on scenes with holes, and obviously the program is worthless for scenes with shadows. If I map my labelings into Guzman's binary scheme there are examples of virtually every possible labeling for each junction type within my data base. Thus it becomes obvious

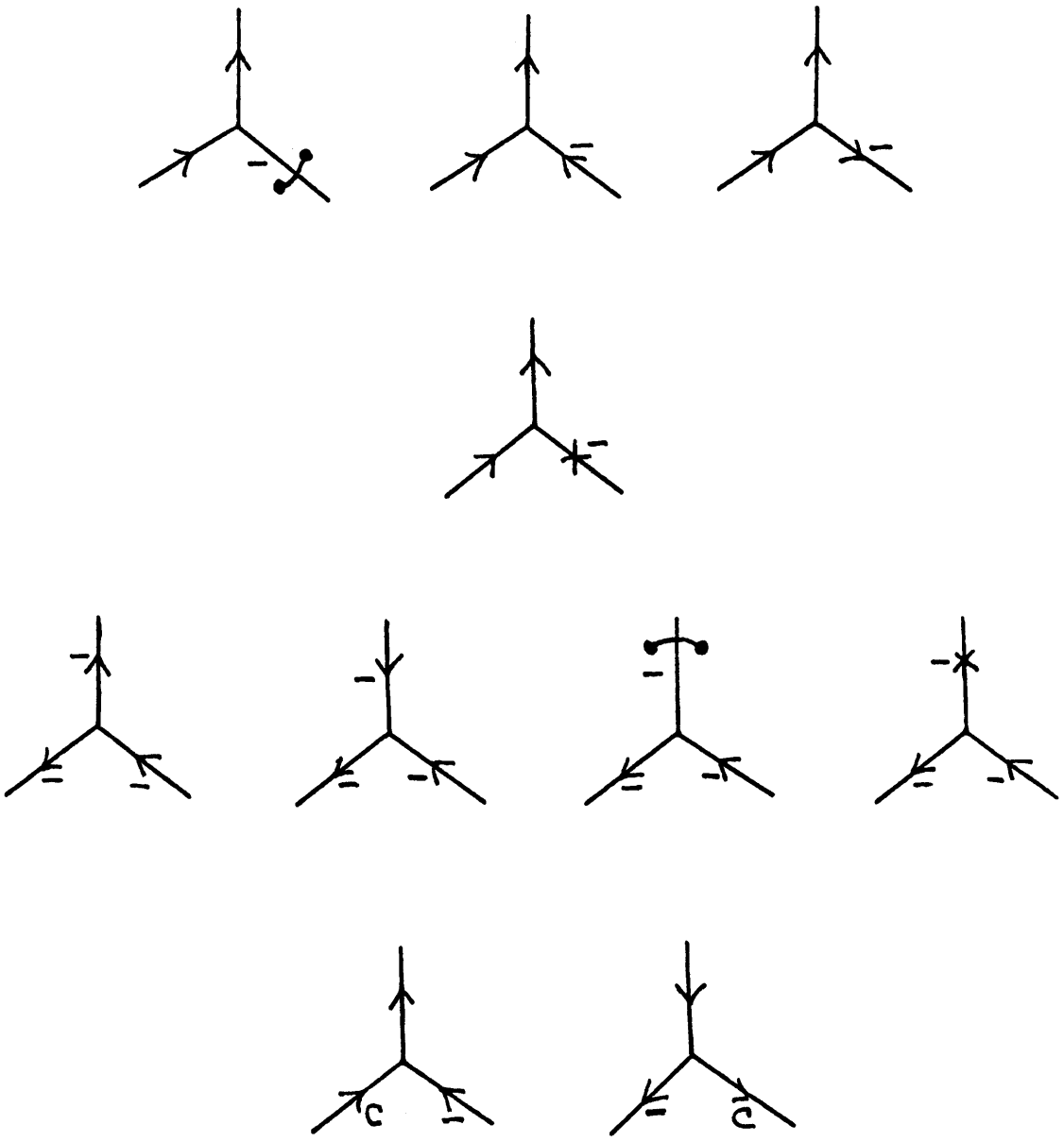


FIGURE 9.2

SOME COMMON FORKS WITH NON-LINKED EDGES (LINKS SUPERIMPOSED WHERE APPLICABLE)

that Guzman's labels are simply the most probable combinations of links for scenes without shadows. As such, his program really has very little understanding of the world (see Winston 1972a, 1972b).

Second, Guzman's approach is difficult to extend. This is due to the use of only one labeling for each junction and consequent heavy dependence on special purpose heuristics, and due also to the fact that virtually all the linking information for a line comes only from the two junctions at the ends of the line. There is no systematic way to use any information except locally. (The only exceptions are Guzman's use of matched Ts, the link inhibition information, and regions which meet along more than one edge.) As an example, Orban's extension of Guzman's program to include shadows (Orban 1970) depends exclusively on the observation that shadows frequently have chained L and X junctions. But despite the fact that Orban's program does have a slightly greater understanding of the meaning that scene features can have, it is not a systematic extension. Like almost all the extensions suggested for Guzman's work, it is a patchwork method: to handle a new distinction, pick a few common features that display the distinction and then adjust the rest of the program to avoid making disastrous errors.

Third, this approach leaves a great deal unexplained. Certainly there is a great deal more to understanding a scene than simply being able to connect the regions into bodies.

So far I have been dealing with the ways in which Guzman's approach was deficient, but it has strong features as well. Guzman was the first person, to my knowledge, to get away from the idea of storing descriptions of particular objects and trying to match these descriptions to a given scene. Roberts (Roberts 1963) had used this method and in fact others continued to do even less sophisticated template matching of sorts well after Guzman published his work. In contrast, Guzman's method works for arbitrary scenes containing trihedral vertices and gives some answer for any scene presented to it. Perhaps the most appealing feature of SEE was its simplicity and clarity; there are no tranformations, coordinates, or hidden lines, and in fact only topology is used. Guzman's great insight was that by describing the physical characteristics of a relatively small number of local features, one can use simple decision procedures to derive much less local facts about arbitrary, unfamiliar scenes.

Guzman's work was also instrumental in initiating two fruitful lines of research which are still active. This paper is along the line defined by Huffman and Clowes (Huffman 1971, Clowes 1971). The other line is the work on heterarchy (For excellent discussions of both Guzman and heterarchy see Winston 1972a or 1972b, and Minsky & Papert 1972).

9.2 WORK AFTER GUZMAN; HUFFMAN & CLOWES

Huffman was motivated partly by his observation of the lack of semantic content in Guzman's program to suggest a richer set of labels than link and do-not-link. (Whether Clowes came upon the same ideas independent of Guzman or not I do not know.) Clearly both were influenced by Guzman's "grammatical" approach to scene processing. Their great insight was that by describing edges more precisely one could use definite rules rather than probabilistically based heuristics to choose scene interpretations. Moreover they showed that one could even say with some assurance that certain line drawing could not even correspond to real physical scenes; compare this with the fact that Guzman's program rather blindly returns some decomposition into bodies for any line drawing, and you will get some idea of the

increase in understanding implicit in Huffman's and Clowes' work.

Both Huffman and Clowes also worked with a construction for representing region orientations called the dual graph which influenced my thinking on region orientations (Huffman 1971, Clowes et al 1971). Unfortunately, there is no neat way that I could see to integrate the dual graph into a labeling scheme. In any case, I owe Huffman and Clowes a considerable debt.

9.3 AN ACCOUNT OF MY EFFORTS

When Dowson and I began working in this area, we envisioned a tree searching program which would attempt to assign labelings from a reasonably small set (like those of Huffman and Clowes) to a line drawing. Dowson came up with a set of junctions involving cracks, and I generated a list of shadow junctions (Dowson & Waltz 1971). Dowson then developed VIRGIN, a tree search type labeling program (Dowson 1971b) to apply this knowledge to real scenes. He immediately ran into serious problems, since even the simplest scenes required huge amounts of computer space, and the program ended up with many possible labelings for each

scene. Most of these labelings only differed by one or two line labels, but each of which took a considerable amount of time to produce. It did not become obvious to me until somewhat later that tree search was the wrong model for this problem.

In my proposal for this work (Waltz 1971) I suggested a rather heterarchical model for labeling line drawings. At this time I had already noted that by beginning with the scene/background boundary I could cut down the search space considerably, and I listed a number of rules (related to the selection rules and region illumination types) which I thought could further speed up and increase the power of a program. I also showed that region orientations could be handled easily if I restricted the universe of objects to include only those with right-angle edges.

My major breakthrough came when I saw that the region orientations could be included as part of the edge labels, and then saw that I could also subdivide each edge type into several types according to the way that each edge could be decomposed. This idea was first suggested to me by Freuder (see Freuder 1971a) nearly a year before I used it.

The last pieces fell into place when I made the decision to try using a filtering program before doing a tree search, based on my observations of Dowson's difficulties. Since the set of labelings I now had was far larger than the set which had clogged his program, I felt that I needed such a program to clear away the clutter of unneeded labelings and make tree searching feasible. I was genuinely surprised when the filtering program returned unique labelings for most of the junctions in the first scenes I gave to it. From here on my work followed directly from the success of the combination of this filtering program and the much enlarged junction labeling sets. I think it is noteworthy that this workis the direct result of my interaction with the program, as opposed to being the result of a system I worked out first by hand and only then implemented in a program.

There is one lesson which I think is important, perhaps more important than any other in terms of the ways it might aid future research. For a long time after I had found the ways of describing region illuminations and edge decompositions, I tried to find a clever way to collapse the large set of line labels these distinctions implied into a smaller and more manageable set which would retain all the "essential" distinctions, whatever they were. Frustrated in

this attempt for quite a while, I finally decided to go ahead and include every possible labeling in the program, even though this promised to involve a good deal of typing. I hoped that when I ran the program certain regularities would appear, i.e. that when the program found a particular labeling for a junction it would always find another as well, so that the two labelings could be collapsed into one new one with no loss of information. Of course, as it turned out, it was the fact that I had made such precise distinctions that allowed the program to find unique labelings. The moral of this is that one should not be afraid of semi-infinities; a large number of simple facts may be needed to represent what can be deduced by computation using a few general ideas.

It also seems logical that, if anything, people are able to make much finer distinctions than I was considering, and that these distinctions had value for perception. For example, people can distinguish between obtuse or "blunt" edges (such as those of a regular dodecahedron), right angle edges (such as those of a cube), and acute or "sharp" edges (such as those of a regular tetrahedron).

Finally, I do not see any reason to suppose that we should be able to get along with distinctions on the order of one or two hundred, any more than a language program with a vocabulary of this size could comprehend or express anything very interesting. But by the same token, it may be that a vision system does not have to be too large for available computers in order to reach a point of diminishing returns, just as an increase in vocabulary beyond 10,000 words would probably not add much to a language program's (or a person's) ability.

BIBLIOGRAPHY

Blum M, Griffith A & Neuman B
A Stability Test for Configurations of Blocks
AI Memo 188
MIT Artificial Intelligence Laboratory
February 1970

Clowes M B
On Seeing Things
AI Journal
Spring 1971

Clowes M B, Mackworth A K & Stanton R B
Picture Interpretation as a
Problem Solving Process
Laboratory of Experimental Psychology
University of Sussex (England)
June 1971

Dowson M
What Corners Look Like
Vision Flash 13
Vision Group
MIT Artificial Intelligence Laboratory
June 1971

Dowson M & Waltz D L
Shadows and Cracks
Vision Flash 14
Vision Group
MIT Artificial Intelligence Laboratory
June 1971

Dowson M
Progress in Extending the VIRGIN Program
Vision Flash 20
Vision Group
MIT Artificial Intelligence Laboratory
October 1971

Finin T
Two Problems in Analyzing Scenes
Vision Flash 12
Vision Group
MIT Artificial Intelligence Laboratory
June 1971

Finin T
Finding the Skeleton of a Brick
Vision Flash 19
Vision Group
MIT Artificial Intelligence Laboratory
August 1971

Finin T
A Vision Potpourri
Vision Flash 26
Vision Group
MIT Artificial Intelligence Laboratory
June 1972

Freuder E
The Object Partition Problem
Vision Flash 4
Vision Group
MIT Artificial Intelligence Laboratory
February 1971

Freuder E
Views on Vision
Vision Flash 5
Vision Group
MIT Artificial Intelligence Laboratory
February 1971

Gaschnig J
Resolving Visual Ambiguity With a Probe
Vision Flash 17
Vision Group
MIT Artificial Intelligence Laboratory
July 1971

Gibson J J
The Perception of the Visual World
Houghton Mifflin
Boston
1950

Guzman A
Computer recognition of Three-dimensional
Objects in a Visual Scene
Technical Report AI-TR-228
MIT Artificial Intelligence Laboratory
December 1968

Horn B K P
Shape from Shading: A Method for Obtaining
the Shape of a Smooth Opaque Object from One View
Technical Report AI-TR-232
MIT Artificial Intelligence Laboratory
November 1970

Horn B K P
The Binford-Horn Line Finder
Vision Flash 16
Vision Group
MIT Artificial Intelligence Laboratory
June 1971

Huffman D A
Impossible Objects as Nonsense Sentences
Machine Intelligence 6
Edinburgh University Press
1970

Koffka K
Principles of Gestalt Psychology
Harcourt-Brace
New York
1935

Lerman J B
Computer Processing of Stereo Images
for the Automatic Extraction of Range
MIT Department of Electrical Engineering Thesis
June 1970

Mahabala H N V
Preprocessor for Programs Which Recognize Scenes
AI Memo 177
MIT Artificial Intelligence Laboratory
August 1969

McDermott D L & Sussman G J
The CONNIVER Reference Manual
AI Memo 259
MIT Artificial Intelligence Laboratory
May 1972

Minsky M L & Papert S
Progress Report
AI Memo 252
MIT Artificial Intelligence Laboratory
January 1972

Orban R
Removing Shadows in a Scene
AI Memo 192
MIT Artificial Intelligence Laboratory
August 1970

