2 UNDERSTANDING LINE DRAWINGS OF SCENES WITH SHADOWS

David Waltz

2.1 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 Figs. 2.1 and 2.2? 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?

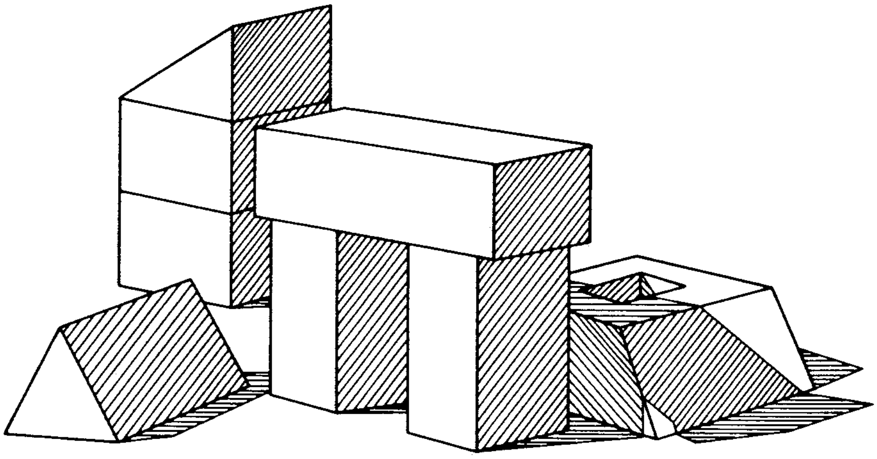


Fig. 2.1

In this essay 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. 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 spatial orientation of various regions, all using the description it has generated.

2.1.1 Descriptions

The overall goal of the system is to provide a precise description of a plausible scene which could give rise to a particular line drawing. It is therefore

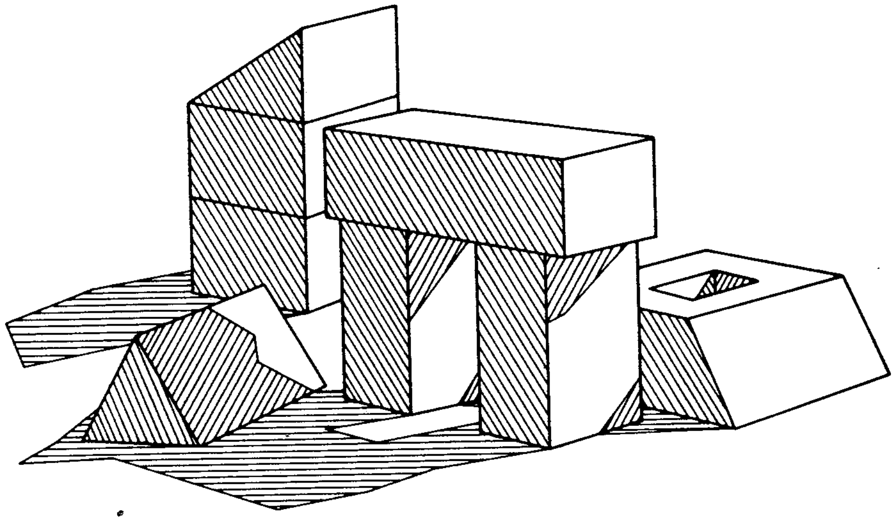


Fig. 2.2

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[1] in the United States and Clowes[2] 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, 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.

2.1.2 Junction Labels

Much of the program's power is based on access to lists of possible line label assignments for each type of junction in a line drawing. 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 is interesting in its own right. While some junction types require many dictionary entries, others require relatively few. Moreover, in some cases local information about the relative brightness of the surrounding regions and about the directions of the lines may severely limit the number of relevant dictionary entries for any particular junction. In other cases such information has little effect.

Figure 2.3 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). There is a considerable amount of local information which can be used to select a subset of the total number of junction configurations which are consistent with physical reality.

