

AI TR-281

PROGRESS IN VISION AND ROBOTICS

Patrick H. Winston, Editor

MAY 1973

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Cambridge

Massachusetts 02139

MIT Document Services

Room 14-0551
77 Massachusetts Avenue
Cambridge, MA 02139
ph: 617/253-5668 | fx: 617/253-1690
email: docs@mit.edu
<http://libraries.mit.edu/docs>

DISCLAIMER OF QUALITY

Due to the condition of the original material; there are unavoidable flaws in this reproduction. We have made every effort to provide you with the best copy available. If you are dissatisfied with this product and find it unusable, please contact Document Services as soon as possible.

Thank you.

Poor quality typeset

INTRODUCTION

The Vision Flashes are informal working papers intended primarily to stimulate internal interaction among participants in the A.I. Laboratory's Vision and Robotics group. Many of them report highly tentative conclusions or incomplete work. Others deal with highly detailed accounts of local equipment and programs that lack general interest. Still others are of great importance, but lack the polish and elaborate attention to proper referencing that characterizes the more formal literature.

Nevertheless, the Vision Flashes collectively represent the only documentation of an important fraction of the work done in machine vision and robotics. The purpose of this report is to make the findings more readily available, but since they are not revised as presented here, readers should keep in mind the original purpose of the papers!

Many report on details of the M.I.T. blocks world vision system. The entire spectrum of vision processing is represented, from low level feature finding to high level scene analysis requiring extensive world knowledge and deductive power. On all levels, they reflect a movement from ad hoc programs and testing toward

the sound theory that one expects from successful science.

The most recent papers shift attention away from the now well understood plane polyhron world and toward two new foci:

- 1) real world vision

- 2) applications for machines with vision. Careful study will show that work in these new areas is productively guided by the evolving ideas and metaphors of artificial intelligence in general and by our earlier work in simple visual worlds.

SUMMARY OF SELECTED VISION TOPICS

Patrick F. Winston

July 1972

ABSTRACT

This is an introduction to some of the M.I.T. A.I. vision work of the last few years. The topics discussed are 1) Waltz's work on line drawing semantics 2) heterarchy 3) the ancient learning business and 4) copying scenes. All topics are discussed in more detail elsewhere in vision flashes or theses.

This thesis was originally published as Vision Flash 30.

INTRODUCTION

Research in machine vision is an important activity in artificial intelligence laboratories for two major reasons: First, understanding vision is a worthy subject for its own sake. The point of view of artificial intelligence allows a fresh new look at old questions and exposes a great deal about vision in general, independent of whether man or machine is the seeing agent. Second, the same problems found in understanding vision are of central interest in the development of a broad theory of intelligence. Making a machine see brings one to grips with problems like that of knowledge interaction on many levels and of large system organization. In vision these key issues are exhibited with enough substance to be nontrivial and enough simplicity to be tractable.

These objectives have led vision research at MIT to focus on two particular goals: learning from examples and copying from spare parts. Both goals are framed in terms of a world of bricks, wedges, and other simple shapes like those found in children's toy boxes.

Good purposeful description is often fundamental to research in artificial intelligence, and learning how to do description constitutes a major part of our effort in vision research. This essay begins with a discussion of that part of scene analysis known as body finding. The intention is to show how our understanding has evolved away from blind fumbling toward

substantive theory.

The next section revolves around the organizational metaphors and the rules of good programming practice appropriate for thinking about large knowledge-oriented systems. Finding groups of objects and using the groups to get at the properties of their members illustrates concretely how some of the ideas about systems work out in detail.

The topic of learning follows. Discussing learning is especially appropriate here not only because it is an important piece of artificial intelligence theory but also because it illustrates a particular use for the elaborate analysis machinery dealt with in the previous sections.

EVOLUTION OF A SEMANTIC THEORY

Guzman and the Body Problem

The body finding story begins with an ad hoc but crisp syntactic theory and ends in a simple, appealing theory with serious semantic roots. In this the history of the body finding problem seems paradigmatic of vision system progress in general.

Aldolfo Guzman started the work in this area (Guzman 1968). I review his program here in order to anchor the discussion and show how better programs emerge through the interaction of observation, experiment, and theory.

The task is simply to partition the observed regions of a scene into distinct bodies. In figure 1, for example, a reasonable program would report something like (A F C) and (D E) as a plausible partitioning of the five regions into, in this case, two bodies. Keep in mind that the program is after only one good, believable answer. Many simple scenes have several equally justifiable interpretations.

Guzman's program operates on scenes in two distinct passes, both of which are quite straightforward. The first pass gathers local evidence and the second weighs that evidence and offers an opinion about how the regions should be grouped together into bodies.

The local evidence pass uses the vertices to generate little pieces of evidence indicating which of the surrounding regions belong to the same body. These quanta of evidence are

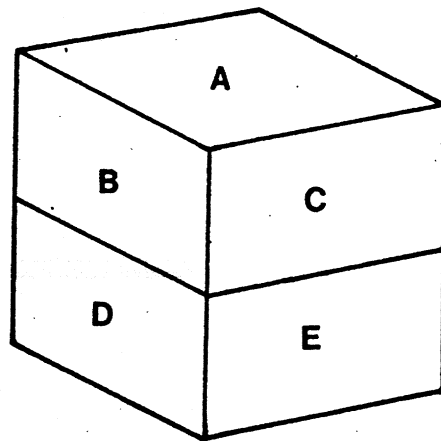


Figure 1
The task of the body finding program is to understand how the regions of the scene form bodies.

called links. Figure 2 lists each vertex type recognized and shows how each contributes to the set of links. The arrow links always argue that the shaft-bordering regions belong together, the fork more ambitiously provides three such links, one for each pair of surrounding regions, and so on. The resulting links for the scene in figure 1 are displayed superimposed on the original drawing in figure 3a. Internally the links are represented in list structure equivalent to the abstract diagram in figure 3b. There the circles each represent the correspondingly lettered region from figure 3a. The arcs joining the circles represent links.

The job of pass two is to combine the link evidence into a parsing hypothesis. How Guzman's pass two approached its final form may be understood by imagining a little series of theories about how to use the evidence to best advantage. Figure 3a is so simple that almost any method will do. Consequently figure 4 and figure 5 are used to further illustrate the experimental observations behind the evolving sequence of theories.

The first theory to think about is very simple. It argues that any two regions belong to the same body if there is a link between them. The theory works fine on many scenes, certainly on those in figure 3a and figure 4. It is easy, however, to think of examples that fool this theory because it is far too inclined toward enthusiastic region birding. Wherever a coincidence produces an accidental link, as for example the links

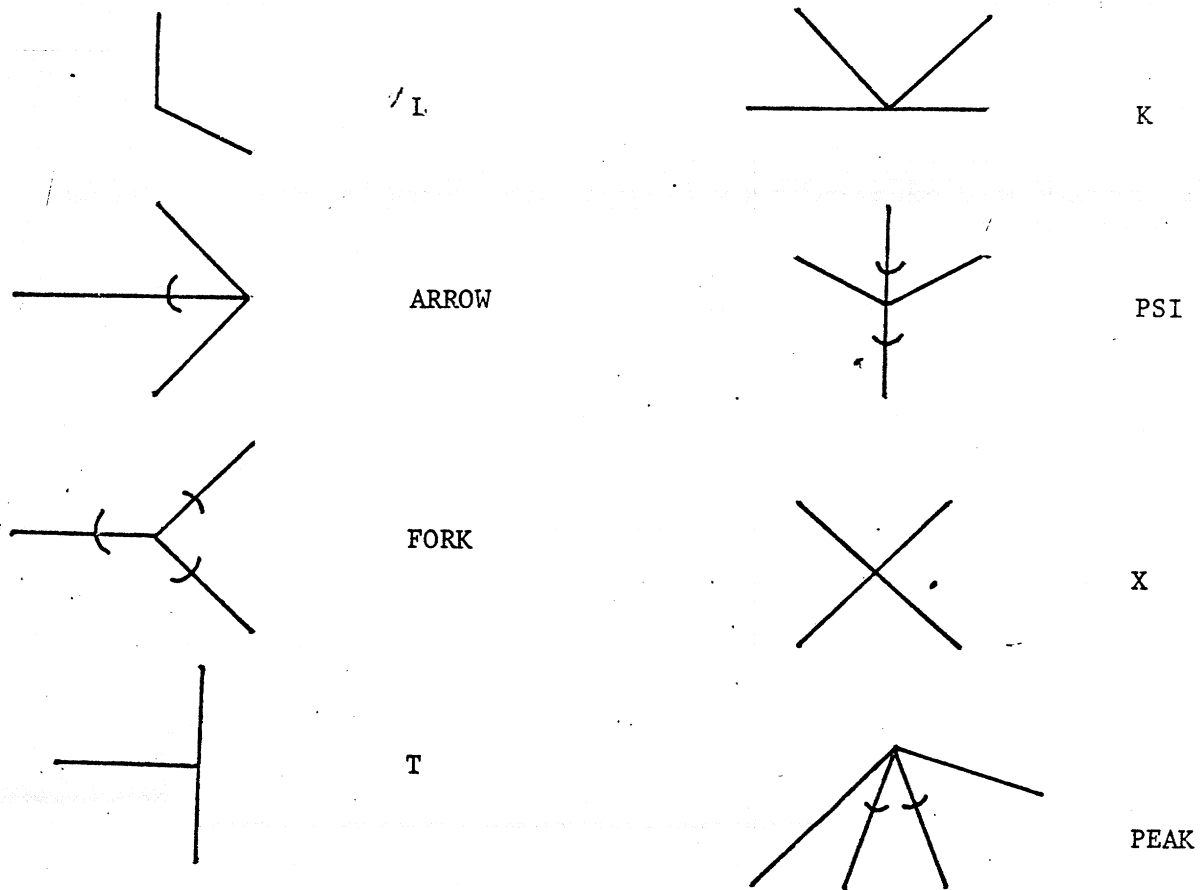
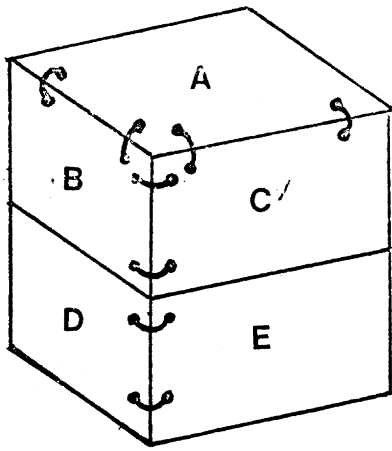
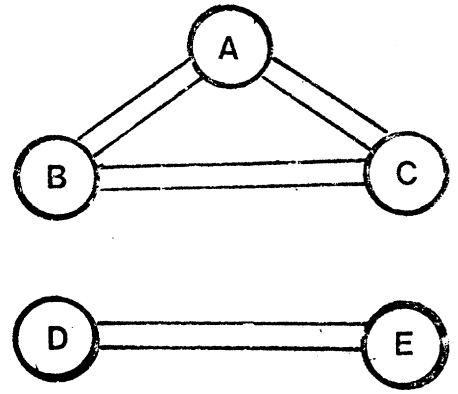