Rattner M H
Extending Guzman's SEE Program
AI Memo 204
MIT Artificial Intelligence Laboratory
July 1970

Roberts L
Machine Perception of Three-Dimensional Solids
Technical Report 315
MIT Lincoln Laboratory
May 1963

Shirai Y
Extraction of the Line Drawing of
3-Dimensional Objects by Sequential Lighting
from Multidirections
Electrotechnical Laboratory Technical Report
Tokyo
1971

Shirai Y
A Heterarchical Program for Recognition
of Polyhedra
AI Memo 263
MIT Artificial Intelligence Laboratory
June 1972

Sussman G J, Winograd T & Charniak E
MICRO-PLANNER Reference Manual
AI Memo 203A
MIT Artificial Intelligence Laboratory
December 1971

Waltz D L
Understanding Scenes with Shadows
Vision Flash 21
Vision Group
MIT Artificial Intelligence Laboratory
November 1971

Waltz D L
Shedding Light on Shadows
Vision Flash 29
Vision Group
MIT Artificial Intelligence Laboratory
July 1972

Winston P H
Holes
AI Memo 163
MIT Artificial Intelligence Laboratory
August 1968

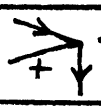
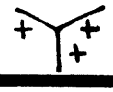
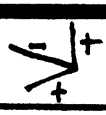
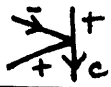
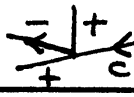
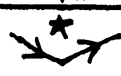
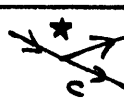
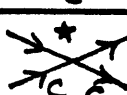
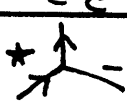
Winston P H
Learning Structural Descriptions from Examples
Technical Report MAC-TR-76
MIT Artificial Intelligence Laboratory
September 1970

Winston P H
The MIT Robot
Machine Intelligence 7
Edinburgh University Press
1972

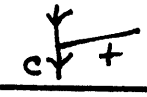
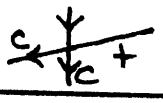
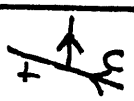
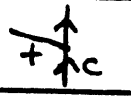
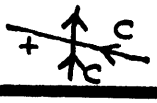
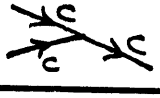
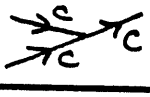
Winston P H
Summary of Selected Vision Topics
Vision Flash 30
Vision Group
MIT Artificial Intelligence Laboratory
July 1972

APPENDIX 1

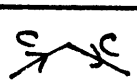
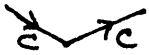
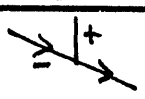
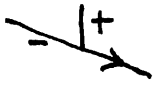
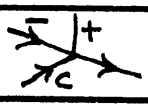
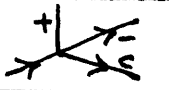
(TO FIND MEANING OF OCTANT NUMBERS, SEE FIG 3.1)
(OCTANTS WHICH HAVE THE SAME LETTER = OBJECT)

111	110	101	100	011	010	001	000	LABELING	NAME
							A		ARROW-1
						A			L-1
					A				FORK-1
						A	B		T-2A
A	B								T-2B
				A	B				X-2
				A	A		A		ARROW-3A
				A	A		B		K-3A
				A	B		B		K-3B
				A	B		C		KXX-3A
	A		A				A		L-3A
	A		B				B		T-3A
	A		A				B		T-3B
	A		B				C		XX-3A
A	A		A						FORK-3A

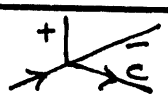
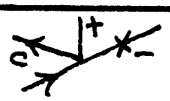
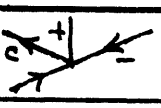
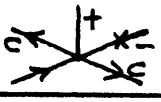
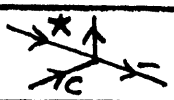
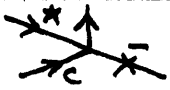
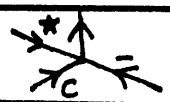
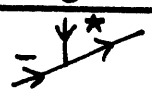
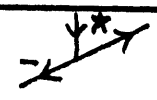
APPENDIX 1

111	110	101	100	011	010	001	000	LABELING	NAME
A	B		B						FORK-3B
A	A		B						FORK-3C
A	B		C						FORK-3D
					A	A	A		L-3B
					A	B	A		T-3C
					A	B	B		T-3D
					A	B	C		XX-3B
A	A				A				L-3C
A	B				B				T-3E
A	A				B				T-3F
A	B				C				XX-3C
	A		B		C		D		XX-4A
	A		B		C		C		T-4A
	A		B		A		C		T-4B
	A		A		B		C		T-4C

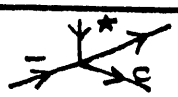
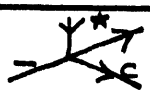
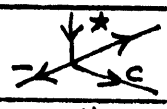
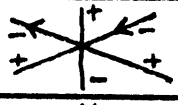
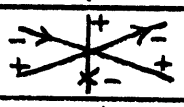
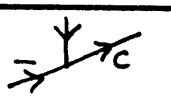
APPENDIX 1

111	110	101	100	011	010	001	000	LABELING	NAME
	A		B		C		B		T-4D
	A		B		A		A		L-4A
	A		A		A		B		L-4B
	A		A		B		A		L-4C
	A		B		B		B		L-4D
			A	B	B		A		T-4E
			A	A	A		A		T-4F
			A	B	B		C		X-4A
			A	B	B		B		X-4B
			A	B	C		A		K-4A
			A	B	A		A		K-4B
			A	B	C		D		KXX-4A
	A		A	B	B				T-4G
	A		A	A	A				T-4H
	A		B	C	C				X-4C

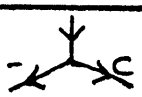
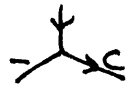
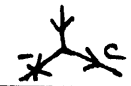
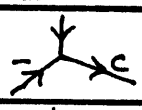
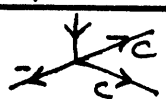
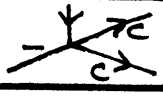
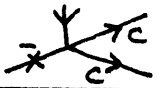
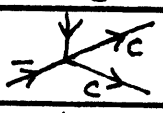
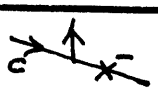
APPENDIX 1

111	110	101	100	011	010	001	000	LABELING	NAME
	A		B	A	A				X-4D
	A		A	B	C				K-4C
	A		A	B	A				K-4D
	A		B	C	D				KXX-4B
A	A		B				B		T-4I
A	A		A				A		T-4J
A	B		B				B		T-4K
A	B		C				C		T-4L
A	A		B				C		X-4E
A	A		A				B		X-4F
A	B		C				D		X-4G
A	B		B				C		X-4H
	A		A			B	B		T-4M
	A		A			A	A		T-4N
	A		A			B	A		T-4O

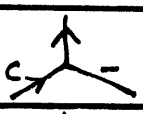
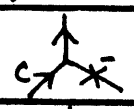
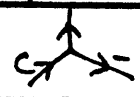
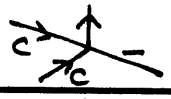
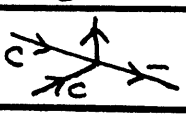
APPENDIX 1

111	110	101	100	011	010	001	000	LABELING	NAME
	A		A			B	C		T-4P
	A		B			C	B		X-4I
	A		B			B	B		X-4J
	A		B			C	D		X-4K
	A		B			C	B		X-4L
	A			A	A		A		SPECIAL-4A
	A			B	A		A		SPECIAL-4B
	A			B	B		C		SPECIAL-4C
	A			B	C		D		SPECIAL-4D
	A		A		A	B	A		1-5A
	A		A		A	A	A		1-5B
	A		A		A	B	B		1-5C
	A		A		A	B	C		1-5D
	A		A		C or D	B	C		T-5A
	A		A		B or C	B	B		T-5B

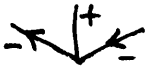
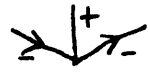
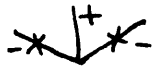
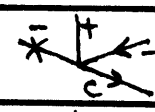
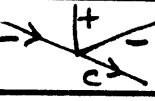
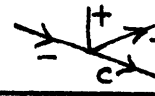
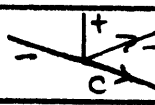
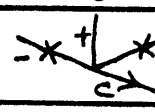
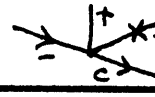
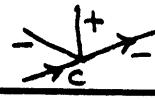
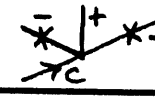
APPENDIX 1

111	110	101	100	011	010	001	000	LABELING	NAME
	A		A		B	C	A		T-5C
	A		A		B	A	A		T-5D
	A		B		A	C	B		FORK-5A
	A		B		A	B	B		FORK-5B
	A		B		A	C	D		FORK-5C
	A		B		A	C	C		FORK-5D
	A		B		B or D	C	B		X-5A
	A		B		B or C	B	B		X-5B
	A		B		D or E	C	D		X-5C
	A		B		C or D	C	C		X-5D
A	B		B		B		B		L-5E
A	A		A		A		A		L-5F
A	A		B		B		B		L-5G
A	B		C		C		C		L-5H
A	B		C		B or D		C		T-5E

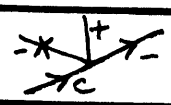
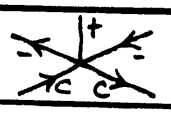
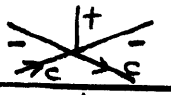
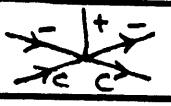
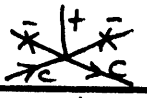
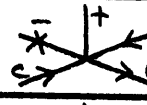
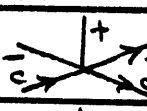
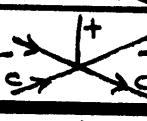
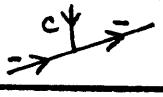
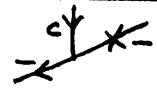
APPENDIX 1

111	110	101	100	011	010	001	000	LABELING	NAME
A	A		A		A		A		L-5F
A	A		B		B		B		L-5G
A	B		C		C		C		L-5H
A	B		C		B or D		C		T-5E
A	A		B		A or C		B		T-5F
A	B		C		B or D		C		T-5G
A	A		A		B		A		T-5H
A	B		B		C		C		FORK-5E
A	A		A		B		B		FORK-5F
A	B		C		D		D		FORK-5G
A	A		B		C		C		FORK-5H
A	B		B		B or D		C		X-5E
A	A		A		A or C		B		X-5F
A	B		C		B or E		D		X-5G
A	A		B		A or D		C		X-5H

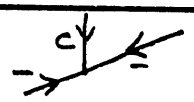
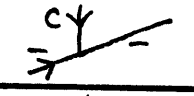
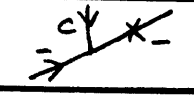
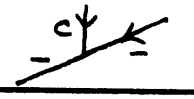
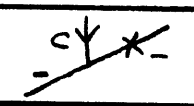
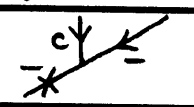
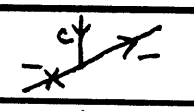
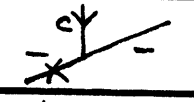
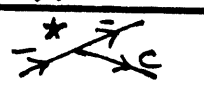
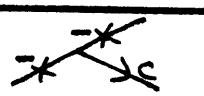
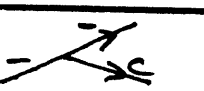
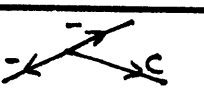
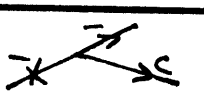
APPENDIX 1

111	110	101	100	011	010	001	000	LABELING	NAME
	A		A	B	A		A		ARROW-5A
	A		A	A	A		A		ARROW-5B
	A		A	B	B		A		ARROW-5C
	A		A	B	C		A		ARROW-5D
	A		B	C	A		B		K-5A
	A		B	A	A		B		K-5B
	A		B	C	C		B		K-5C
	A		B	B	B		B		K-5D
	A		B	C	D		B		K-5E
	A		B	C	B		B		K-5F
	A		A	B	C		C		K-5G
	A		A	B	B		B		K-5H
	A		A	B	B		C		K-5I
	A		A	A	A		B		K-5J
	A		A	B	C		D		K-5K

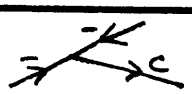
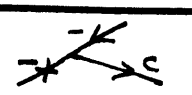
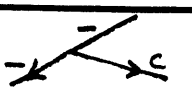
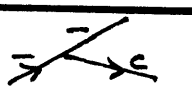
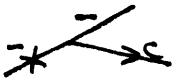
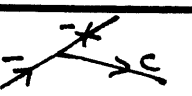
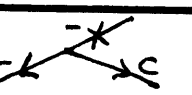
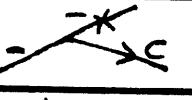
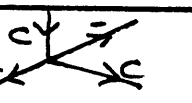
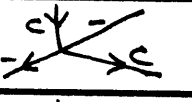
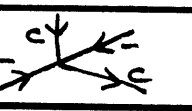
APPENDIX 1

111	110	101	100	011	010	001	000	LABELING	NAME
	A		A	B	A		C		K-5L
	A		B	C	A		A		KXX-5A
	A		B	A	A		A		KXX-5B
	A		B	C	C		D		KXX-5C
	A		B	C	D		E		KXX-5D
	A		B	C	D		D		KXX-5E
	A		B	C	A		D		KXX-5F
	A		B	C	C		C		KXX-5G
	A		B	A	A		C		KXX-5H
	A		A	B	A	C	A		T-6A
	A		A	B	B	C	C		T-6B
	A		A	B	C	D	C or E		T-6C
	A		A	A	A	B	A		T-6D
	A		A	B	B	C	A		T-6E
	A		A	B	C	D	A		T-6F