For example the program can use local region brightness and line segment direction to preclude the assignment of certain labels to lines. If it

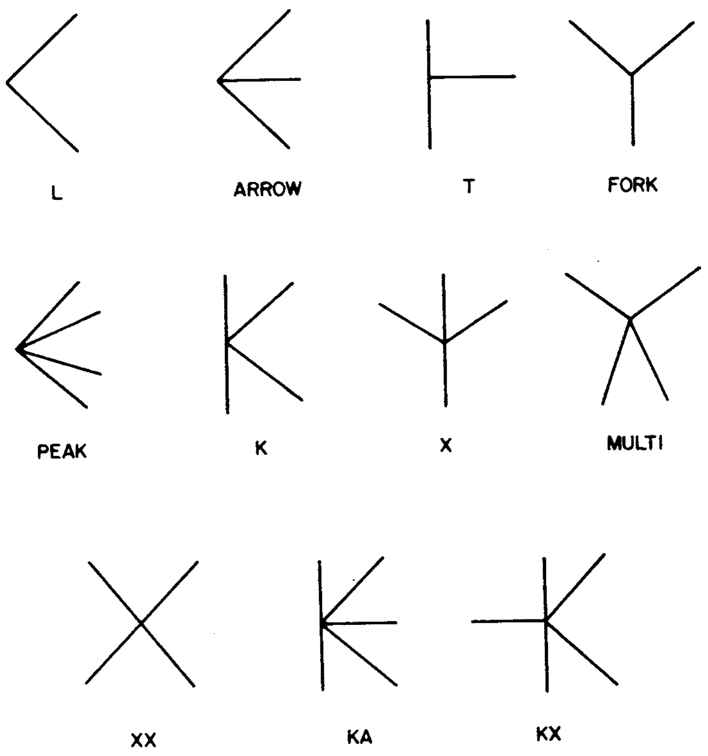


Fig. 2.3

knows that one region is brighter than an adjacent region, then the line which separates the regions can be labeled as a shadow region in only one way. There are other rules which relate region orientation, light placement and region illumination as well as rules which limit the number of labels which can be assigned to line segments which border the support surface for the scene. The program is able to combine all these types of information in finding a list of appropriate labels for a single junction.

2.1.3 Combination Rules

Combination rules are used to select 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.

I thought at the outset of my work that it might be necessary to construct models of hidden vertexes 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, it may 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.