Figure 2
The Guzman links for various vertex types.



(a)



(b)

Figure 3
The links formed by the vertices of a simple scene.

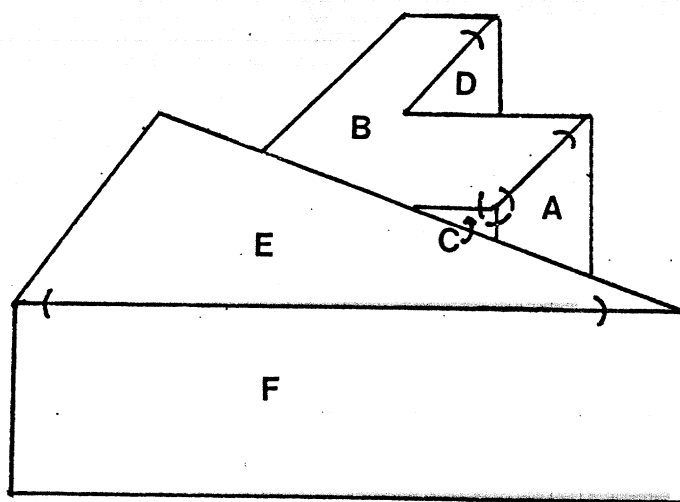


Figure 4
Various linking algorithms cause this to be seen as two, three,
or four bodies.

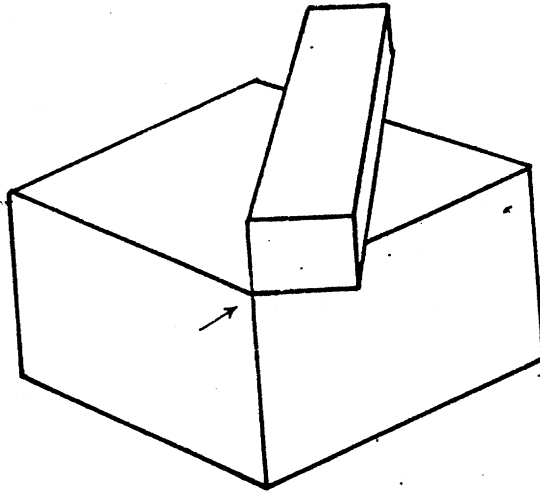


Figure 5
A coincidence causes placement of an incorrect link.

placed by the spurious psi vertex in figure 5, an error occurs in the direction of too much conglomeration.

The problem is corrected in theory two. Theory two differs from theory one because it requires two links for binding rather than just one. By insisting on more evidence, local evidence anomalies are diluted in their potential to damage the end result. Such a method works fine for figure 5, but as a general solution the two link scheme also falters, now on the side of stinginess. In figure 4, partitioning by this second theory yields (A B) (C) (D) (E F).

This stinginess can also be fixed. The first step is to refine theory two into theory three by iterating the amalgamation procedure. The idea is to think of previously joined together region groups as subject themselves to conglomeration in the same way regions are joined. After one pass over the links of figure 4, we have A and B joined together. But the combination is linked to C by two links, causing C to be sucked in on a second run through the linking loop. Theory three then produces (A B C) (D) (E F) as its opinion.

Theory four supplements three by adding a simple special-case heuristic. If a region has only a single link to another region, they are combined. This brings figure 4 around to (A B C D) (E F) as the result, without re-introducing the generosity problem that came up in figure 5 when using theory one. That scene is now also correctly separated into bodies.

Only one more refinement is necessary to complete this sequence of imagined theories and bring us close to Guzman's final program. The required addition is motivated by the scenes like that of figure 6. There we have again too much linking as a result of the indicated fork vertex. Although not really wrong, the one object answer seems less likely to humans than a report of two objects. Guzman overcame this sort of problem toward the end of his thesis work not by augmenting still further the evidence weighing but rather by refining the way evidence is originally generated. The basic change is that all placement of links is subject to inhibition by contrary evidence from adjacent vertices. In particular, no link is placed across a line if its other end is the barb of an arrow, a leg of an L, or a part of the crossbar of a T. This is enough to correctly handle the problem of figure 6. Adding this link inhibition idea gives us Guzman's program in its final form. In the first pass the program gathers evidence through the vertex inspired links that are not inhibited by adjacent vertices. In the second pass, these links cause binding together whenever two regions or sets of previously bound regions are connected by two or more links. It is a somewhat complex but reasonably talented program which usually returns the most likely partition of a scene into bodies.

But does this program of Guzman's constitute a theory? If we use an informal definition which associates the idea of useful theory with the idea of description, then certainly

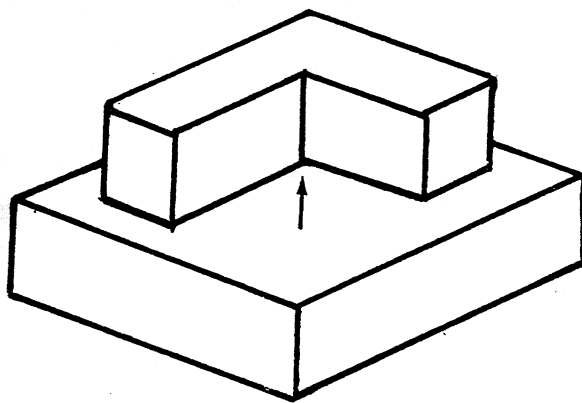


Figure 6
The fork vertex causes the two bodies to be linked together
unless the offending links are inhibited by the adjacent arrows.

Guzman's work is a theory of the region parsing aspect of vision, either as described here or manifested in Guzman's actual machine program. I must hasten to say, however, that it stands incomplete on some of the dimensions along which the worth of a theory can be measured. Guzman's program was insightful and decisive to future developments, but as he left it, the theory had little of the deep semantic roots that a good theory should have.

Let us ask some questions to better understand why the program works instead of just how it works. When does it do well? Why? When does it stumble? How can it be improved?

Experimentation with the program confirms that it works best on scenes composed of objects lacking holes (Winston 1971) and having trihedral vertices. (A vertex is trihedral when exactly three faces of the object meet in three-dimensional space at that vertex.)

Why should this be the case? The answer is simply that trihedral vertices most often project into a line drawing as L's, which we ignore, and arrows and forks, which create links. The program succeeds whenever the weak reverse implication that arrows and forks come from trihedral vertices happens to be correct. Using the psi vertex amounts to a corollary which is necessary because we humans often stack things up and bury an arrow-fork pair in the resulting alignment. From this point of view, the Guzman program becomes a one-heuristic theory in which

a link is created whenever a picture vertex may have come from a trihedral space vertex.

But when does the heuristic fail? Again experiments provide something of an answer. The trihedral vertex heuristic most often fails when alignment creates perjurious arrows. Without some sort of link inhibition mechanism, it is easy to construct examples littered with bad arrows. To combat poor evidence, two possibilities must be explored. One is to demand more evidence, and the other is to find better evidence. The complexity and much of the arbitrary quality of Guzman's work results from electing to use more evidence. But using more evidence was not enough. Guzman was still forced to improve the evidence via the link inhibition heuristic.