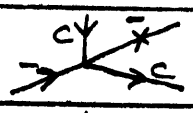
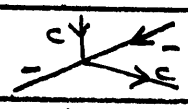
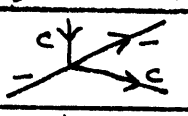
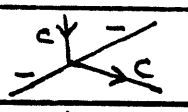
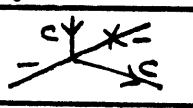
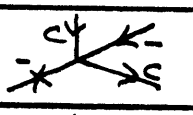
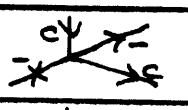
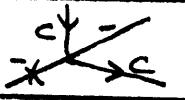
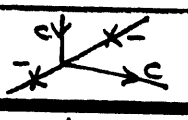
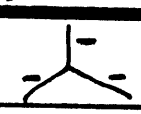
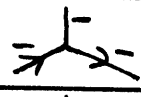
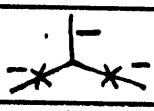
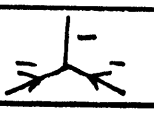
APPENDIX 1

111	110	101	100	011	010	001	000	LABELING	NAME
	A		A	B	A	C	C		T-6G
	A		A	A	A	B	B		T-6H
	A		A	B	C or D	C	C		T-6I
	A		A	B	A	A	A		T-6J
	A		A	B	B	A	A		T-6K
	A		A	B	C	A	A		T-6L
	A		A	B	A	C	D		T-6M
	A		A	B	B	C	D		T-6N
	A		A	A	A	B	C		T-6O
	A		B	C	C	C	C		T-6P
	A		B	C	A	C	B		T-6Q
	A		B	C	D	C	D or E		T-6R
	A		B	B	A	B	B		T-6S
	A		B	C	C	C	B		T-6T
	A		B	C	C	C	D		T-6U

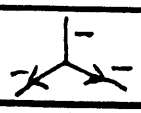
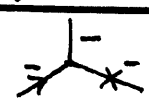
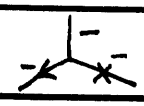
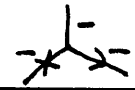
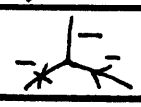
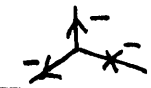
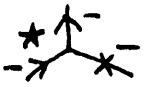
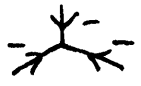
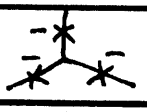
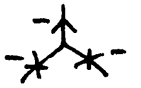
APPENDIX 1

111	110	101	100	011	010	001	000	LABELING	NAME
	A		B	C	A	C	C		T-6V
	A		B	B	A	B	B		T-6W
	A		B	C	A	C	A or D		T-6X
	A		B	A	A	A	B		T-6Y
	A		B	A	A	A	A		T-6Z
	A		B	A	A	A	C		T-6AA
	A		B	C	D	C	C		T-6AB
	A		B	C	B or D	C	B		T-6AC
	A		B	B	C	B	B		T-6AD
	A		B	C	A	D	B		X-6A
	A		B	C	C	D	B		X-6B
	A		B	A	A	C	B		X-6C
	A		B	C	B or E	D	B		X-6D
	A		B	C	A	D	D		X-6E
	A		B	C	C	D	D		X-6F

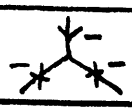
APPENDIX 1

111	110	101	100	011	010	001	000	LABELING	NAME
	A		B	A	A	C	C		X-6G
	A		B	C	D or E	D	D		X-6H
	A		B	C	A	B	B		X-6I
	A		B	C	C	B	B		X-6J
	A		B	A	A	B	B		X-6K
	A		B	C	B or D	B	B		X-6L
	A		B	C	A	D	A or E		X-6M
	A		B	C	C	D	C or E		X-6N
	A		B	A	A	C	A or D		X-6O
	A		B	C	D	E	F or D		X-6P
A	A		A	A	?	A	A		FORK-7A
A	B		B	A	?	A	B		FORK-7B
A	A		B	A	?	A	A		FORK-7C
A	B		C	A	?	A	B or D		FORK-7D
A	B		B	A	?	A	A		FORK-7E

APPENDIX 1

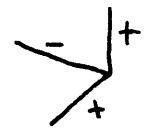
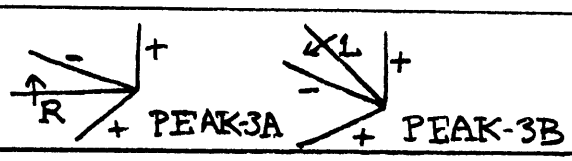
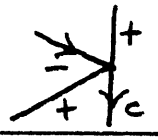
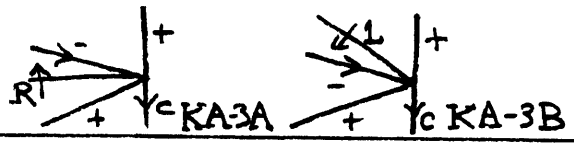
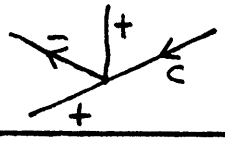
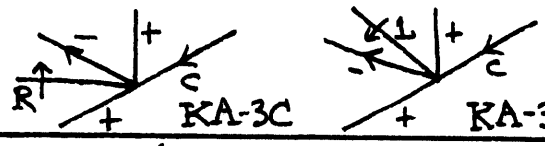
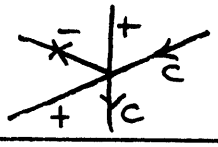
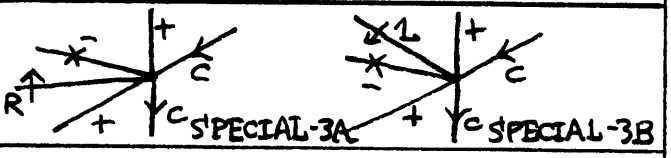
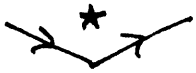
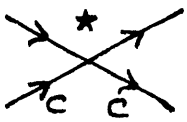
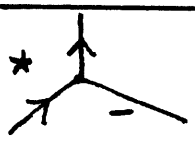
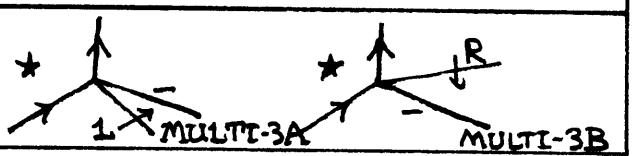
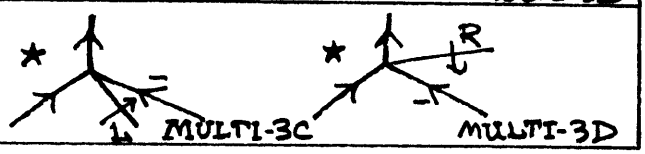
111	110	101	100	011	010	001	000	LABELING	NAME
A	A		B	A	?	A	B		FORK-7F
A	B		C	A	?	A	A		FORK-7G
A	B		C	A	?	A	C		FORK-7H
A	A		B	A	?	A	C		FORK-7I
A	B		B	A	?	A	C		FORK-7J
A	B		B	C	?	C	B		FORK-7K
A	B		C	D	?	D	C		FORK-7L
A	B		B	C	?	C	D		FORK-7M
A	B		B	A	?	C	B		FORK-7N
A	B		C	A	?	D	D		FORK-7O
A	B		C	D	?	D	D		FORK-7P
A	A		B	C	?	C	B		FORK-7Q
A	B		B	A	?	C	C		FORK-7R
A	B		C	D	?	E	D or B or F		FORK-7S
A	B		C	D	?	D	B or E		FORK-7T

APPENDIX 1

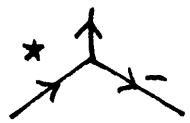
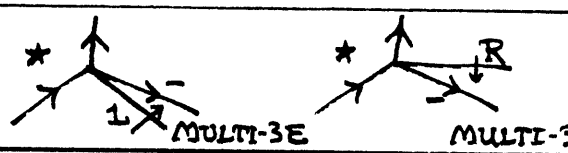
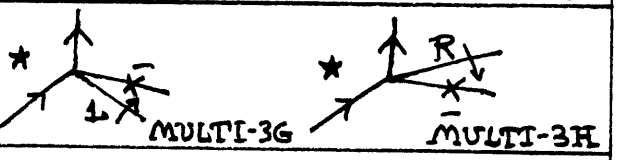
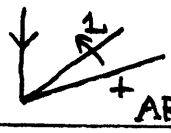
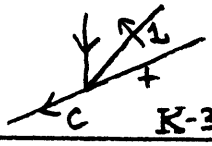
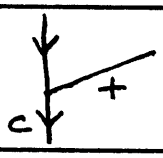
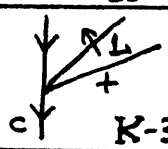
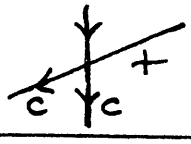
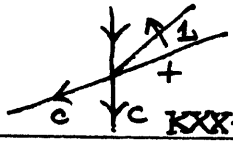
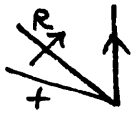
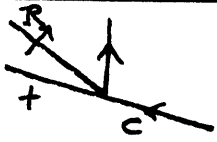
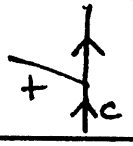
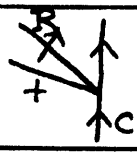
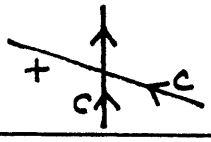
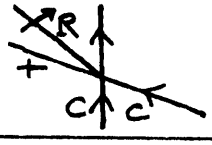
111	110	101	100	011	010	001	000	LABELING	NAME
A	B		C	A	?	D	B or E		FORK-7U

APPENDIX 2

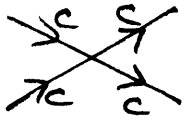
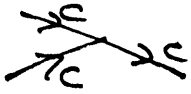
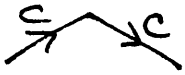
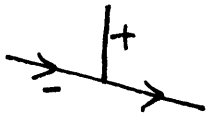
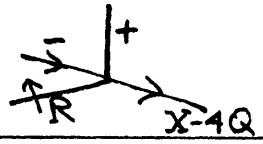
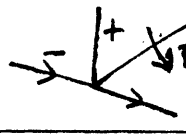
SHADOW VARIATIONS OF JUNCTIONS FROM APPENDIX 1.

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES
ARROW-3A		
K-3A		
K-3B		
KX-3A		
1-3A		NONE
T-3A		NONE
T-3B		NONE
XX-3A		NONE
FORK-3A		
FORK-3B		

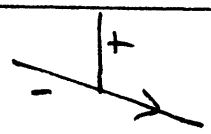
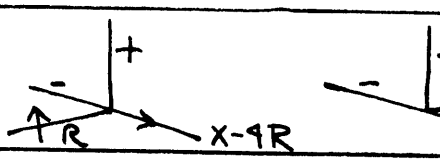
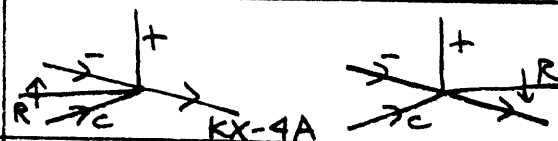
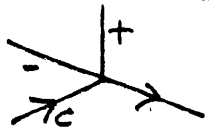
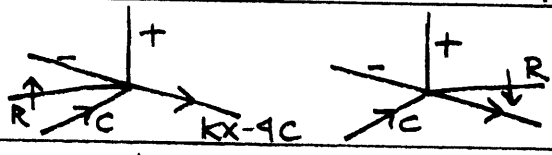
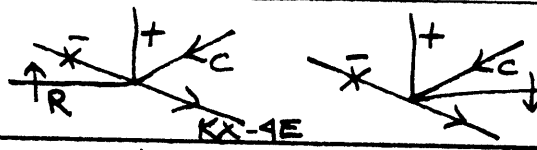
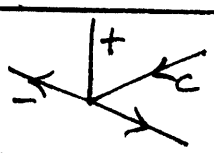
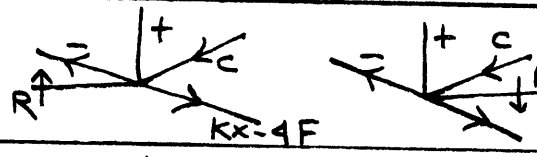
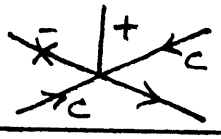
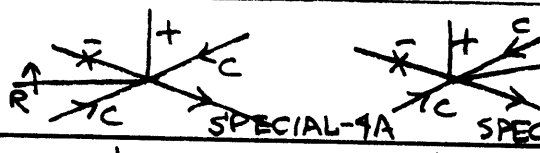
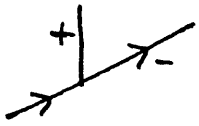
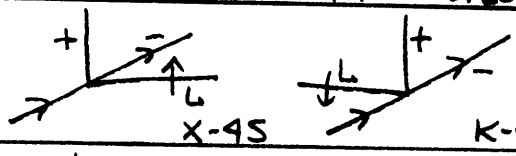
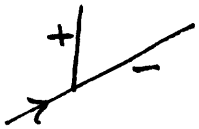
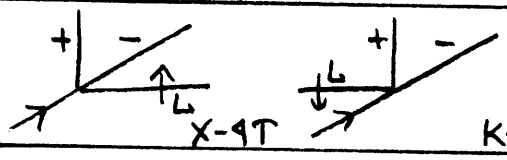
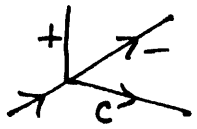
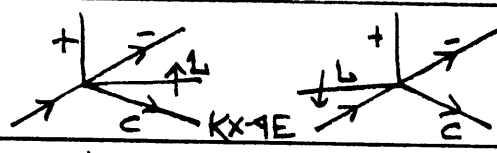
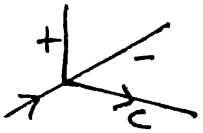
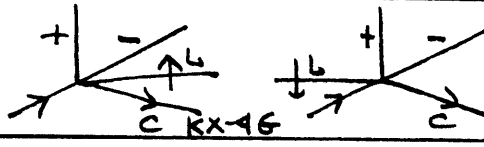
APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES
FORK-3C		
FORK-3D		
L-3B		
T-3C		
T-3D		
XX-3B		
L-3C		
T-3E		
T-3F		
XX-3C		