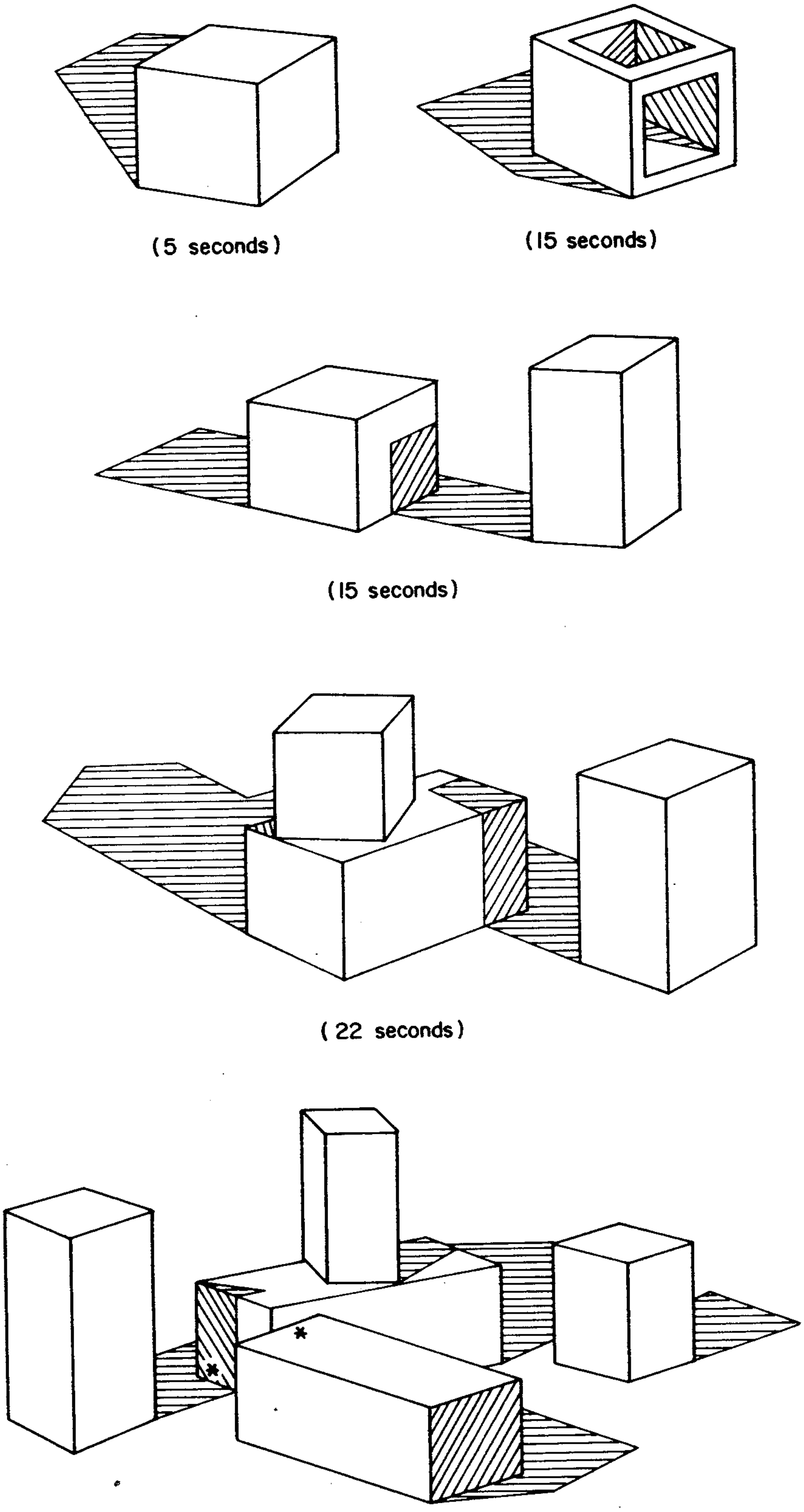
2.1.4 Experimental Results

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 a 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 an 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.

Initially I had intended to have a tree search program follow the filtering program, but to my amazement I found that in the first few scenes I tried, this program alone 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 recognize 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.

Figure 2.4 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[3] and partially in compiled LISP.



(5 seconds)

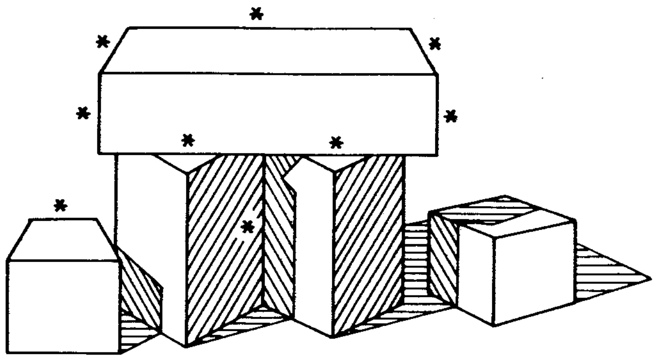
(15 seconds)

(15 seconds)

(22 seconds)

(39 seconds)

Fig. 2.4



(48 seconds)