The startling fact discovered by Eugene Freuder is that link inhibition is enough! With some slight extensions to the Guzman inhibition heuristics (Rattner 1970), complicated evidence weighing is unnecessary. A program that binds with one link does about as well as more involved ones. By going into the semantic justification for the generation of links, we have a better understanding of the body linking problem and we have a better, more adequate program to replace the original one. This was a serious step in the right direction.

Shadows

Continuing to trace the development of MIT's scene understanding programs, the next topic is a sortie into the

question of handling shadows. The first work at MIT on the subject was done by Orban (Orban 1970). His purpose was to eliminate or erase shadows from a drawing. The approach was quite Guzman-like in flavor as Orban worked empirically with vertices, trying to learn their language and discover heuristic clues that would help establish shadow hypotheses. He found that quite complex scenes could be handled through the following simple facts: 1) a shadow boundary often displays two or more L type vertices in a row 2) shadow boundaries tend to form psi type vertices when they intersect a straight line and 3) shadows may often be found by way of the L's and followed through psi's.

Orban's program is objectionable in the same way Guzman's is. Namely, it is largely empirical and lacking in firm semantic roots. The ideas work in some complex scenes only to fail in others. Particularly troublesome is the common situation where short shadow boundaries involve no L type vertices.

After Orban's program, the shadow problem remained at pasture for some time. The issue was avoided by placing the light source near the eye, thus eliminating the problem by eliminating the shadows. Aside from being disgusting aesthetically, this is a poor solution because shadows should be a positive help rather than a hindrance to be erased out and forgotten.

Interest in shadows was reawakened in conjunction with a desire to use more knowledge of the three-dimensional world in

scene analysis. Among the obvious facts are the following:

- 1) The world of blocks and wedges has a preponderance of vertical lines. Given that a scene has a single distant light source, these vertical lines all cast shadows at the same angle on the retina. Hence when one line is identified as a shadow, it renders all other lines at the same angle suspect.
- 2) Vertical lines cast vertical shadows on vertical faces.
- 3) Horizontal lines cast shadows on horizontal faces that are parallel to the shadow casting edges.
- 4) If a shadow line emerges from a vertex, that vertex almost certainly touches the shadow bearing surface.

With these facts, it is easy to think about a program that would crawl through the scene of figure 7, associating shadow boundaries with their parent edges as shown. One could even implement something, through point four, that would allow the system to know that the cube in figure 7 is lying on the table rather than floating above it. Such a set of programs would be on the same level as Freuder's refinement of Guzman's program with respect to semantic flavor. We were in fact on the verge of implementing such a program when Waltz radicalized our understanding of both the shadow work and the old body-finding problem.

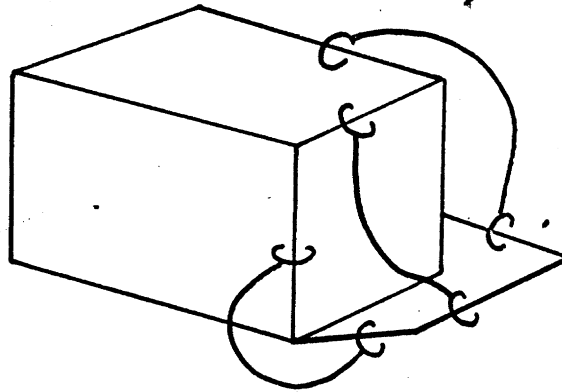


Figure 7
Simple heuristics allow shadow lines to be associated with the edges causing them.

Waltz and Semiotic Interpretation

This section deals with the enormously successful work of Waltz (Waltz 1972a) (Waltz 1972b). Readers familiar with either the work of Huffman (Huffman 1971) or that of Clowes (Clowes 1971) will instantly recognize that their work is the considerable foundation on which Waltz's theory rests.

A line in a drawing appears because of one or another of several possibilities in the physical structure: The line may be a shadow, it may be a crack between two aligned objects, it may be the seam between two surfaces we see, or it may be the boundary between an object and whatever is in back of it.

It is easy enough to label all the lines in a drawing according to their particular cause in the physical world. The drawing in figure 8, for example, shows the Huffman labels for a cube lying flat on the table. The plus labels represent seams where the observer sees both surfaces and stands on the convex side of the surfaces with the inside of the object lying on the concave. The minus labels indicate the observer is on the concave side. And the arrowed lines indicate a boundary where the observer sees only one of the surfaces that form the physical edge.

A curious and amazing thing about such labeled line drawings is that only a few of the combinatorially possible arrangements of labels around a vertex are physically possible. We will never see a T type vertex with both wings labeled plus no

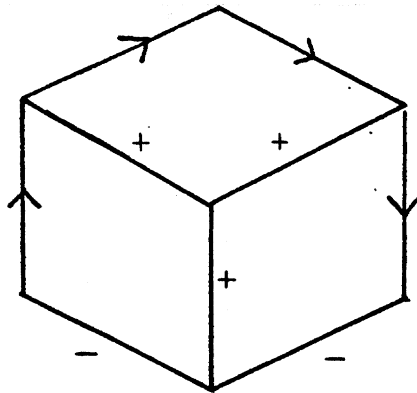


Figure 8
Huffman labels for a cube. Plus implies a convex edge, minus implies concave, and an arrow implies only one of the edge-forming surfaces is visible.

matter how many legal line drawings we examine. (It is presumed that the objects are built of trihedral vertices and that the viewpoint is such that certain types of coincidental alignment in the picture domain are lacking.) Indeed it is easy to prove that an enumeration of all possibilities allowed by three-dimensional constraints includes only six possible L vertex labelings and three each of the fork and arrow types. These are shown in figure 9.

Given the constraints the world places or the arrangements of line labels around a vertex, one can go the other way. Instead of using knowledge of the real physical structure to assign semantic labels, one can use the known constraints on how a drawing can possibly be labeled to get at an understanding of what the physical structure must be like.

The vertices of a line drawing are like the pieces of a jigsaw puzzle in that both are limited as to how they can fit together. Selections for adjacent vertex labelings simply cannot require different labels for the line between them. Given this fact a simple search scheme can work through a drawing, assigning labels to vertices as it goes, taking care that no vertex labeling is assigned that is incompatible with a previous selection at an adjacent vertex. If the search fails without finding a compatible set of labels, then the drawing cannot represent a real structure. If it does find a set of labels, then the successful set or sets of labels yield much information

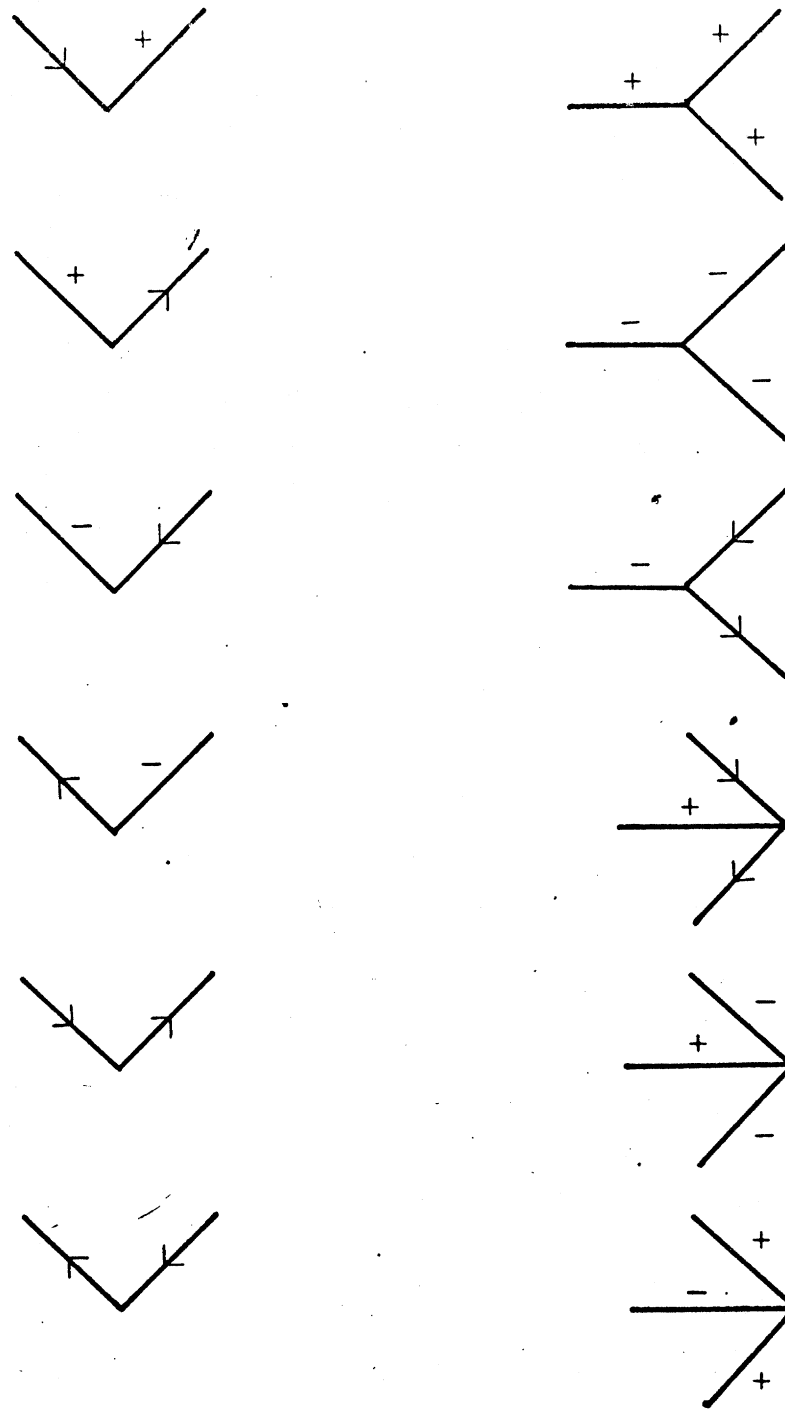


Figure 9
Physically possible configurations of lines around vertices.

about the structure.