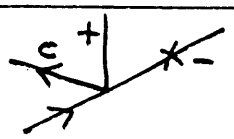
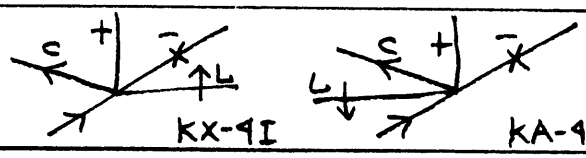
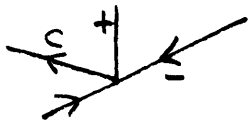
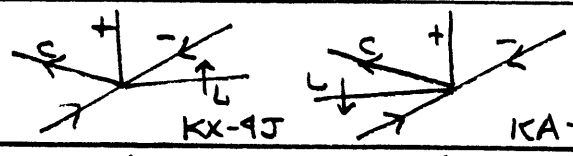
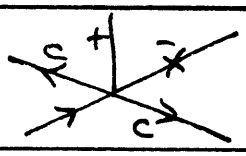
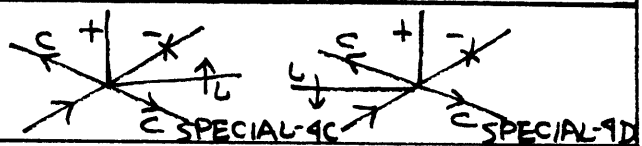
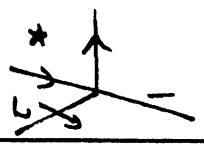
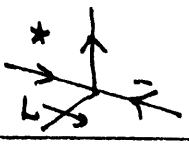
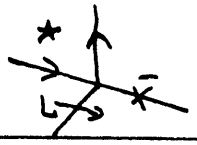
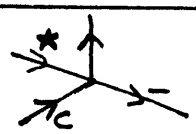
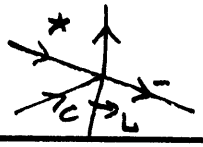
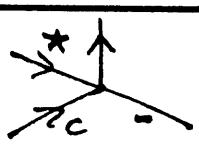
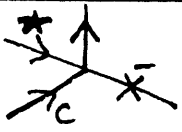
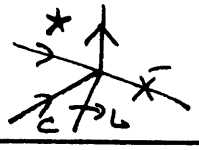
APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES
XX-4A		NONE
T-4A		NONE
T-4B		NONE
T-4C		NONE
T-4D		NONE
L-4A		NONE
L-4B		NONE
L-4C		NONE
L-4D		NONE
T-4E		 

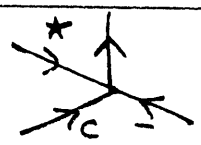
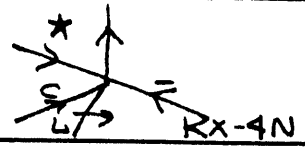
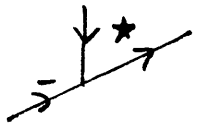
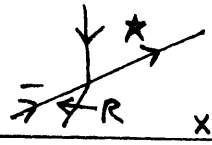
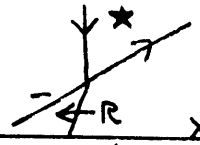
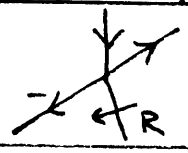
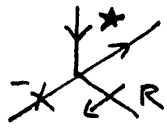
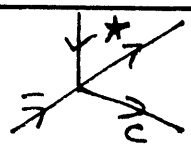
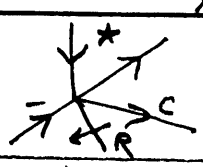
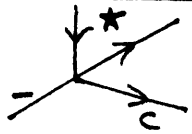
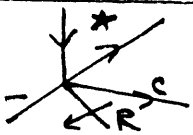
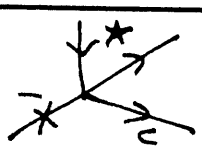
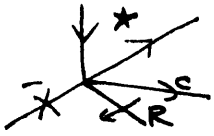
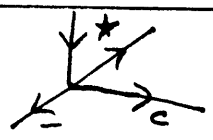
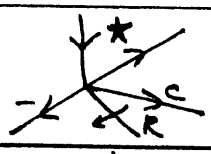
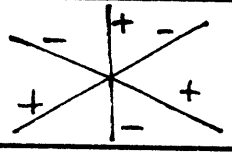
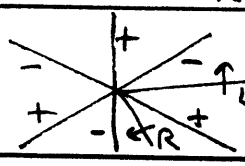
APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES
T-4F		
X-4A		
X-4B		
K-4A		
K-4B		
KXX-4A		
T-4G		
T-4H		
X-4C		
X-4D		

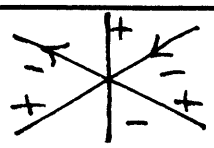
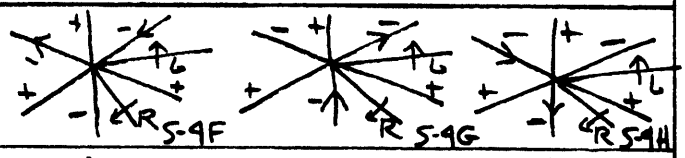
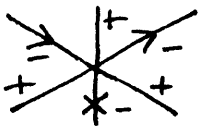
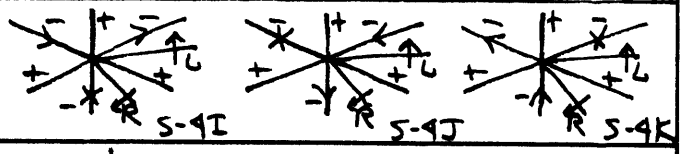
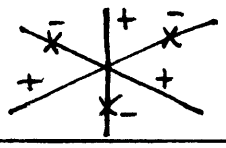
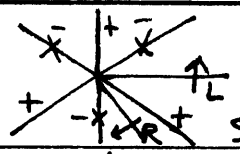
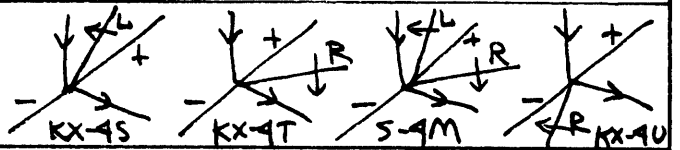
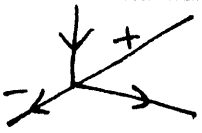
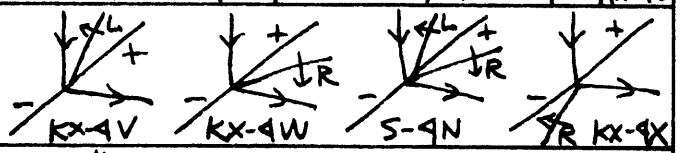
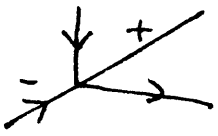
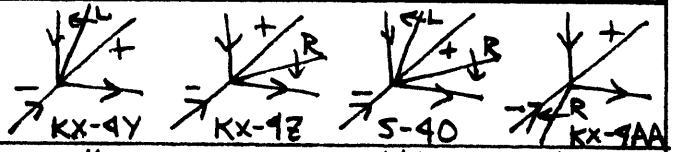
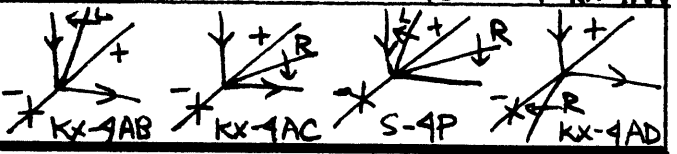
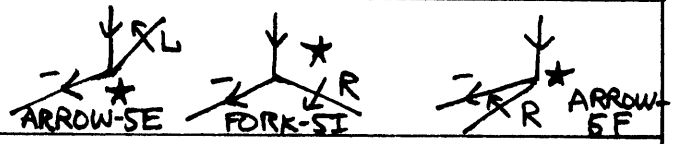
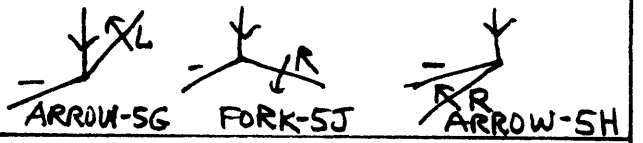
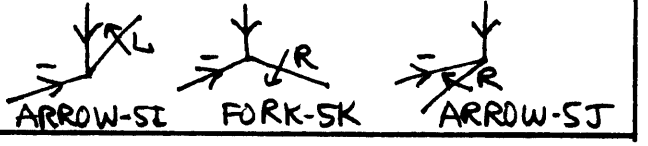
APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES
K-4C		 KX-4I KA-4C
K-4D		 KX-4J KA-4D
KXX-4B		 SPECIAL-4C SPECIAL-4D
T-4I		X-4U
T-4J		 X-4V
T-4K		 X-4W
T-4L		 X-4X
X-4E		 KX-4K
X-4F		KX-4L
X-4G		 KX-4M

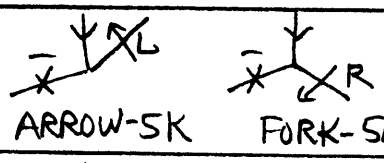
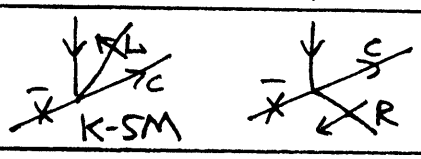
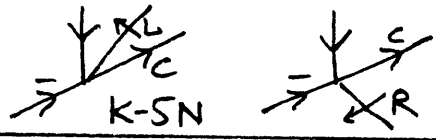
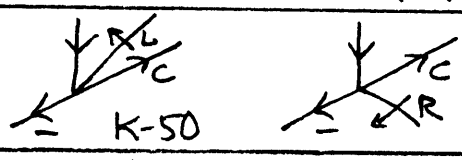
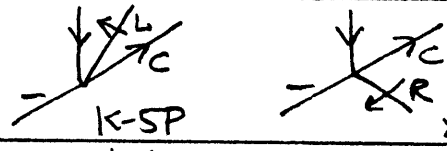
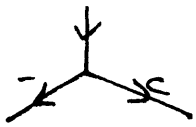
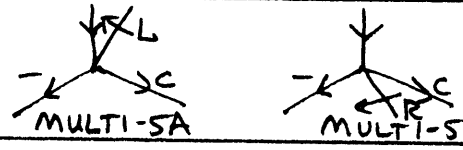
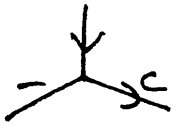
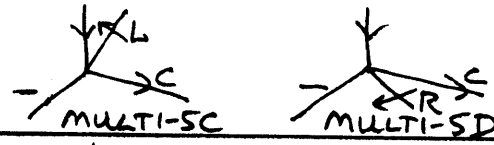
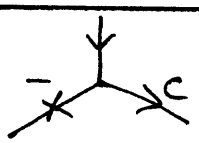
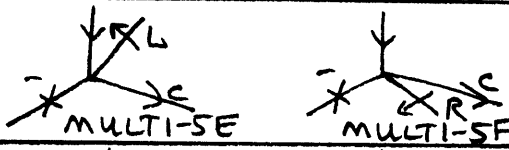
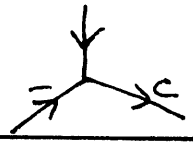
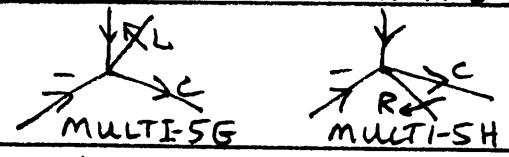
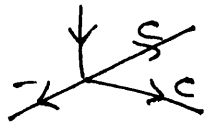
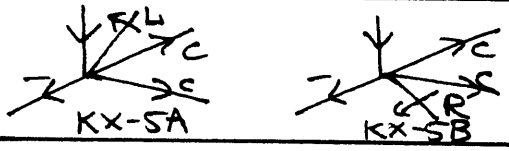
APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES
X-4H		 RX-4N
T-4M		 X-4Y
T-4N		 X-4Z
T-4O		 X-4AA
T-4P		 X-4AB
X-4I		 KX-4O
X-4J		 KX-4P
X-4K		 KX-4Q
X-4L		 KX-4R
SPECIAL-4A		 SPECIAL-4E

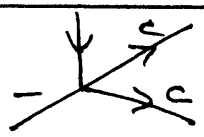
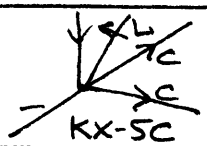
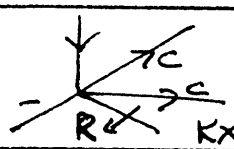
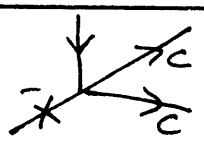
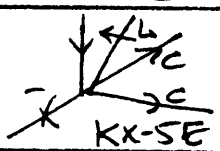
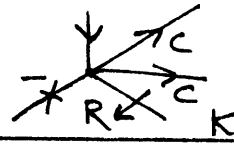
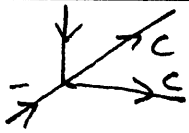
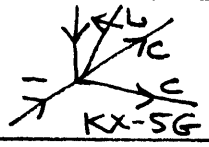
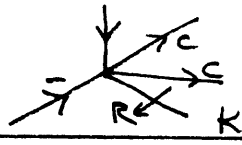
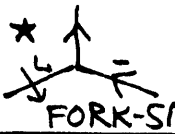
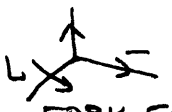
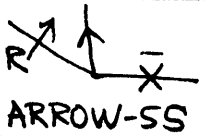
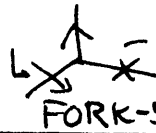
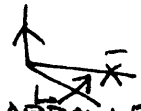
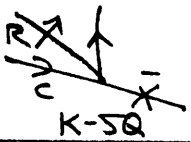
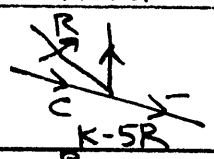
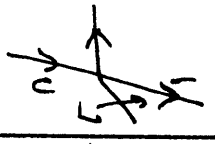
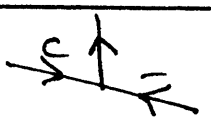
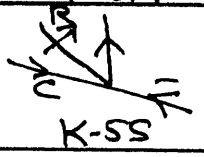
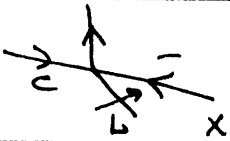
(S=SPECIAL) APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES
S-4B		
S-4C		
S-4D		
X-4M		
X-4N		
X-4O		
X-4P		
L-5A		
L-5B		
L-5C		