Fig. 2.4 *(continued)*

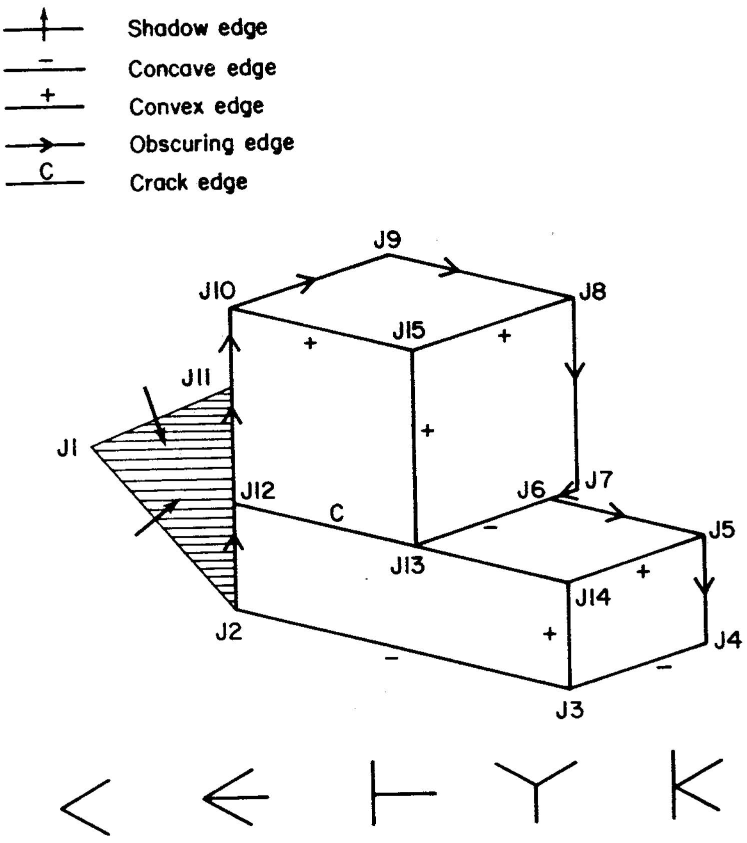
2.2 LINE LABELS

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 a 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[1] and Clowes.[2] Thus, in Fig. 2.5 line segment J1-J2 is labeled as a shadow edge, line J2-J3 is labeled as a concave edge, line J3-J14 is labled 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.

Pay particular attention to the notation used to label the lines. When I talk of junction labels I refer to the various possible combinations of such line labels around a junction. Each such combination is thought of as a particular junction labeling.

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?



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

Fig. 2.5

2.2.1 System Knowledge

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 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 section 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 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 (1) which represented trihedral vertexes (i.e., vertexes 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 vertexes 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.

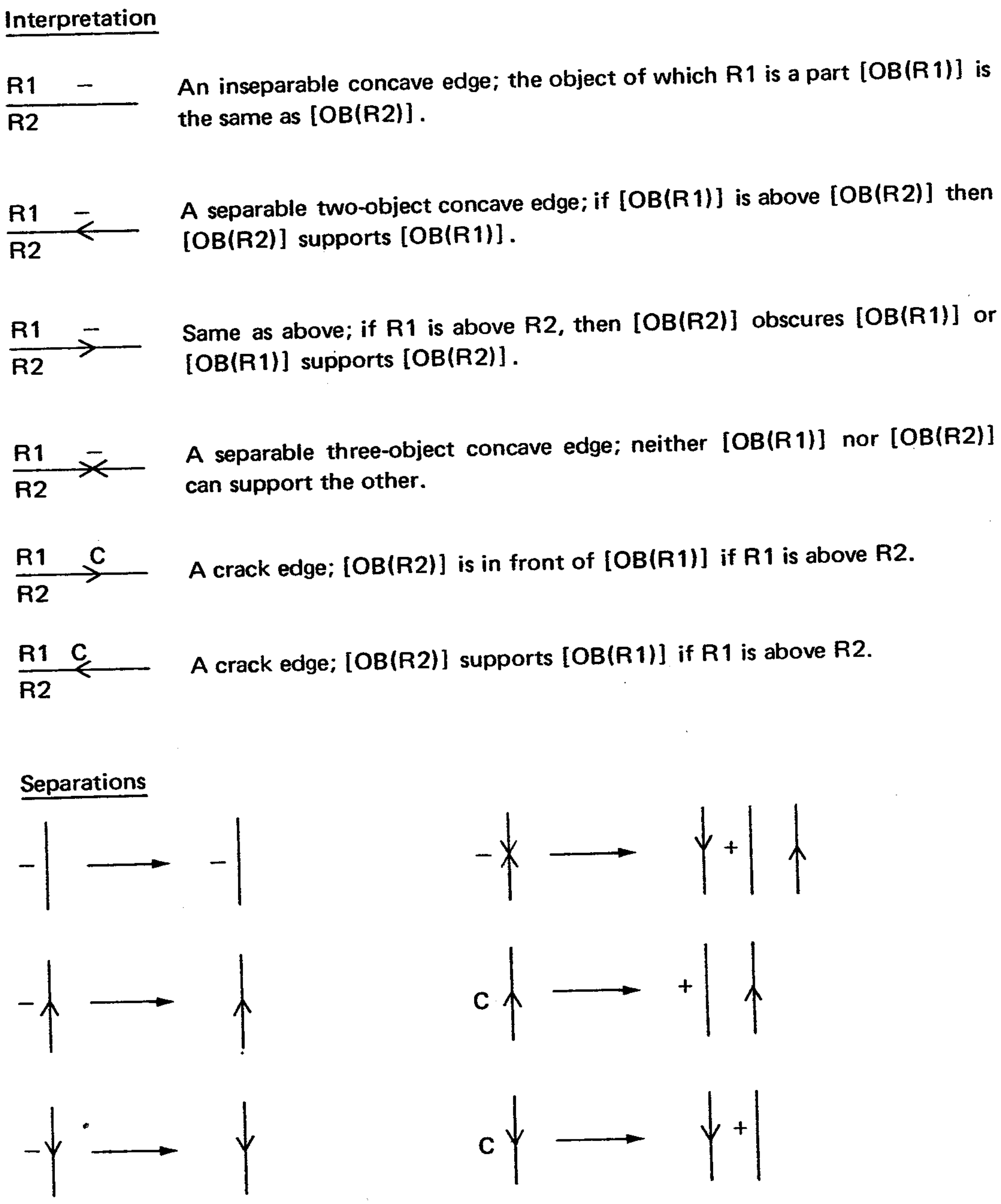
Each type of junction (L, ARROW, FORK) can only be labeled in a relatively small number of ways; thus if we can say with certainty what the label for a particular line must be, we can greatly constrain all other lines which intersect that line segment. As a specific example, if one branch of an L junction 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.