Waltz generalized the basic ideas in two fundamental ways. First he expanded the set of line labels such that each includes much more information about the physical situation. Second, he devised a filtering procedure that converges on the possible interpretations with lightning speed relative to a more obvious depth-first search strategy.

Waltz's labels carry information both about the cause of the line and about the illumination on the two adjacent regions. Figure 10 gives Waltz's eleven allowed line interpretations. The set includes shadows and cracks. The regions beside the line are considered to be either illuminated, shadowed by facing away from the light, or shadowed by another object. These possibilities suggest that the set of legal labels would include $11 \times 3 \times 3 = 99$ entries, but a few simple facts immediately eliminate about half of these. A concave edge may not, for example, have one constituent surface illuminated and the other shadowed.

With this set of labels, body finding is easy! The line labels with arrows as part of their symbol (two, three, four, five, nine, ten, and eleven) indicate places where one body obscures another body or the table. Once Waltz's program finds a compatible set of line labels for a drawing, each body is surrounded by line labels from the arrow class.

To create his program, Waltz first worked out what vertex configurations are possible with his set of line labels. Figure

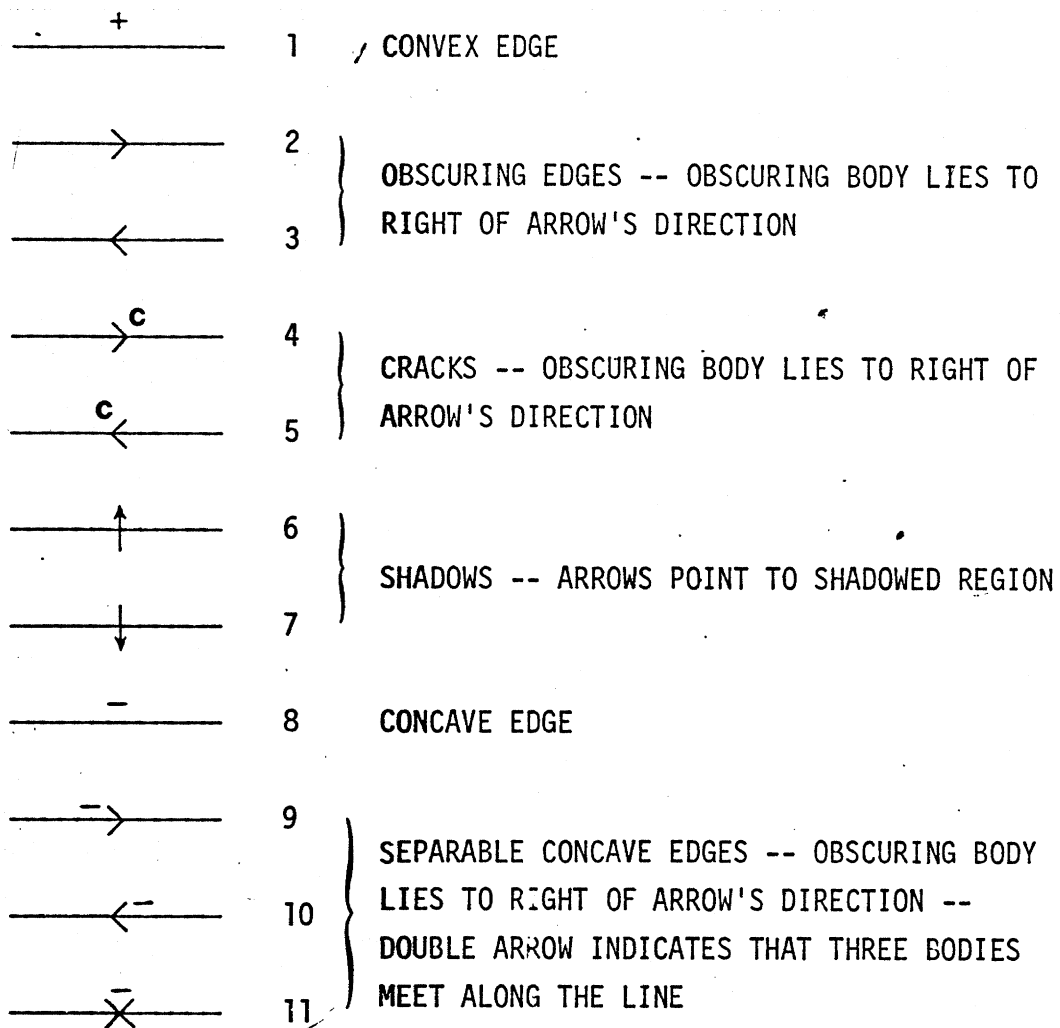


Figure 10
Line interpretations recognized by Waltz's program.


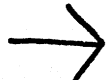








	APPROXIMATE NUMBER OF COMBINATORIALLY POSSIBLE LABELINGS	APPROXIMATE NUMBER OF PHYSICALLY POSSIBLE LABELINGS
	2,500	80
	125,000	70
	125,000	500
	125,000	500
	6×10^6	10
	6×10^6	300
	6×10^6	100
	6×10^6	100
	6×10^6	100
	3×10^8	30

Figure 11
 Only a few of the combinatorially possible labelings are physically possible.

11 gives the result. Happily the possible vertex labelings constitute only a tiny fraction of the ways labels can be arrayed around a vertex. The number of possible vertices is large but not unmanageably so.

Increasing the number of legal vertex labelings does not increase the number of interpretations of typical line drawings. This is because a proper increase in descriptive detail strongly constrains the way things may go together. Again the analogy with jigsaw puzzles gives an idea of what is happening: The shapes of pieces constrain how they may fit together, but the colors give still more constraint by adding another dimension of comparison.

Interestingly, the number of ways to label a fork is much larger than the number for an arrow. A single arrow consequently offers more constraint and less ambiguity than does a fork. This explains why experiments with Guzman's program showed arrows to be more reliable than forks as sources of good links.

Figure 12 shows a fairly complex scene. But with little effort, Waltz's program can sort out the shadow lines and find the correct number of bodies.

What I have discussed of this theory so far is but an *hors d'oeuvre*. Waltz's forthcoming doctoral dissertation has much to say about handling coincidental alignment, finding the approximate orientation of surfaces, and dealing with higher order object relations like support (Waltz 1972b). But without