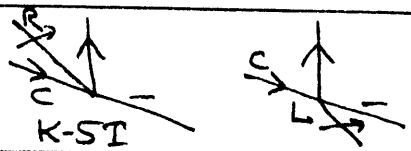
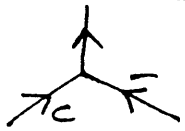
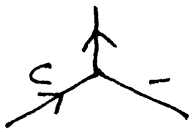
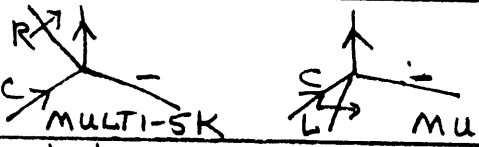
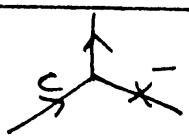
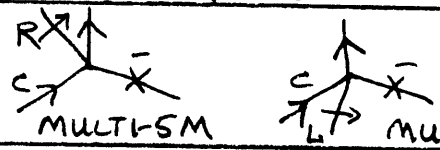
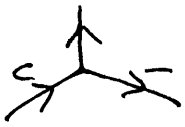
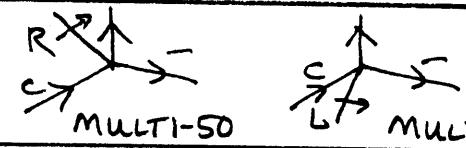
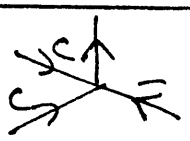
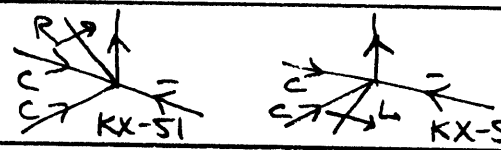
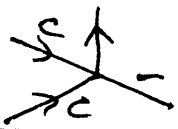
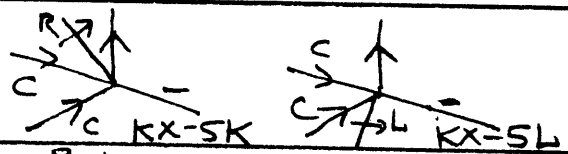
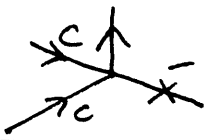
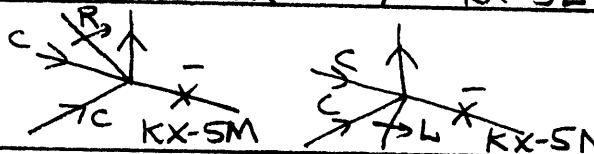
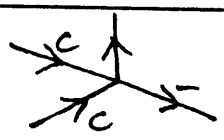
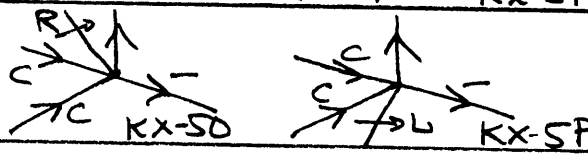
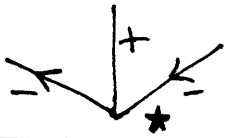
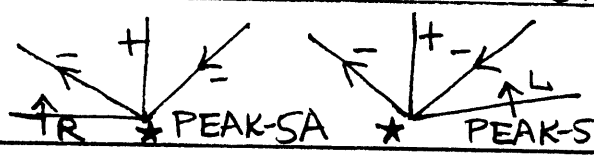
APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES
L-5D		
T-5A		
T-5B		
T-5C		
T-5D		
FORK-5A		
FORK-5B		
FORK-5C		
FORK-5D		
X-5A		

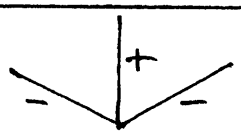
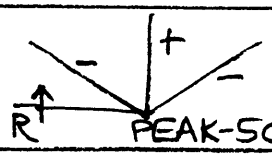
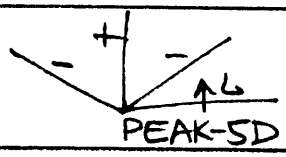
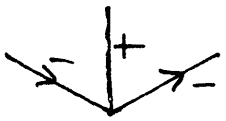
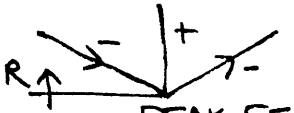
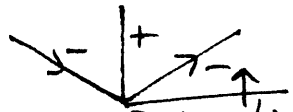
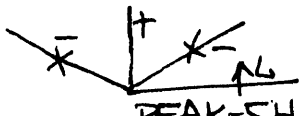
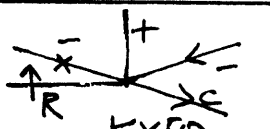
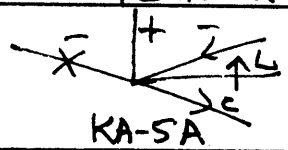
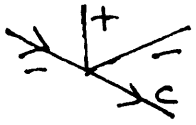
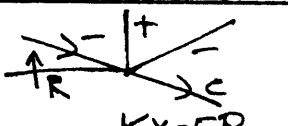
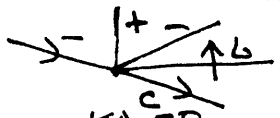
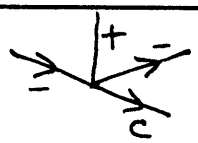
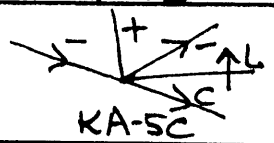
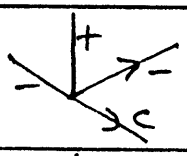
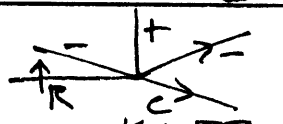
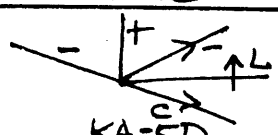
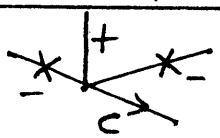
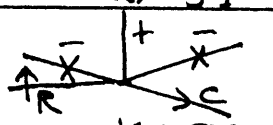
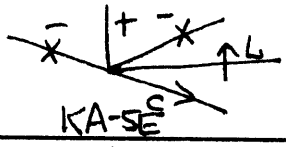
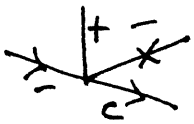
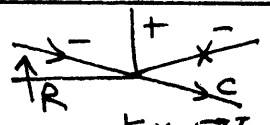
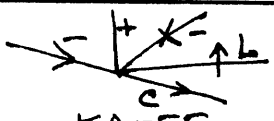
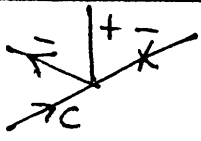
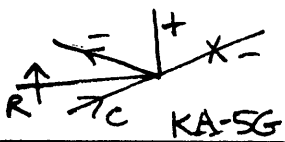
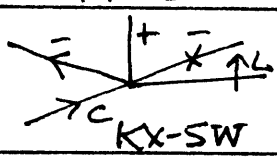
APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES
X-5B		 KX-5C  KX-5D
X-5C		 KX-5E  KX-5F
X-5D		 KX-5G  KX-5H
L-5E		ARROW-5M  FORK-5M ARROW-5N
L-5F		ARROW-5O FORK-5N ARROW-5P
L-5G		ARROW-5Q  FORK-5O ARROW-5R
L-5H		 ARROW-5S  FORK-5P  ARROW-5T
T-5E		 K-5Q X-5M
T-5F		 K-5R  X-5N
T-5G		 K-5S  X-5O

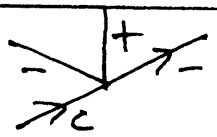
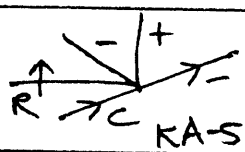
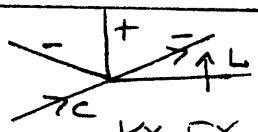
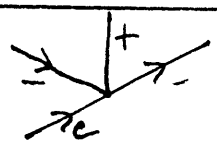
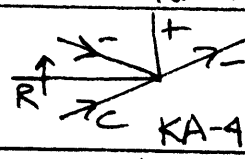
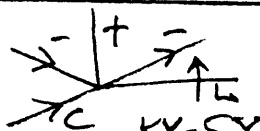
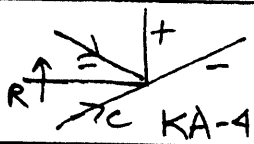
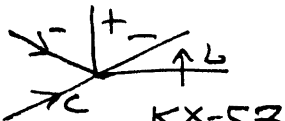
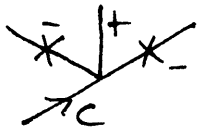
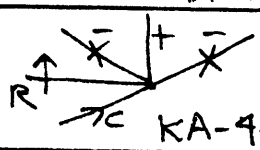
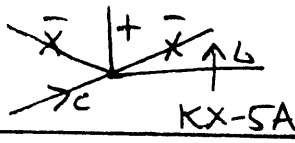
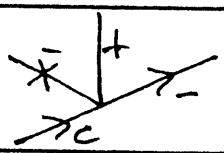
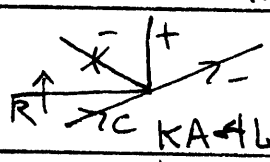
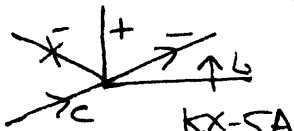
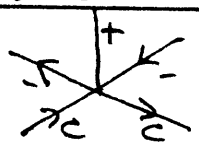
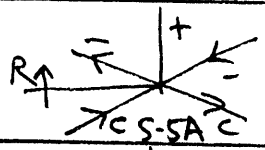
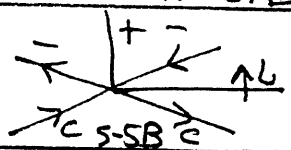
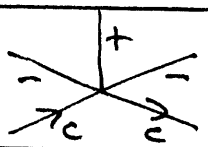
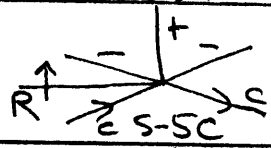
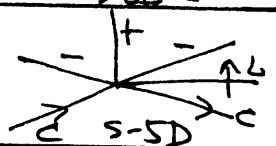
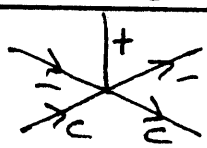
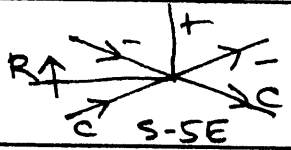
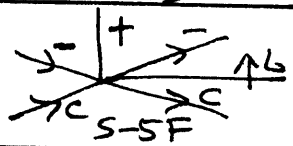
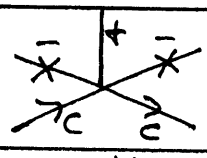
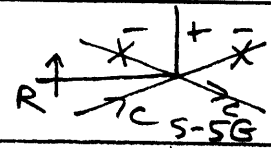
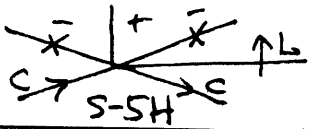
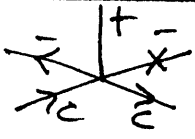
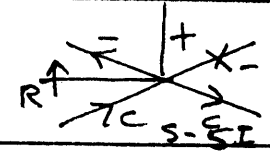
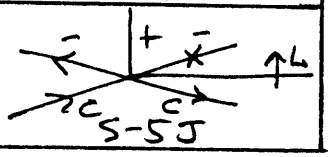
APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES
T-5H		
FORK-5E		
FORK-5F		
FORK-5G		
FORK-5H		
X-5E		
X-5F		
X-5G		
X-5H		
ARROW-5A		

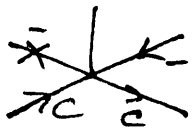
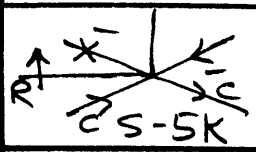
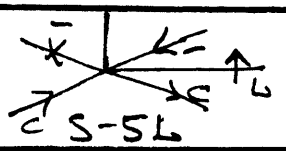
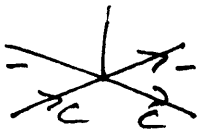
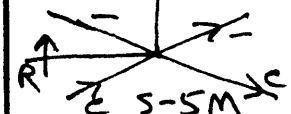
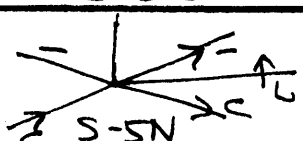
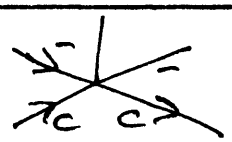
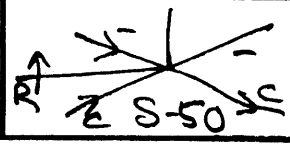
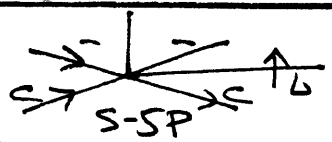
APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES	
ARROW-5B		 PEAK-5C	 PEAK-5D
ARROW-5C		 PEAK-5E	 PEAK-5F
ARROW-5D		PEAK-5G	 PEAK-5H
K-5A		 KX-5Q	 KA-5A
K-5B		 KX-5R	 KA-5B
K-5C		KX-5S	 KA-5C
K-5D		 KX-5T	 KA-5D
K-5E		 KX-5U	 KA-5E
K-5F		 KX-5V	 KA-5F
K-5G		 KA-5G	 KX-5W