2.2.2 Better Edge Description

So far I have classified edges on the basis of geometry (concave, convex, obscuring, or planar) and have subdivided the planar class into crack and shadow subclasses. 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 2.6



An inseparable concave edge; the object of which R1 is a part [OB(R1)] is the same as [OB(R2)].

A separable two-object concave edge; if [OB(R1)] is above [OB(R2)] then [OB(R2)] supports [OB(R1)].

Same as above; if R1 is above R2, then [OB(R2)] obscures [OB(R1)] or [OB(R1)] supports [OB(R2)].

A separable three-object concave edge; neither [OB(R1)] nor [OB(R2)] can support the other.

A crack edge; [OB(R2)] is in front of [OB(R1)] if R1 is above R2.

A crack edge; [OB(R2)] supports [OB(R1)] if R1 is above R2.

Fig. 2.6

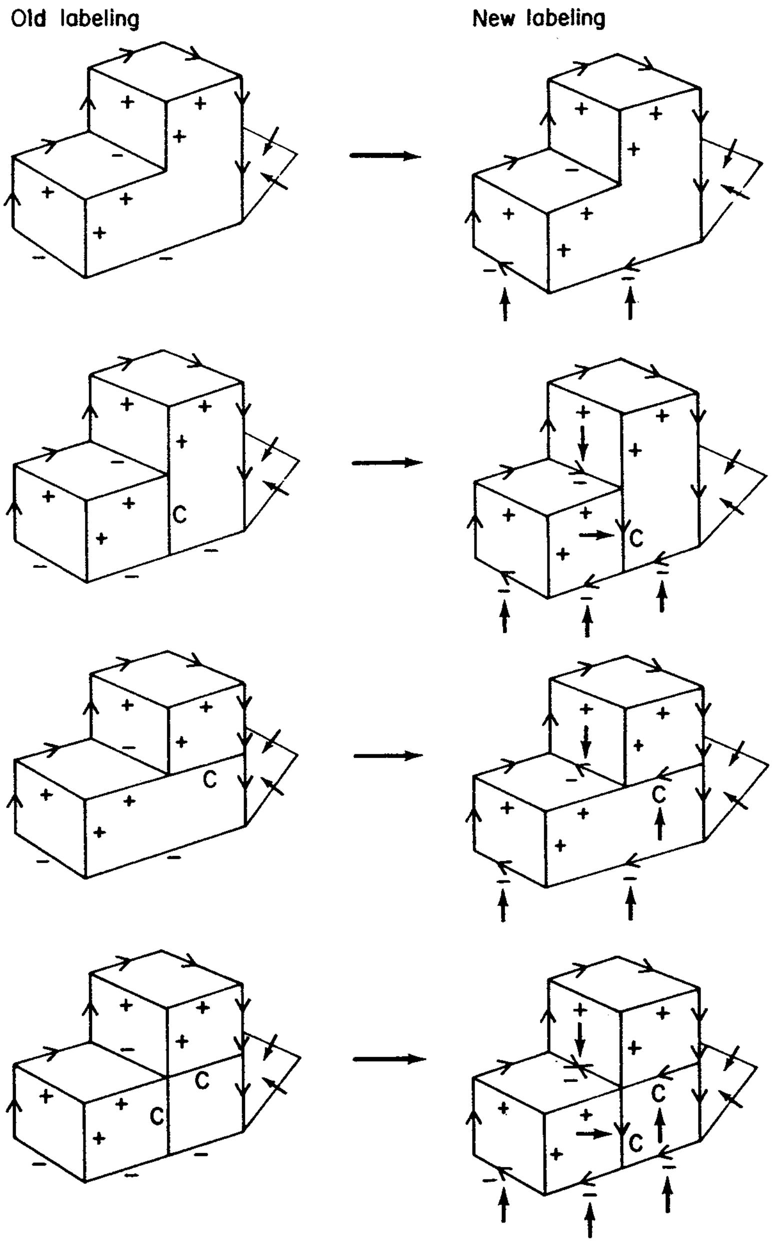


Fig. 2.6 *(continued)*

shows the results of appending a label analogous to the "obscuring edge" mark to crack and concave edges. This approach is similar to one first proposed by Freuder.[4]