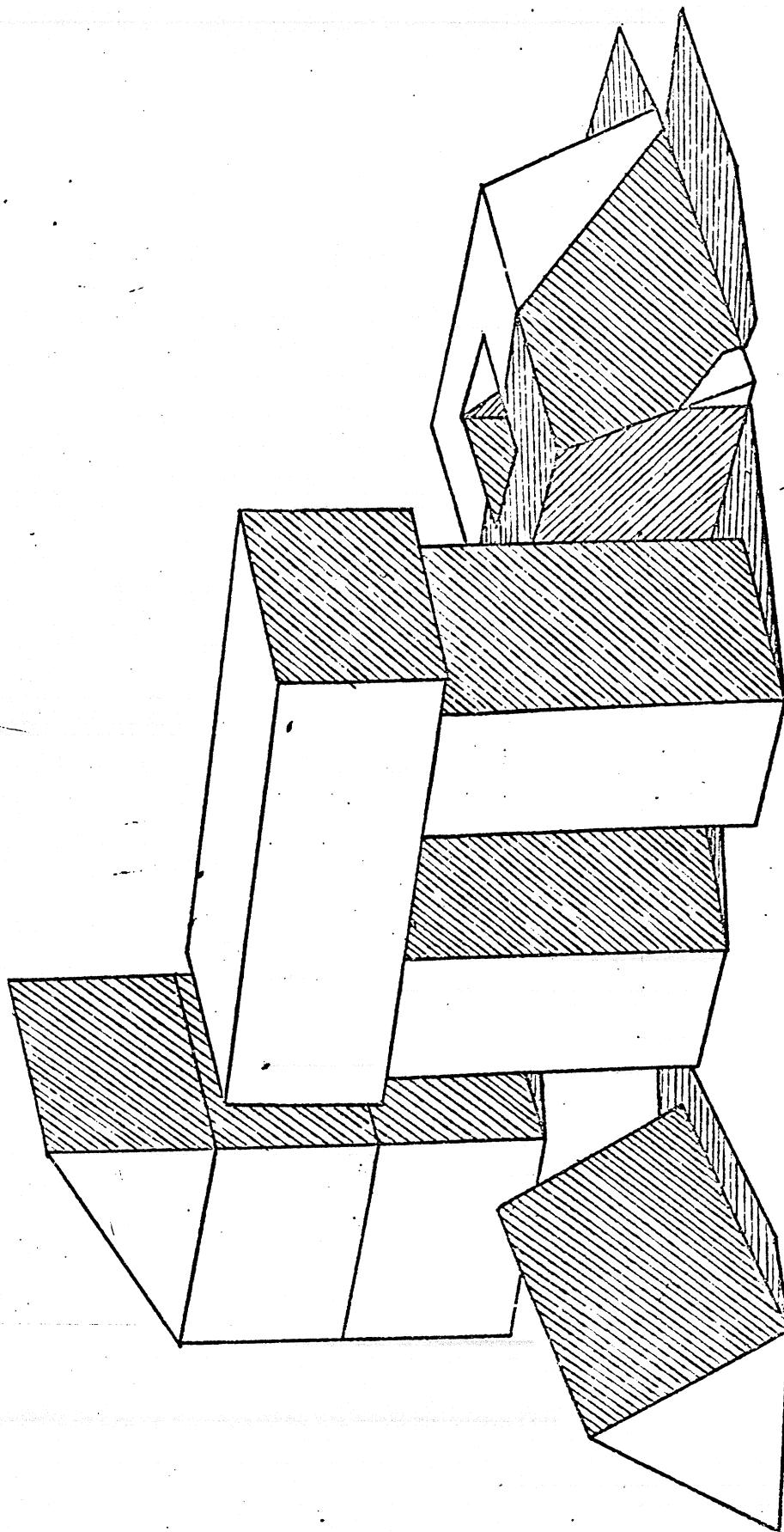


Figure 12
Waltz's program easily handles complicated scenes.

getting into those exciting results, I can comment on how his work fits together with previous ideas on body finding and on shadows.

First of all Waltz's program has a syntactic flavor. The program has a table of possible vertices and on some level can be thought to parse the scene. But it is essential to understand that this is a program with substantive semantic roots. The table is not an amalgam of the purely ad hoc and empirical. It is derived directly from arguments about how real structures can project onto a two dimensional drawing. The resulting label set, together with the program that uses it, can be thought of quite well as a compiled form of those arguments whereby facts about three-dimensional space become constraints on lines and vertices.

In retrospect, I see Waltz's work as the culmination of a long effort beginning with Guzman and moving through the work of Orban, Ratner, Winston, Huffman and Clowes. Each person built on earlier ideas and experiments, producing either a refinement, a reaction, or an explanation. The net result is a tradition moving toward more and better ability to describe and toward more and better theoretical justification behind working programs.

SYSTEM ISSUES

Heterarchy

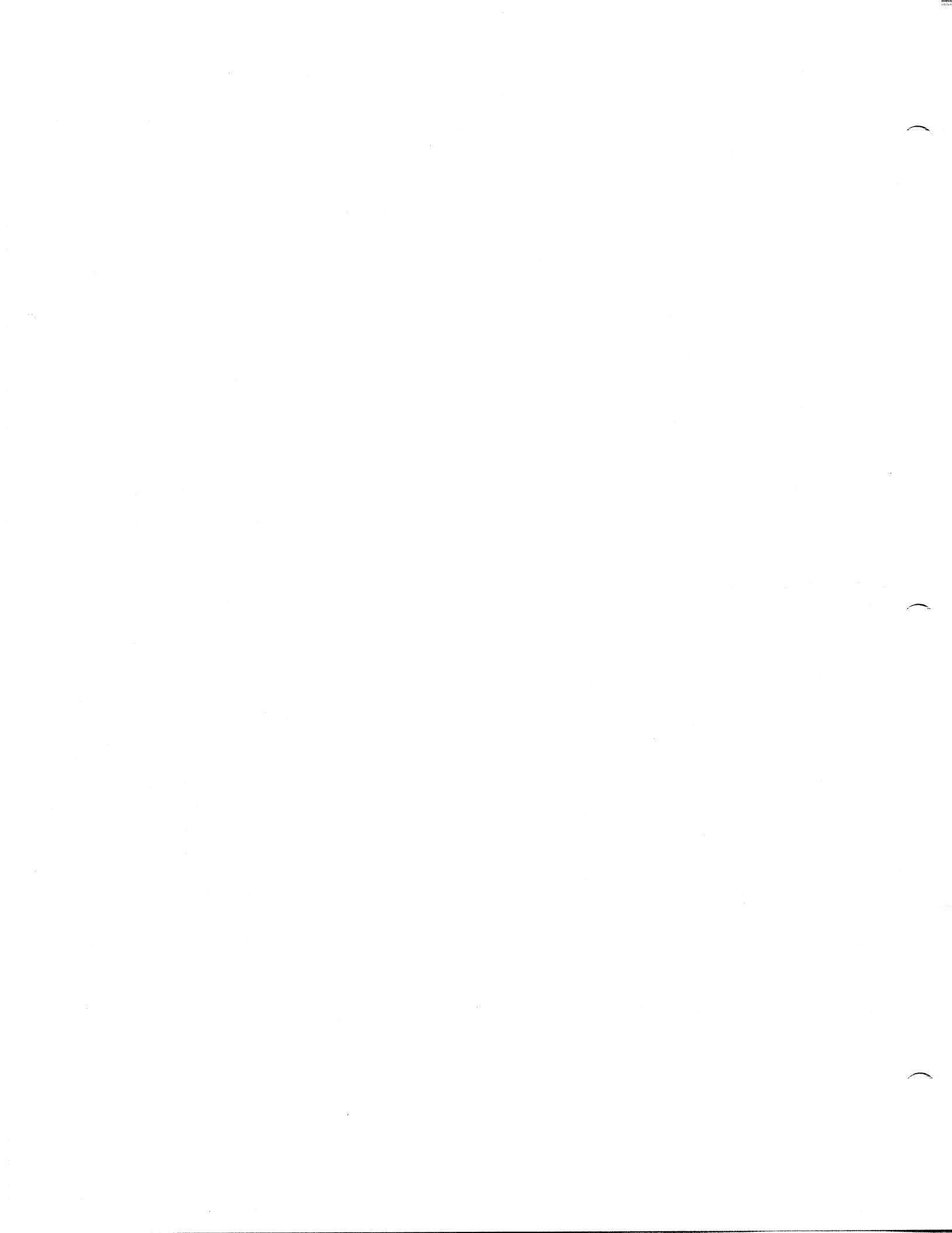
Waltz's work is part of understanding how line drawings convey information about scenes. This section discusses some of our newer ideas about how to get such understanding into a working system.

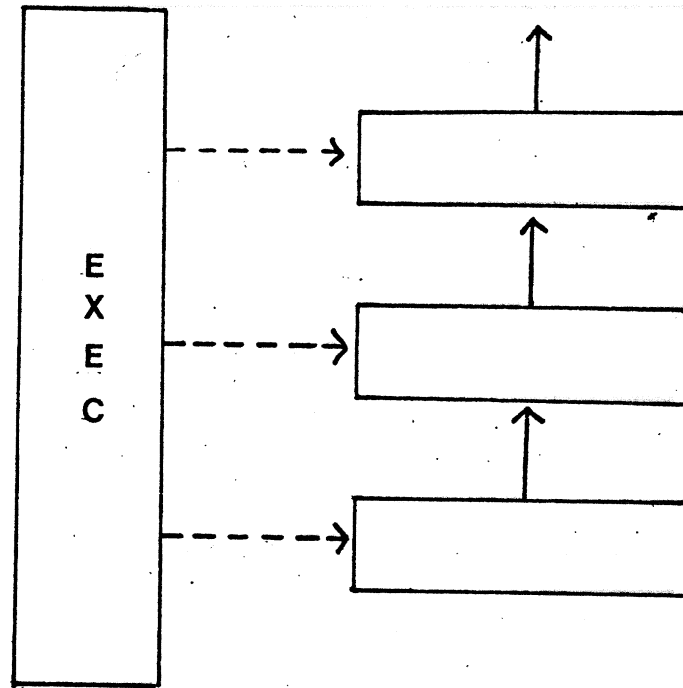
At MIT the first success in copying a simple block structure from spare parts involved using a pass-oriented structure like that illustrated in figure 13. The solid lines represent data flow and the dashed lines, control. The executive in this approach is a very simple sequence of subroutine calls, mostly partitioned into one module. The calling up of the action modules is fixed in advance and the order is indifferent to the peculiarities of the scene. Each action module is charged with augmenting the data it receives according to its labeled specialty.

This kind of organization does not work well. Its purpose is to make a vehicle quickly for testing the modules then available. It is often better to have one system working before expending too much effort in arguing about which system is best.

From this base we have moved toward another style of organization which has come to be called heterarchical (Minsky and Papert 1972). The concept lacks precise definition, but the following are some of the characteristics that we aim for:

1. A complex system should be goal oriented.





class oriented system metaphor.

Procedures at all levels should be short and associated with some definite goal. Goals should normally be satisfied by invoking a small number of subgoals for other procedures or by directly calling a few primitives. A corollary is that the system should be top down. For the most part nothing should be done unless necessary to accomplish something at a higher level.