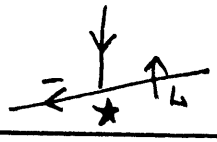
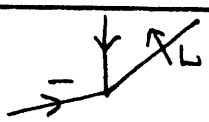
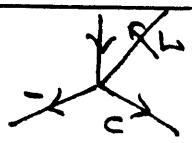
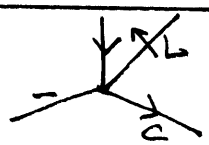
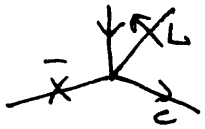
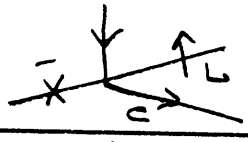
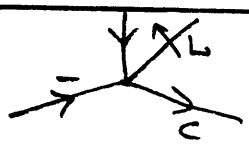
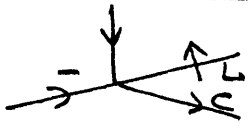
APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES	
K-5H		 KA-5H	 Kx-5X
K-5I		 KA-4I	 KX-5Y
K-5J		 KA-4J	 KX-5Z
K-5K		 KA-4K	 KX-5AA
K-5L		 KA-4L	 KX-5AB
KXX-5A		 S-5A	 S-5B
KXX-5B		 S-5C	 S-5D
KXX-5C		 S-5E	 S-5F
KXX-5D		 S-5G	 S-5H
KXX-5E		 S-5I	 S-5J

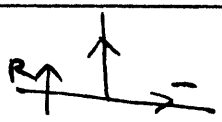
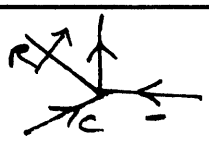
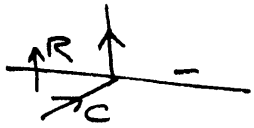
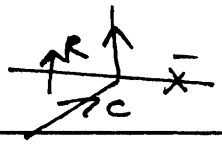
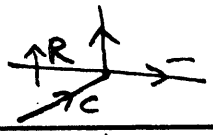
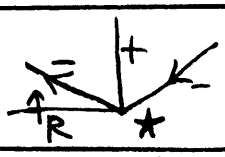
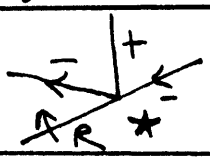
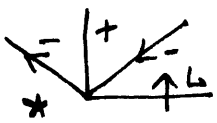
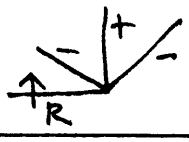
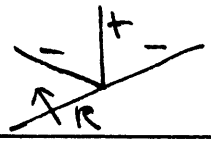
APPENDIX 2

NAME	APPEARANCE & LABELING	SHADOW VARIATIONS & NAMES	
KXX-5F		 S-5K	 S-5L
KXX-5G		 S-5M	 S-5N
KXX-5H		 S-5O	 S-5P

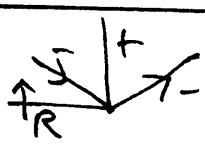
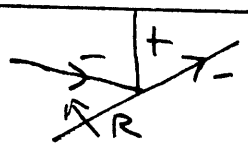
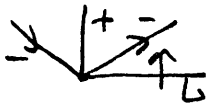
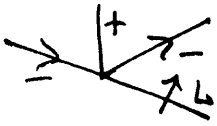
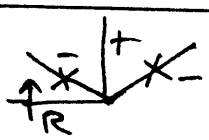
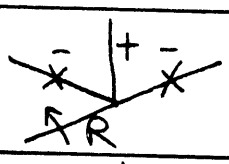
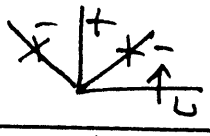
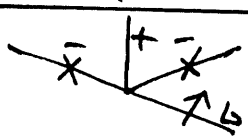
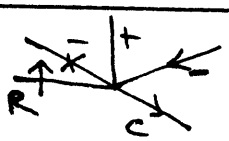
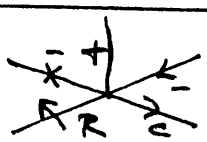
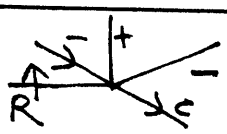
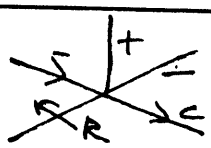
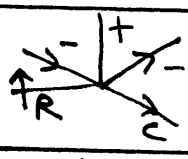
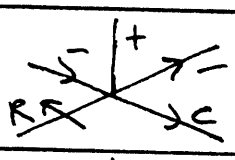
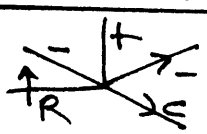
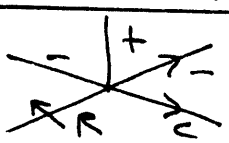
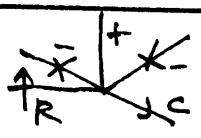
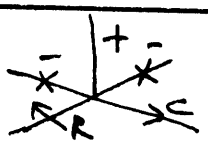
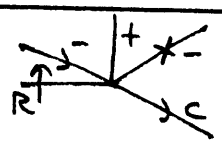
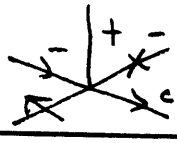
APPENDIX 3

SOURCE JUNCTION NAME	LABELING	ALIGNED LIGHTING JUNCTIONS & NAMES
ARROW-SF		 T-A11
ARROW-SG		T-A12
ARROW-SI		T-A13
ARROW-SK		T-A14
MULTI-SA		X-A11
MULTI-SB		X-A12
MULTI-SE		 X-A13
MULTI-SG		 X-A14
ARROW-SM		T-A15
ARROW-SO		T-A16

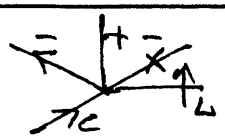
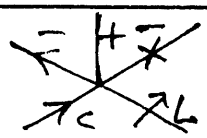
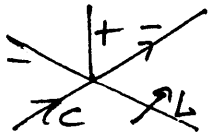
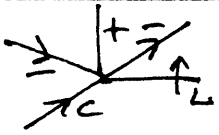
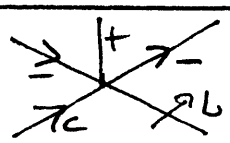
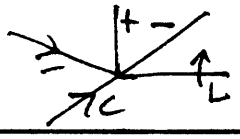
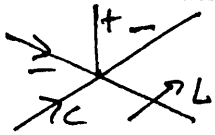
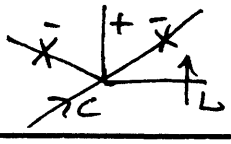
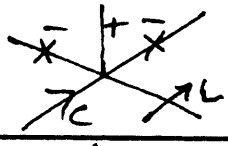
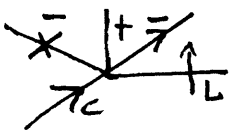
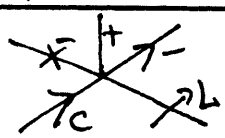
APPENDIX 3

SOURCE JUNCTION NAME	LABELING	ALIGNED LIGHTING JUNCTIONS & NAMES
ARROW-SQ		 T-A17
ARROW-SS		T-A18
MULTI-ST		X-A15
MULTI-SK		 X-A16
MULTI-SM		 X-A17
MULTI-SO		 X-A18
PEAK-SA		 K-A11
PEAK-SB		K-A12
PEAK-SC		 K-A13
PEAK-SD		K-A14

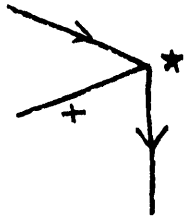
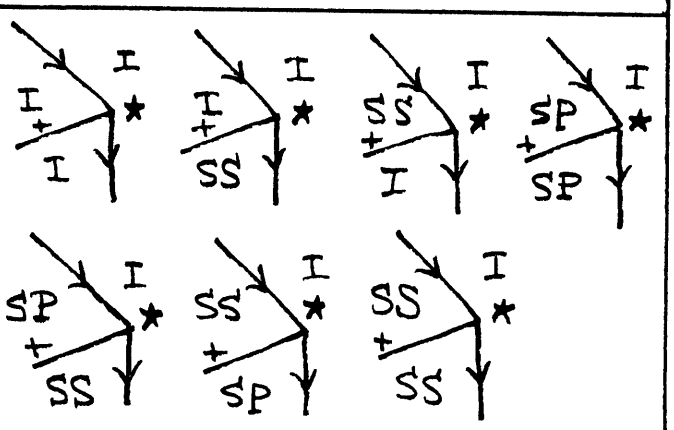
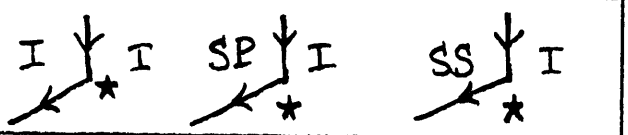
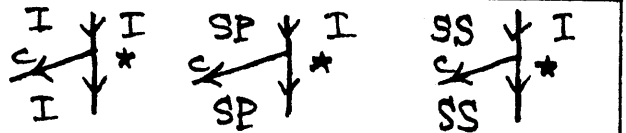
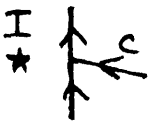
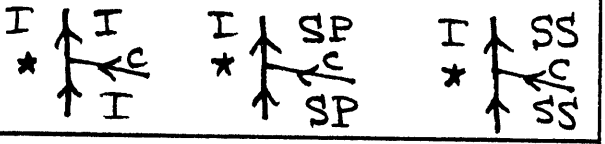
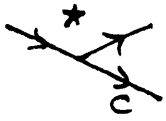
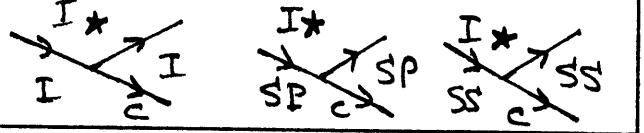
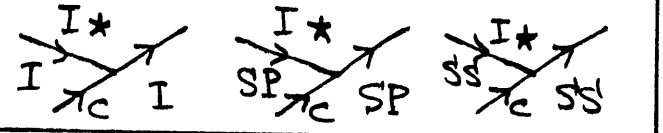
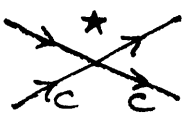
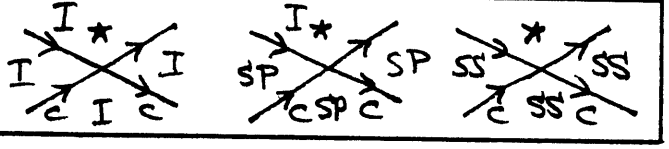
APPENDIX 3

SOURCE JUNCTION NAME	LABELING	ALIGNED LIGHTING JUNCTIONS & NAMES
PEAK-5E		 K-A15
PEAK-5F		 K-A16
PEAK-5G		 K-A17
PEAK-5H		 K-A18
KX-5Q		 KXX-A11
KX-5R		 KXX-A12
KX-5S		 KXX-A13
KX-5T		 KXX-A14
KX-5U		 KXX-A15
KX-5V		 KXX-A16

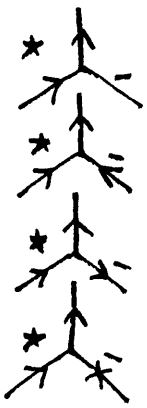
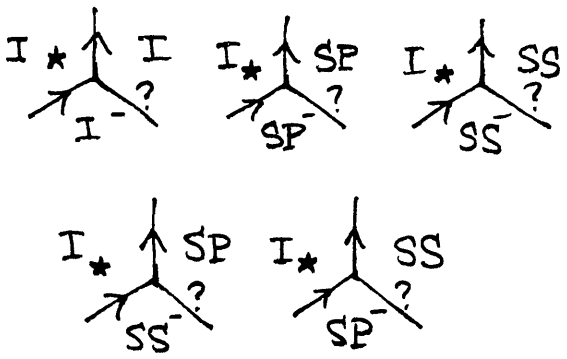
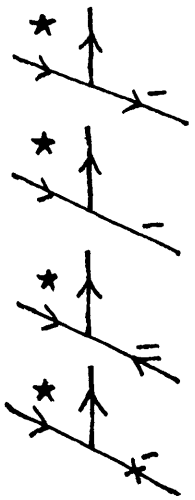
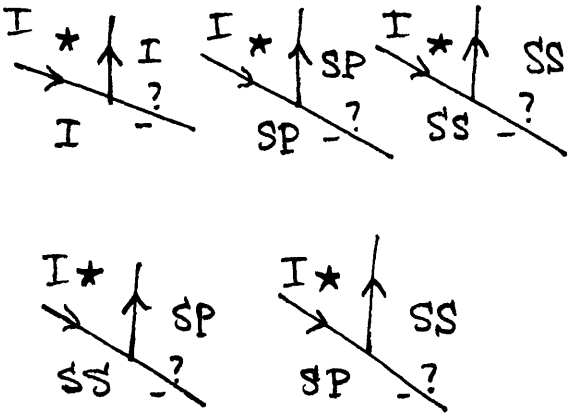
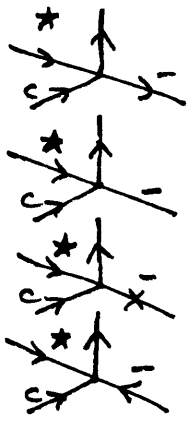
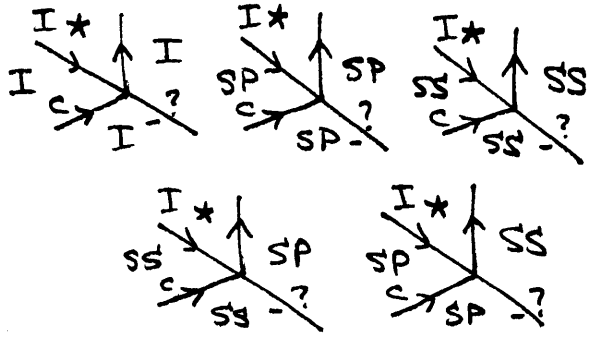
APPENDIX 3