2.2.3 Edge Geometry

The first problem is to find all possible trihedral vertexes. Huffman observed that three intersecting planes, whether mutually orthogonal or not, divide space into eight parts so that the types of trihedral vertex can be

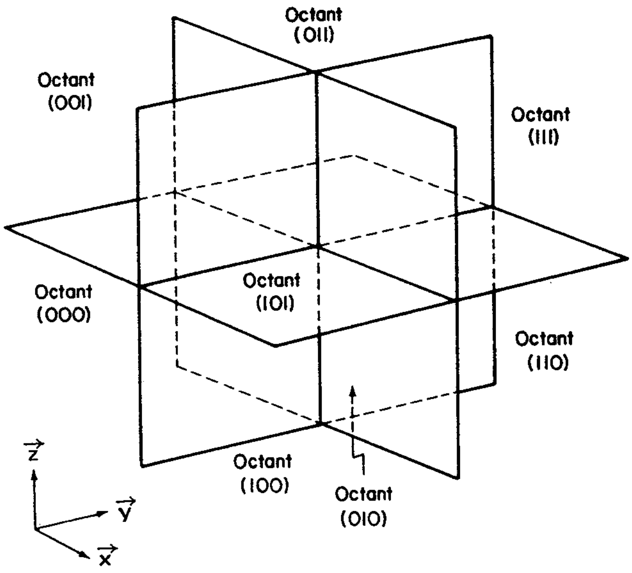


Fig. 2.7

characterized by the octants of space around the vertex which are filled by solid material.[1]

Consider the general intersection of three planes shown in Fig. 2.7. These planes divide space into octants, which can be uniquely identified by three-dimensional binary vectors (x y 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 nondegenerate views by imagining 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 2.8 shows the ten possible geometries that result from filling various octants; when considered from all possible viewing positions these ten geometries produce 196 different junction labelings. There are some other 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 and Clowes correspond to my numbers 1, 3, 5, and 7 in Fig. 2.8.

In Fig. 2.9 I show how the 20 different labels with type 3 geometry can be generated. Basically this process involves taking a geometry from Fig. 2.8, finding all the ways that the solid segments can be connected or separated, and finding all the possible views for each partitioning of the octants. 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.