2. The executive control should be distributed throughout the system. In a heterarchical system, the modules interact not like a master and slaves but more like a community of experts.

3. Programmers should make as few assumptions as possible about the state in which the system will be when a procedure is called. The procedure itself should contain the necessary machinery to set up whatever conditions are required before it can do its job. This is obviously of prime importance when many authors contribute to the system, for they should be able to add knowledge via new code without completely understanding the rest of the system. In practice this usually works out as a list of goals lying like a preamble near the beginning of a routine. Typically

these goals are satisfied by simple reference to the data base, but if not, notes are left as to where help may be found, in the PLANNER (Hewitt 1972) or CONNIVER style (McDermott and Sussman 1972).

4. The system should contain some knowledge of itself. It is not enough to think of executives and primitives. There should be modules that act as critics and complain when something looks suspicious. Others must know how and when the primitives are likely to fail. Communication among these modules should be more colorful than mere flow of data and command. It should include what in human discourse would be called advice, suggestions, remarks, complaints, criticism, questions, answers, lies, and conjectures.

5. A system should have facilities for tentative conclusions. The system will detect mistakes as it goes. A conjectured configuration may be found to be unstable or the hand may be led to grasp air. When this happens, we need to know what facts in the data base are most problematical; we need to know how to try to fix things; and we need to know how far-ranging the consequences of a change are likely to be.

Graphically such a system looks more like a network of procedures than an orderly, immutable sequence. Each procedure is connected to others via potential control transfer links. In practice which of these links is used depends on the context in which the various procedures are used, the context being the joint product of the system and the problem undergoing analysis.

Note particularly that this arrangement forces us to refine our concept of higher versus lower level routines. Now programs normally thought to be low level may very well employ other programs considered high level. The terms no longer indicate the order in which a routine occurs in analysis. Instead a vision system procedure is high or low level according to the sort of data it works with. Line finders that work with intensity points are low level but may certainly on occasion call a stability tester that works with relatively high level object models.

Firin and Environment Driven Analysis

Our earliest MIT vision system interacted only narrowly and in a predetermined way with its environment. The pass oriented structure prevents better interaction. But we are now moving toward a different sort of vision system in which the environment controls the analysis. (This idea was prominent in Ernst's very early work (Ernst 1961).)

Readers who find this idea strange should see an exposition of the notion by Simon (Simon 1969). He argues that

much of what passes as intelligent behavior is in point of fact a happy cooperation between unexpectedly simple algorithms and complex environments. He cites the case of an ant wardering along a beach rift with ant sized obstacles. The ant's curvacious path might seem to be an insarely complex ritual to someone locking only at a history of it traced on paper. But in fact the humble ant is merely trying to circumvent the beach's obstacles and go home.

Watching the locus of control of our current system as it struggles with a complicated scene is like watching Simon's ant. The up and down, the around and backing off, the use of this method then another, all seem to be mysterious at first. But like the ant's, the system's complex behavior is the product of simple algorithms coupled together and driven by the demands of the scene. The remainder of this section discusses some elegant procedures implemented by Finin which illustrate two ways in which the environment influences the MIT vision system (Finin 1972).

The vision system contains a specialist whose task is to determine what we call the skeleton of a brick. A skeleton consists of a set of three lines, one lying along each of the three axes (Finin 1972). Each of the lines in a skeleton must be complete and unobscured so that the dimensions of the brick in question may be determined. Figure 14 shows some of the skeletons found in various situations by this module.

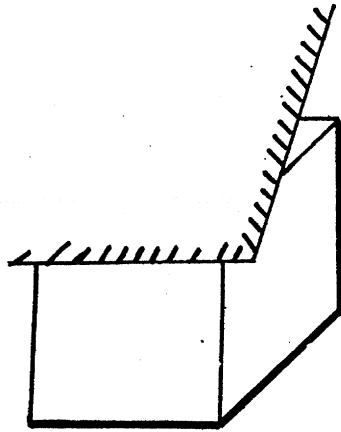
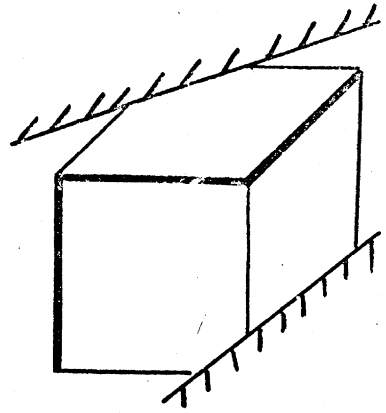
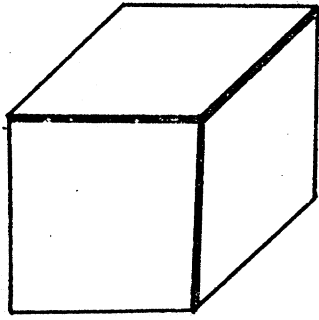


Figure 14
Some skeletons found for bricks.

The only problem with the program lies in the fact that complete skeletons are moderately rare in practice because of heavy obscuring. Even in the simple arch in figure 15a, one object, the left side support, cannot be fully analyzed, lacking as it does a completely exposed line in the depth dimension. But humans have no trouble circumventing this difficulty. Indeed, it generally does not even occur to us that there is a problem because we so naturally assume that the right and left supports have the same dimensions. At this point let us look at the system's internal discourse when working on this scene to better understand how a group - hypothesize - criticize cycle typically works out:

Let me see, what are A's dimensions. First I must identify a skeleton. Cops! We can only get a partial skeleton, two complete lines are there, but only a partial line along the third brick axis. This means I know two dimensions but I have only a lower bound on the third. Let me see if A is part of some group. Oh yes, A and B both support C so they form a group of a sort. Let me therefore hypothesize that A and B are the same and run through my check list to see if there is any reason to doubt that.

Are A and B the same sort of objects?

Yes, Both are bricks.

Are they both oriented the same way?

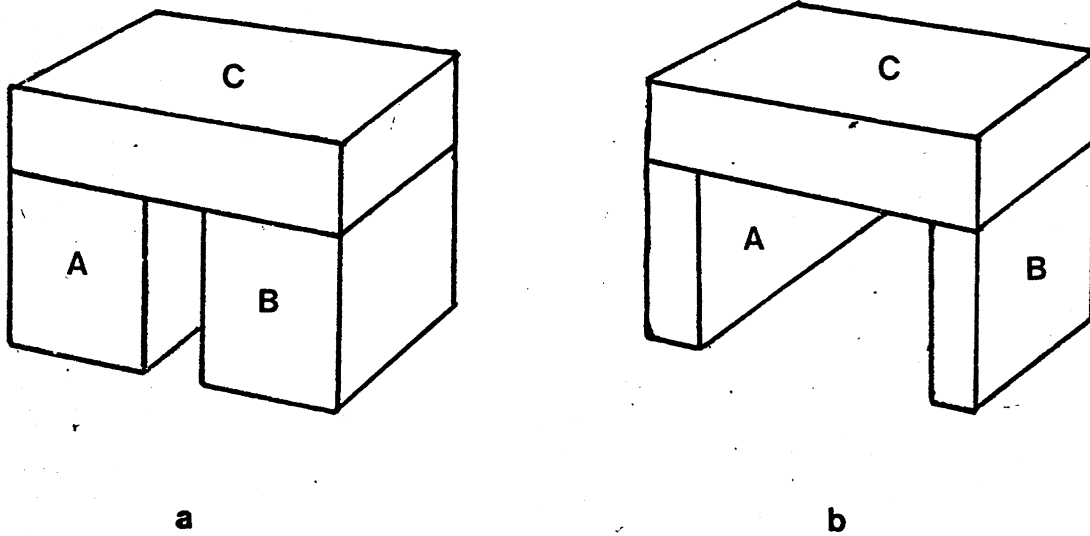


Figure 15
In one case, A's depth is extrapolated from B's. In the other no hypothesis can be confirmed.

Yes, that checks out too.

Well, do the observable dimensions match?

Indeed.

Is there any reason to believe the unobservable dimension of A is different from its analogue on B?

No.

OK. Everything seems all right. I will tentatively accept the hypothesis and proceed.

Through this internal dialogue, the machine succeeds in finding all the necessary dimensions for the obscured support in figure 15a. Figure 15b shows how the conflict search can fail at the very last step.

Grouping amounts, of course, to using a great deal of context in scene analysis. We have discussed how the system uses groups to hypothesize properties for the group's members and we should add that the formation of a group is in itself a matter hypothesis followed by a search for evidence conflicting with the hypothesis. The system now forms group hypotheses from the following configurations, roughly in order of grouping strength:

1. Stacks or rows of objects connected by chains of support or in-front-of relations.
2. Objects that serve the same function such as the sides of an arch or the legs of a table.

3. Objects that are close together.

4. Objects that are of the same type.

To test the validity of these hypotheses, the machine makes tests of good membership on the individual elements. It basically performs conformity tests, throwing out anything too unusual. There is a preliminary theory of how this can be done sensibly (Winston 1971). The basic feature of Winston's theory is that it involves not only a measure of how distant a particular element is from the norm, but also of how much deviation from the norm is typical and thus acceptable.