SOURCE JUNCTION NAME	LABELING	ALIGNED LIGHTING JUNCTIONS & NAMES	
KX-5W			KXX-A1.7
KX-5X			KXX-A1.8
KX-5Y			KXX-A1.9
KX-5Z			KXX-A1.10
KX-5AA			KXX-A1.11
KX-5AB			KXX-A1.12

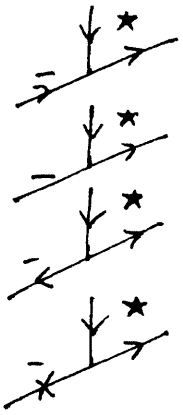
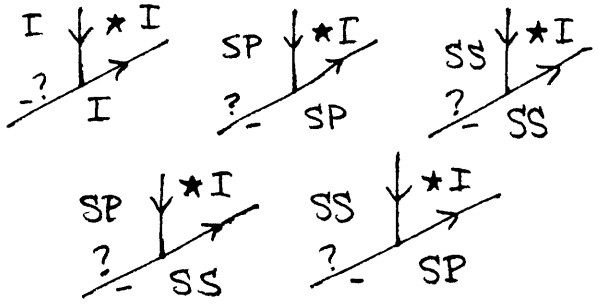
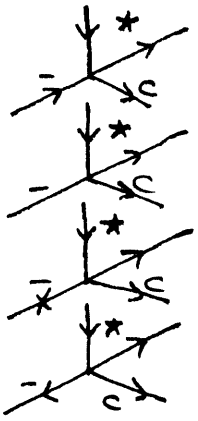
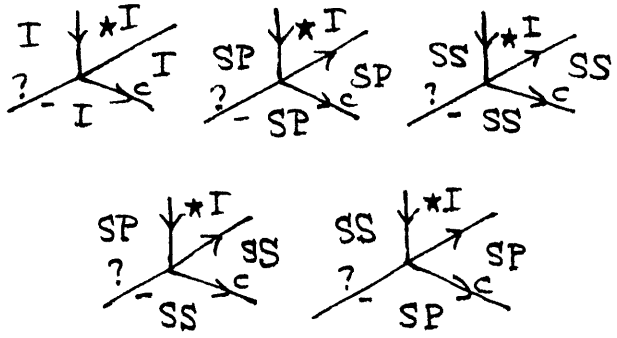
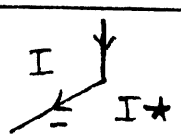
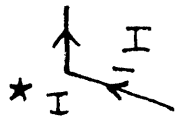
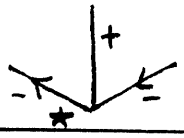
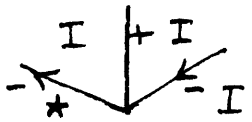
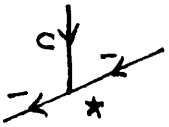
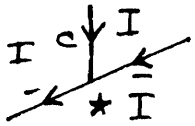
APPENDIX 4

SOURCE NAME	LABELING	SCENE/BACKGROUND BOUNDARY VARIATIONS
ARROW-1		
L-1		
T-2A		
T-2B		
L-3A		
T-3A		
T-3B		
XX-3A		

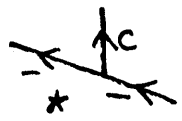
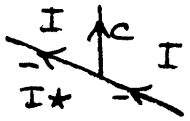
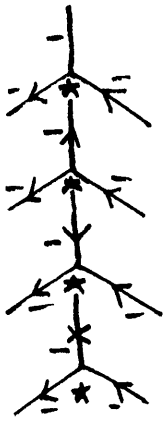
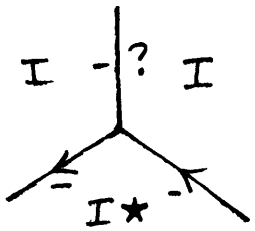
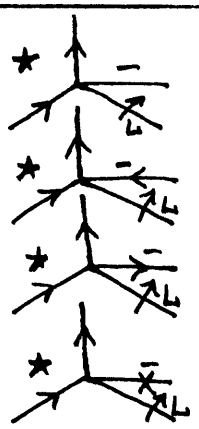
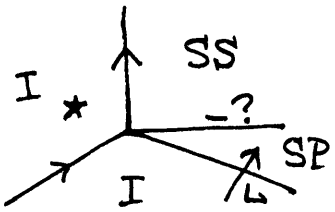
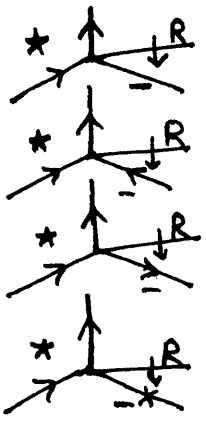
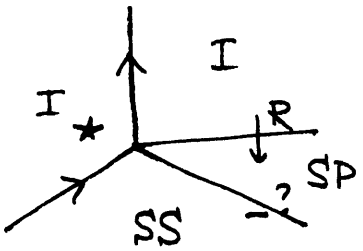
APPENDIX 4

SOURCE NAME	LABELING	SCENE/BACKGROUND BOUNDARY VARIATIONS
FORK-3A FORK-3B FORK-3C FORK-3D		ALL FORM THE SAME COMBINATIONS: 
T-4I T-4J T-4K T-4L		ALL FORM THE SAME COMBINATIONS: 
X-4E X-4F X-4G X-4H		ALL FORM THE SAME COMBINATIONS: 

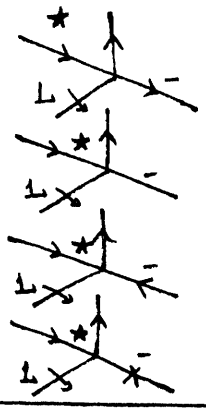
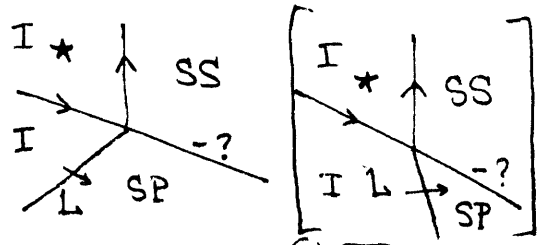
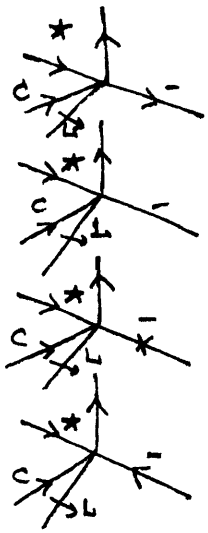
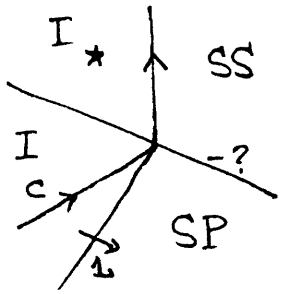
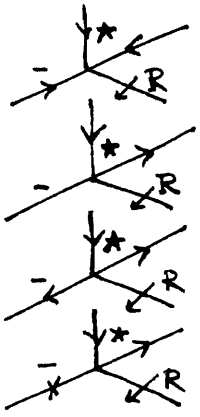
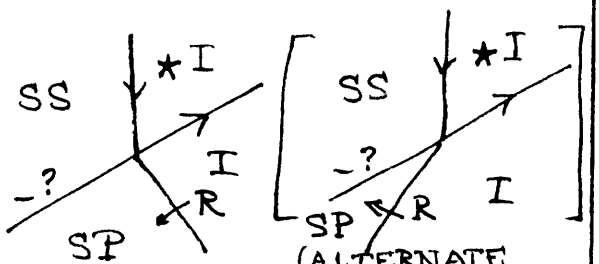
APPENDIX 4

SOURCE NAME	LABELING	SCENE/BACKGROUND BOUNDARY VARIATIONS
T-4M T-4N T-4O T-4P		ALL FORM THE SAME COMBINATIONS: 
X-4I X-4J X-4K X-4L		ALL FORM THE SAME COMBINATIONS: 
L-5A		
L-5E		
ARROW-5A		
T-6A		

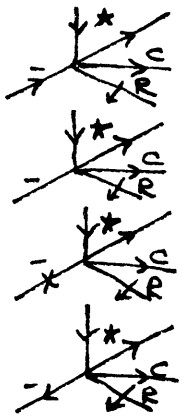
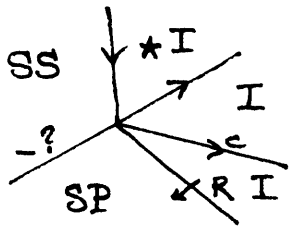
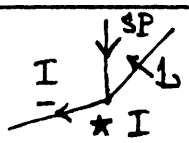
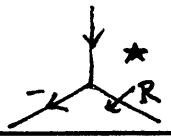
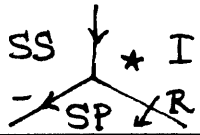
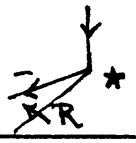
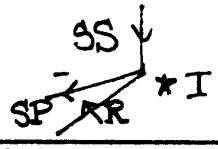
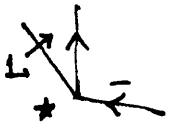
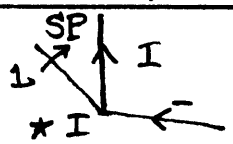
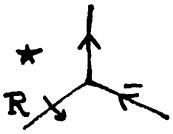
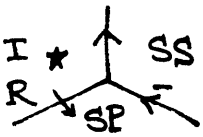
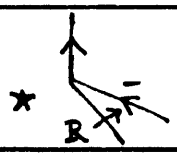
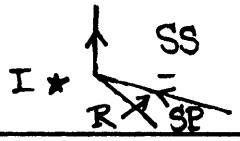
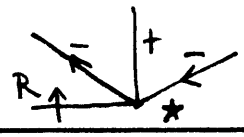
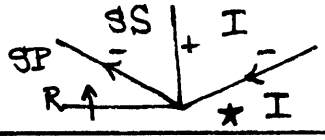
APPENDIX 4

SOURCE NAME	LABELING	SCENE/BACKGROUND BOUNDARY VARIATIONS
T-6P		
FORK-7B FORK-7C FORK-7N FORK-7P		ALL FORM THE SAME COMBINATIONS: 
MULTI-3A MULTI-3C MULTI-3E MULTI-3G		ALL FORM THE SAME COMBINATIONS: 
MULTI-3B MULTI-3D MULTI-3F MULTI-3H		ALL FORM THE SAME COMBINATIONS: 

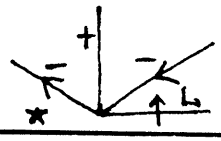
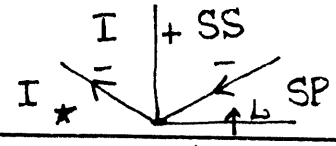
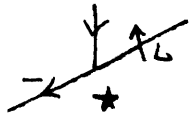
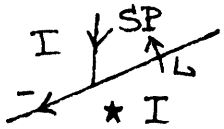
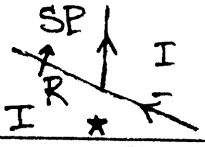
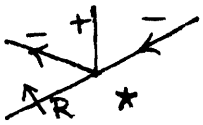
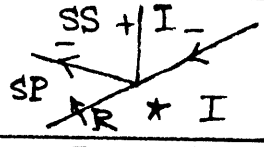
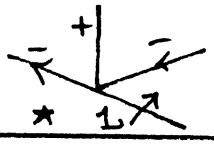
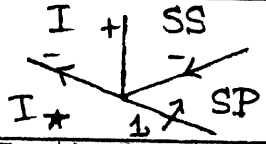
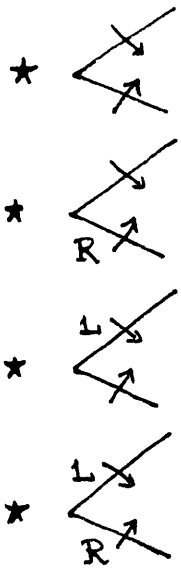
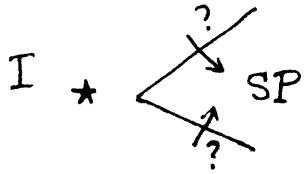
APPENDIX 4

SOURCE NAME	LABELING	SCENE / BACKGROUND BOUNDARY VARIATIONS
X-4U X-4V X-4W X-4X		ALL FORM THE SAME COMBINATIONS:  (ALTERNATE INTERPRETATION; SEE SECTION 4.2)
KX-4K KX-4L KX-4M KX-4N		ALL FORM THE SAME COMBINATIONS: 
X-4Y X-4Z X-4AA X-4AB		ALL FORM THE SAME COMBINATIONS:  (ALTERNATE INTERPRETATION; SEE SECTION 4.2)