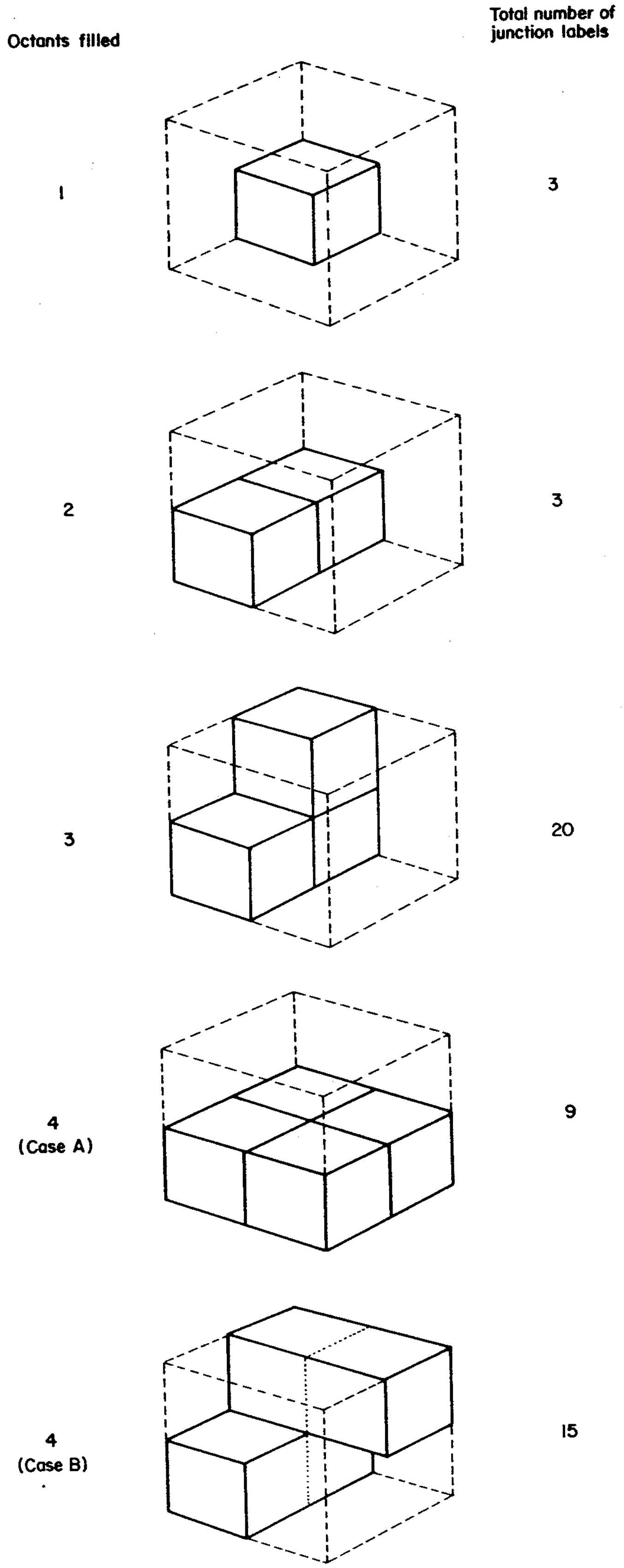

Octants filled
Total number of junction labels
1
3
2
3
3
20
4 (Case A)
9
4 (Case B)
15

Fig. 2.8

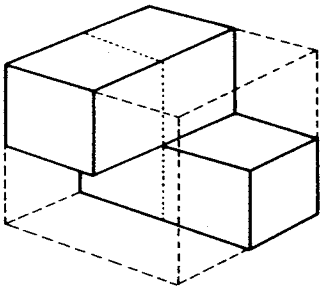
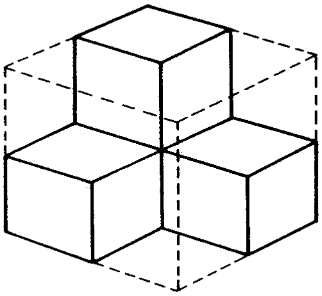
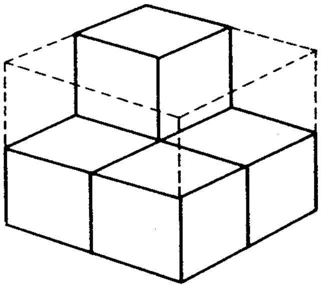
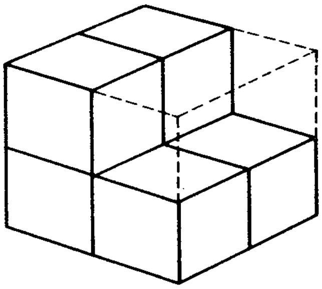
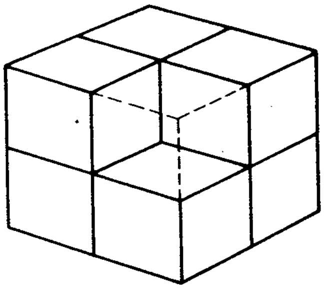
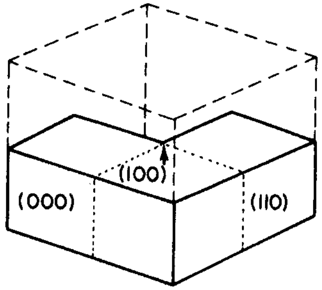
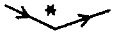
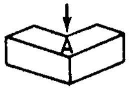
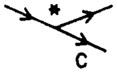
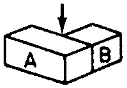
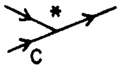
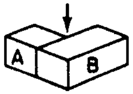
Octants filled		Total number of junction labels
4 (Case C)		15
4 (Case D)		8
5		56
6		46
7		21

Fig. 2.8 *(continued)*



Name of junction	Appearance and labeling of junction	Number 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)
				