Note that this hypothesis-rooted theory is much different from Gestaltist notions of good groups emerging magically from the set of all possible groups. Critics of artificial intelligence correctly point out the computational implausibility of considering all possible groups but somehow fail to see the alternative of using clues to hypothesize a limited number of good candidate groups.

Naturally all of these group - hypothesize - criticize efforts are less likely to work out than are programs which operate through direct observation. It is therefore good to leave data base notes relating facts both to their degree of certainty and to the programs that found them. Thus an assertion that says a particular brick has such and such a size may well have other assertions describing it as only probable, conjectured from the dimensions of a related brick, and owing the discovered

relationship to a particular grouping program. Using such knowledge is as yet only planned, but in preparation we try to refrain from using more than one method in a single program. This makes it easy to describe how a particular assertion was made by simply noting the name of the program that made it.

Visual observation of movement provides another way the environment can influence and control what a vision system thinks about. One of the first successful projects was executed at Stanford (Wickman 1967). The purpose was to align two bricks, one atop the other. The method essentially required the complete construction of a line drawing with subsequent determination of relative position. The Japanese have used a similar approach in placing a block inside a box.

The MIT entry into this area is a little different. We do not require complete recomputation of a scene, as did the Stanford system. The problem is to check the position of a just placed object to be sure it lies within some tolerance of the assigned place for it. (In our arm errors in placement may occasionally be on the order of 1/2".)

Rather than recompute a line drawing of the scene to find the object's coordinates, we use our model of where the object should be to direct the eye to selected key regions. In brief, what happens is as follows:

1. The three-dimensional coordinates for selected vertices are determined for the object whose position

is to be checked.

2. Then the supposed locations of those vertices on the eye's retina are easily computed.

3. A vertex search using circular scans around each of these supposed vertex positions will climb to a set of actual coordinates for the vertices on the retina (Winston and Lerman 1972). Revised three-dimensional coordinates can be determined from these retinal coordinates, given the altitude of the object.

4. Comparing the object's real and supposed coordinates gives a correction which is then effected by a gentle, wrist-dominated arm action.

The vertex-locating program tries to avoid vertices that form alignments with those of other objects already in place. This considerably simplifies the work of the vertex finder. With a bit more work, the program could be made to avoid vertices obscured by the hand, thus allowing performance of the feedback operation more dynamically, without withdrawing the hand.

LEARNING TO IDENTIFY TOY BLOCK STRUCTURES

Learning

This section describes a working computer program which embodies a new theory of learning (Winston 1970). I believe it is unlike previous theories because its basic idea is to understand how concepts can be learned from a few judiciously selected examples. The sequence in Figure 16, for example, generates in the machine an idea of the arch sufficient to handle correctly all the configurations in figure 17 in spite of severe rotations, size changes, proportion changes and changes in viewing angle.

Although no previous theory in the artificial intelligence, psychology, or other literatures can completely account for anything like this competence, the basic ideas are quite simple:

1. If you want to teach a concept, you must first be sure your student, man or machine, can build descriptions adequate to represent that concept.
2. If you want to teach a concept, you should use samples which are a kind of non-example.

The first point on description should be clear. At some level we must have an adequate set of primitive concepts and relations out of which we can assemble interesting concepts at the next higher level which in turn become the primitives for concepts at a still higher level. The operation of the learning

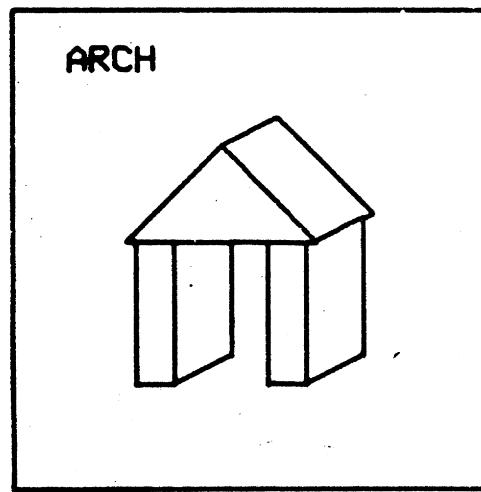
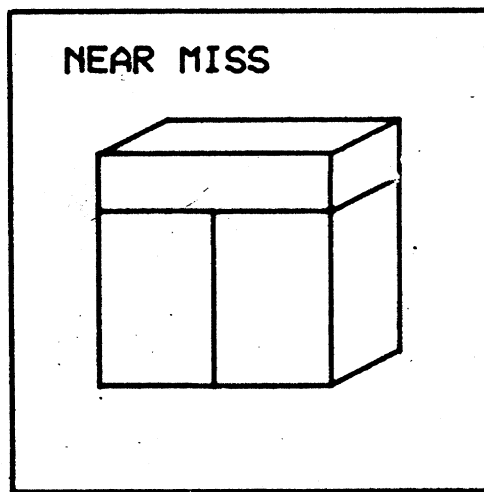
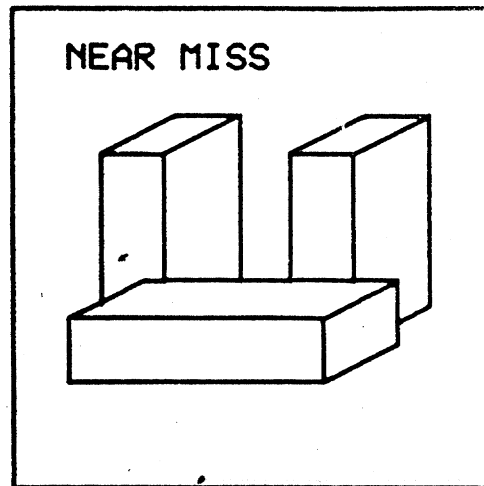
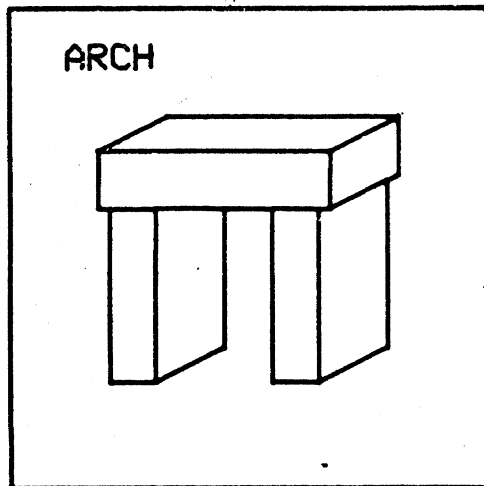


Figure 16
An arch training sequence.

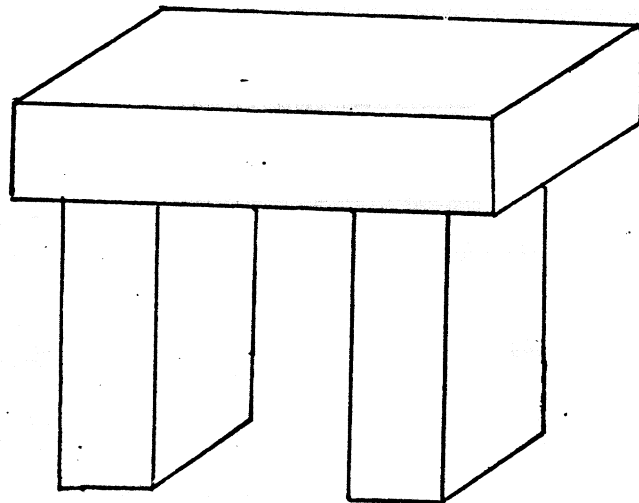
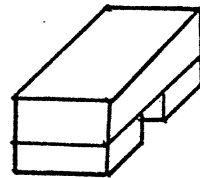
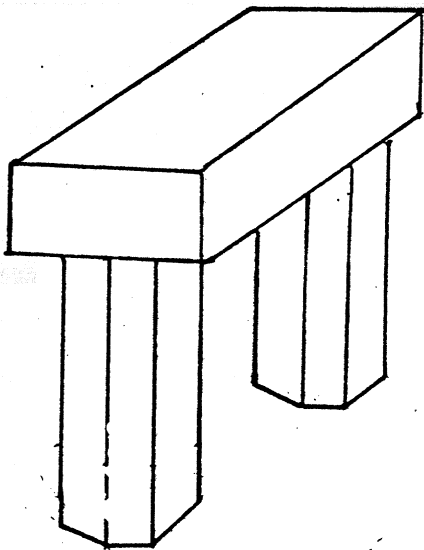
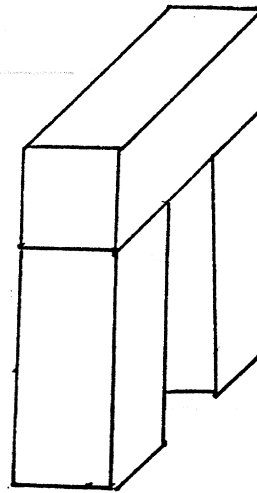
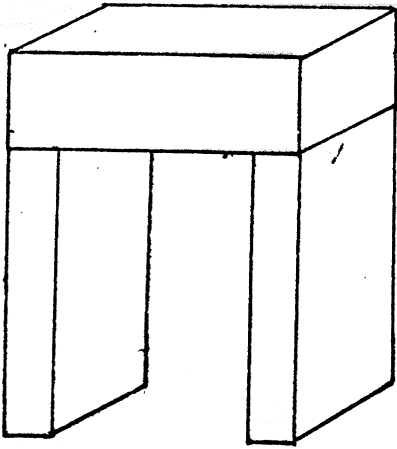


Figure 17
Structures recognized as arches.

program depends completely on the power of the analysis programs described in the previous sections.