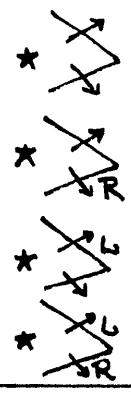
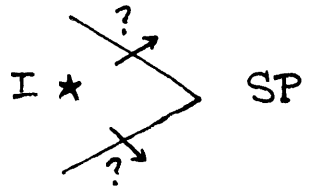
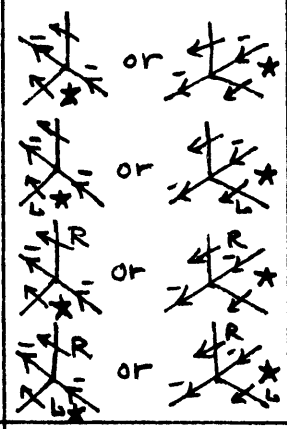
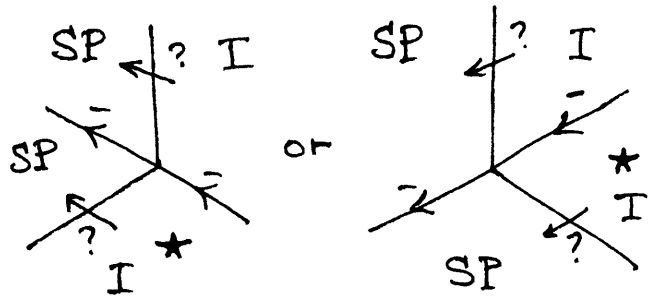
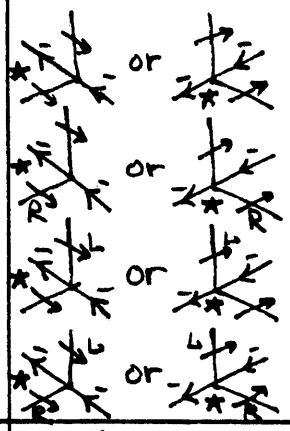
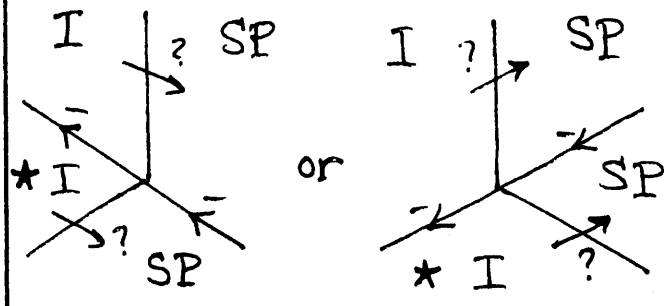
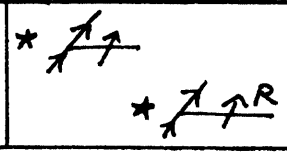
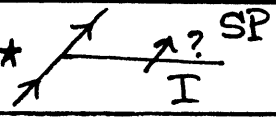
APPENDIX 4

SOURCE NAME	LABELING	SCENE/BACKGROUND BOUNDARY VARIATIONS
KX-4O KX-4P KX-4Q KX-4R		ALL FORM THE SAME COMBINATIONS: 
ARROW-5E		
FORK-5I		
ARROW-5F		
ARROW-5M		
FORK-5M		
ARROW-5N		
PEAK-5A		

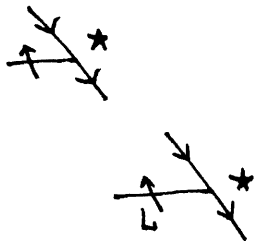
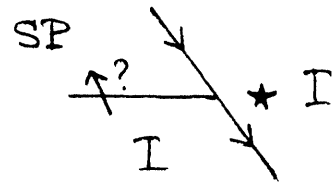
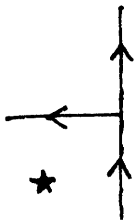
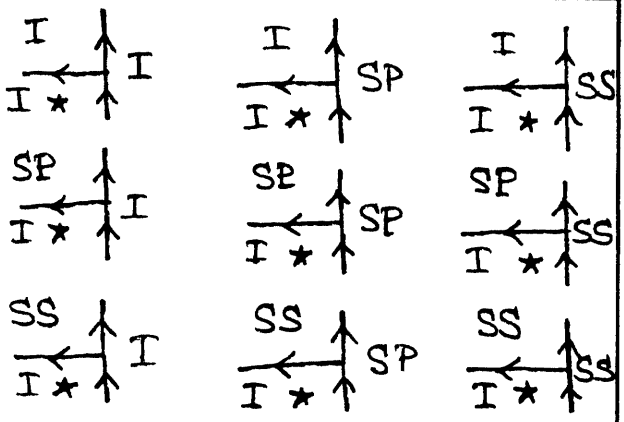
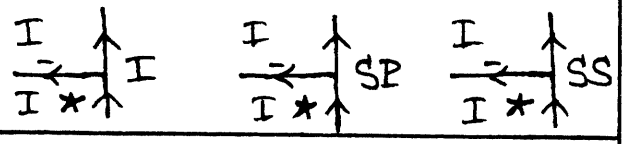
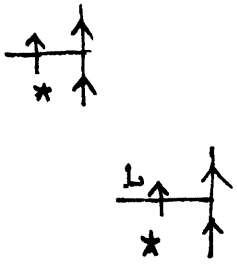
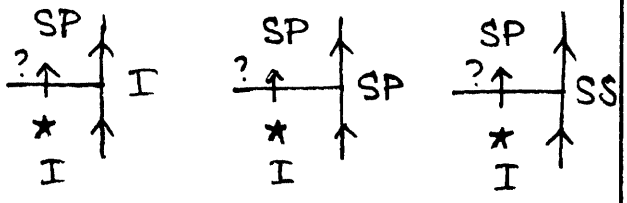
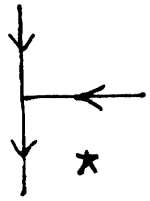
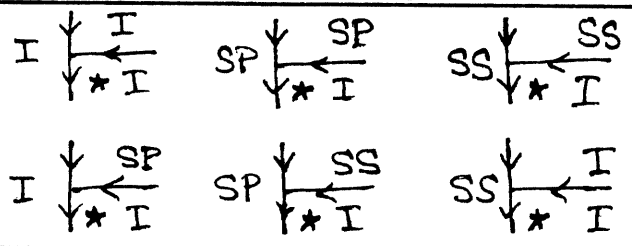
APPENDIX 4

SOURCE NAME	LABELING	SCENE/BACKGROUND BOUNDARY VARIATIONS
PEAK-5B		
T-AL1		
T-AL5		
K-A11		
K-AL2		
L-0A L-0B L-0C L-0D		ALL THESE HAVE THE SAME REGION ILLUMINATION ASSIGNMENT: 

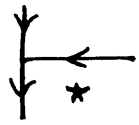
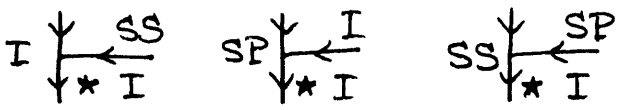
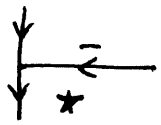
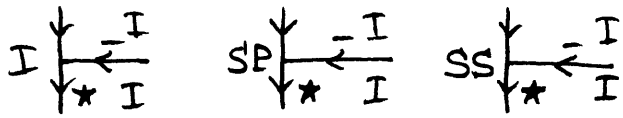
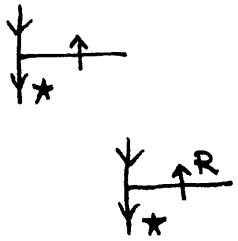
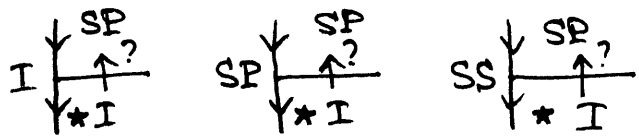
APPENDIX 4

SOURCE NAME	LABELING	SCENE / BACKGROUND BOUNDARY VARIATIONS
L-OE L-OF L-OG L-OH		THESE ALL HAVE THE SAME REGION ILLUMINATION ASSIGNMENT: 
X-OA X-OB X-OC X-OD		THESE ALL HAVE THE SAME REGION ILLUMINATION ASSIGNMENTS: 
X-OE X-OF X-OG X-OH		THESE ALL HAVE THE SAME REGION ILLUMINATION ASSIGNMENTS: 
T-OC T-OD		BOTH HAVE THE SAME ASSIGNMENTS: 

APPENDIX 4

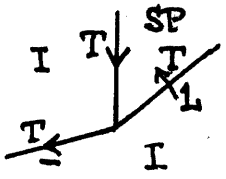
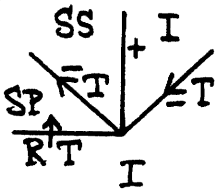
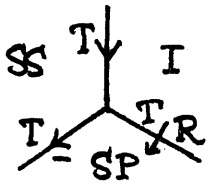
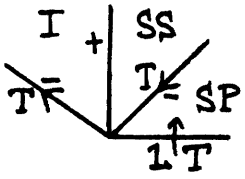
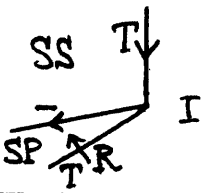
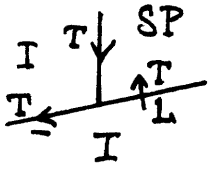
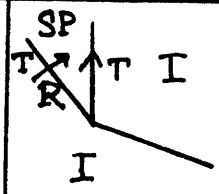
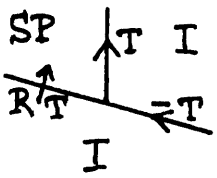
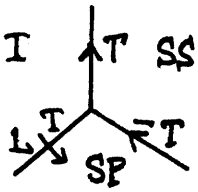
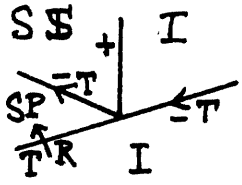
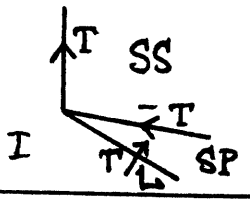
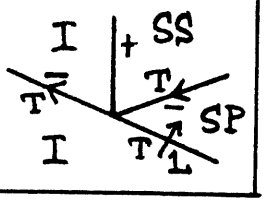
SOURCE NAME	LABELING	SCENE/BACKGROUND BOUNDARY VARIATIONS
T-0G T-0H		BOTH HAVE THE SAME ASSIGNMENTS: 
T-0I		
T-0J		
T-0K T-0L		BOTH HAVE THE SAME ASSIGNMENTS: 
T-0M (CONTINUED ON NEXT PAGE)		SEE NEXT PAGE FOR REST: 

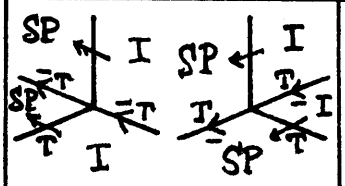
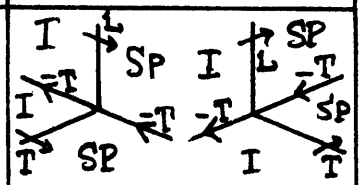
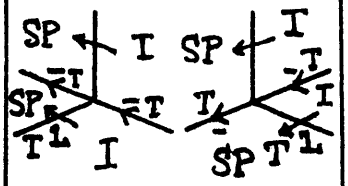
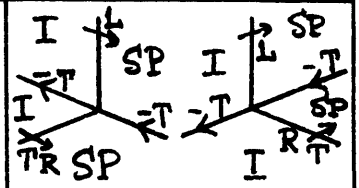
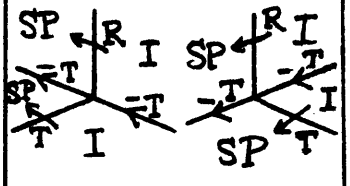
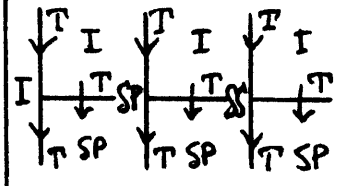
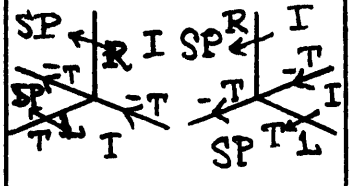
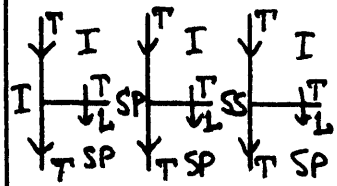
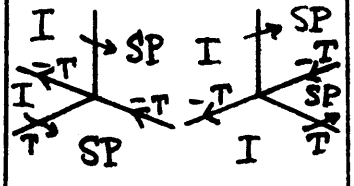
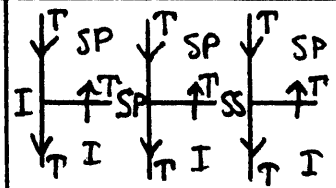
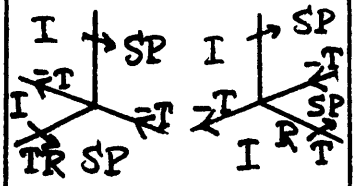
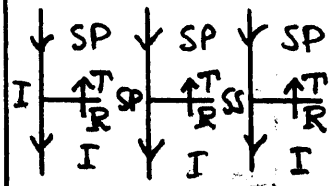
APPENDIX 4

SOURCE NAME	LABELING	SCENE/BACKGROUND BOUNDARY VARIATIONS
T-OM (CONTINUED)		
T-ON		
T-OO T-OP		BOTH HAVE THE SAME ASSIGNMENTS: 

APPENDIX 5

JUNCTION LABELINGS WHICH CANNOT BE MODIFIED TO INCLUDE TABLE INFORMATION BY SIMPLY ADDING "T" TO THE TWO LINE SEGMENTS THAT CAN APPEAR ON THE SCENE/BACKGROUND BOUNDARY IN LABELINGS FROM APPENDIX 4.

JUNCTION NAME	NEW LABELING	JUNCTION NAME	NEW LABELING
ARROW-5F		PEAK-5A	
FORK-5I		PEAK-5B	
ARROW-5F		T-AL1	
ARROW-5M		T-AL5	
FORK-5M		K-AL1	
ARROW-5N		K-AL2	

JUNCTION NAME	NEW LABELINGS	JUNCTION NAME	NEW LABELINGS
X-0A		X-0G	
X-0B		X-0H	
X-0C		T-0K	
X-0D		T-0L	
X-0E		T-00	
X-0F		T-0P