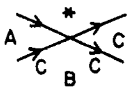
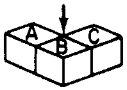
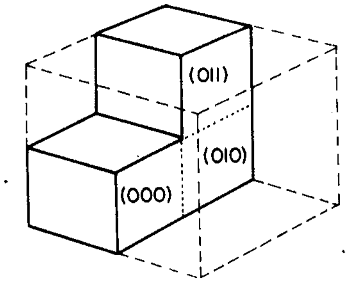
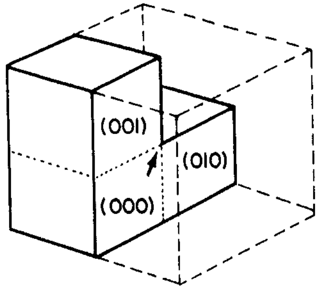
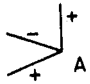
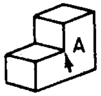
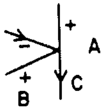
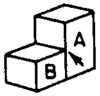
XX-3A		3	A = (000)
			B = (100)
			C = (110)
				

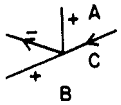
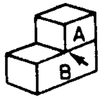
Fig. 2.9

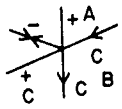
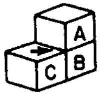


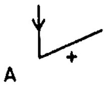
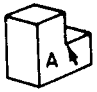


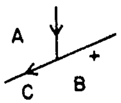
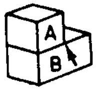
Name of junction	Appearance and labeling of junction	Number of objects at vertex	Objects at vertex are	Appearance of corresponding object(s)
Arrow-3A		1	A = (000) ∪ (010) ∪ (011)	

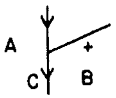
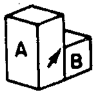
K-3A		2	A = (010) ∪ (011)
			B = (000)
				

K-3B		2	A = (011)
			B = (000) ∪ (010)
				

KXX-3A		3	A = (011)
			B = (010)
			C = (000)
				

Name of junction	Appearance and labeling of junction	Number 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)
				

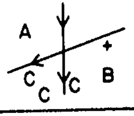
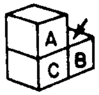
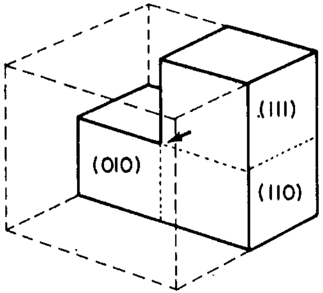
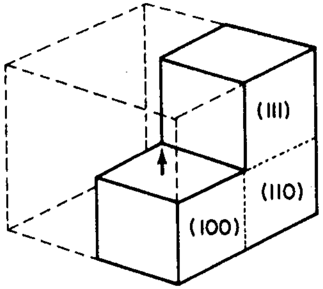
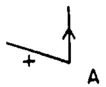
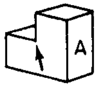
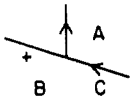
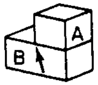
XX-3B		3	A = (001)
			B = (010)
			C = (000)
				

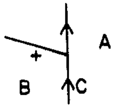
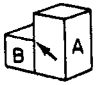
Fig. 2.9 *(continued)*

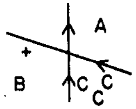
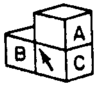


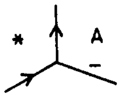
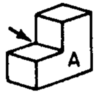


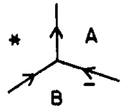
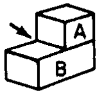
Name of junction	Appearance and labeling of junction	Number of objects at vertex	Objects at vertex are	Appearance of corresponding object(s)
L-3C		1	A = (111) ∪ (110) ∪ (010)	

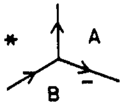
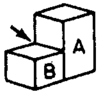
T-3E		2	A = (111)
			B = (110) ∪ (010)
				

T-3F		2	A = (111) ∪ (110)
			B = (010)
				

XX-3C		3	A = (111)
			B = (010)
			C = (110)
				

Name of junction	Appearance and labeling of junction	Number 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)
				

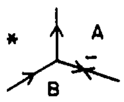
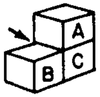
Fork-3D		3	A = (111)
			B = (100)
			C = (110)
				

Fig. 2.9 *(continued)*