But what is meant by the second claim that one must show the machine not just examples of concepts but something else? First of all, something else means something which is close to being an example but fails to be admissible by way of one or a few crucial deficiencies. I call these samples near-misses. My view is that they are more important to learning than examples and they provide just the right information to teach the machine directly, via a few samples, rather than laboriously and uncertainly through many samples in some kind of reinforcement mode.

The purpose of this learning process is to create in the machine whatever is needed to identify instances of learned concepts. This leads directly to the notion of a model. To be precise, I use the term as follows:

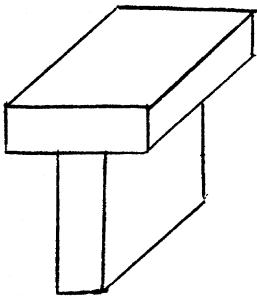
A model is a proper description augmented by information about which elements of the description are essential and by information about what, if anything, must not be present in examples of the concept.

The description must be a proper description because the descriptive language — the possible relations — must naturally be appropriate to the definitions expected. For this reason one cannot build a model on top of a data base that describes the scene in terms of only vertex coordinates, for such a description

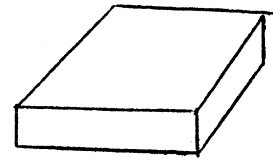
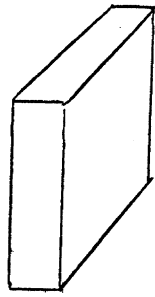
is on too low a level. Nor can one build a model on top of a higher level description that contains only color information, for example, because that information is usually irrelevant to the concept in question.

The key part of the definition of model is the idea that some elements of the description must be underlined as particularly important. Figure 18 shows a training sequence that conveys the idea of the pedestal. The first step is to show the machine a sample of the concept to be learned. From a line drawing, the scene analysis routines produce a hierarchical symbolic description which carries the same sort of information about a scene that a human uses and understands. Blocks are described as bricks or wedges, as standing or lying, and as related to others by relations like in-front-of or supports.

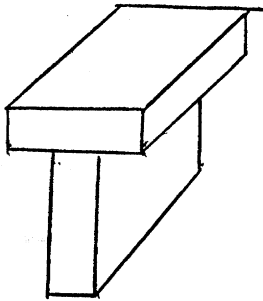
This description resides in the data base in the form of list structures, but I present it here as a network of nodes and pointers, the nodes representing objects and the pointers representing relations between them. See figure 19 where a pedestal network is shown. In this case, there are relatively few things in the net: just a node representing the scene as a whole and two more for the objects. These are related to each other by the supported-by pointer and to the general knowledge of the net via pointers like is-a, denoting set membership, and has-posture, which leads in one case to standing and in the other to lying.



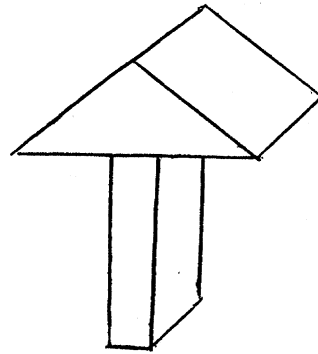
PEDESTAL



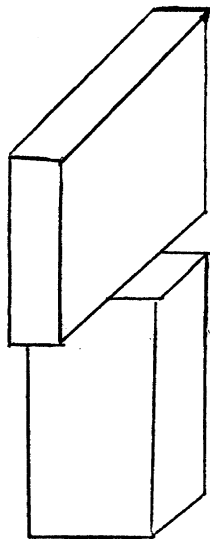
NEAR MISS



NEAR MISS



NEAR MISS



NEAR MISS

Figure 18
A pedestal training sequence.

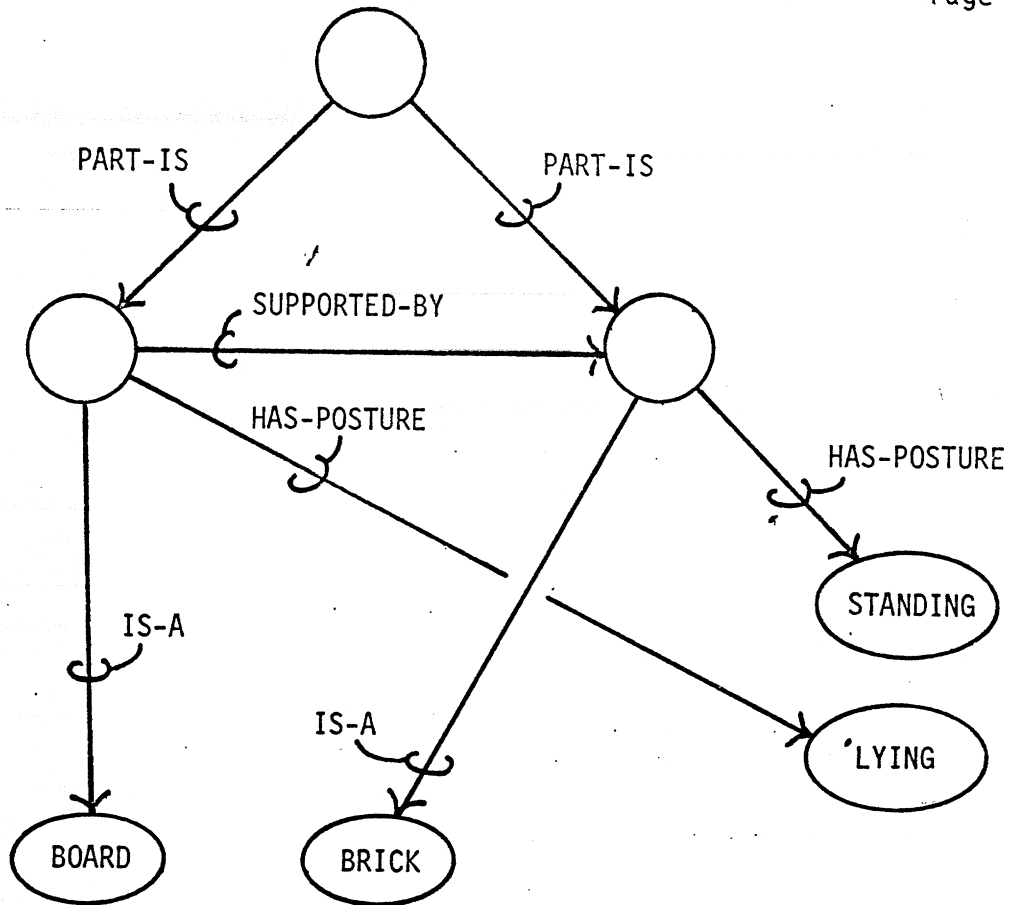


Figure 19
A pedestal description.

Now in the pedestal, the support relation is essential — there is no pedestal without it. Similarly the posture and identity of the board and brick must be correct. Therefore, the objective in a teaching sequence is to somehow convey to the machine the essential, emphatic quality of those features (Later on we will see further examples where some relations become less essential and others are forbidden).

Returning to figure 18 note that the second sample is a near-miss in which nothing has changed except that the board no longer rests on the standing brick. This is reflected in the description by the absence of a supported-by pointer. It is a simple matter for a description comparison program to detect this missing relation as the only difference between this description and the original one which was an admissible instance. The machine can only conclude, as we would, that the loss of this relation explains why the near-miss fails to qualify as a pedestal. This being the case, the proper action is clear. The machine makes a note that the supported-by relation is essential by replacing the original pointer with must-be-supported-by. Again note that this point is conveyed directly by a single drawing, not by a statistical inference from a boring hoard of trials. Note further that this information is quite high level. It will be discerned in scenes as long as the descriptive routines have the power to analyze that scene. Thus we need not be as concerned about the simple changes that incapacitate older,

lower level learning ideas. Rotations, size dilations and the like are easily handled, given the descriptive power we have in operating programs.

Continuing now with our example, the teacher proceeds to basically strengthen the other relations according to whatever prejudices he has. In this sequence the teacher has chosen to reinforce the pointers which determine that the support is standing and the pointers which similarly determine that the supported object is a lying board. Figure 20 shows the model resulting.

Now that the basic idea is clear, the slightly more complex arch sequence will bring out some further points. The first sample, shown back in Figure 16 is an example, as always. From it we generate an initial description as before. The next step is similar to the one taken with the pedestal in that the teacher presents a near-miss with the supported object now removed and resting on the table. But this time not one, but two differences are noticed in the corresponding description networks as now there are two missing supported-by pointers.

This opens up the big question of what is to be done when more than one relationship can explain why the near-miss misses. What is needed, of course, is a theory of how to sort out observed differences so that the most important and most likely to be responsible difference can be hypothesized and reacted to.

The theory itself is somewhat detailed, but it is the

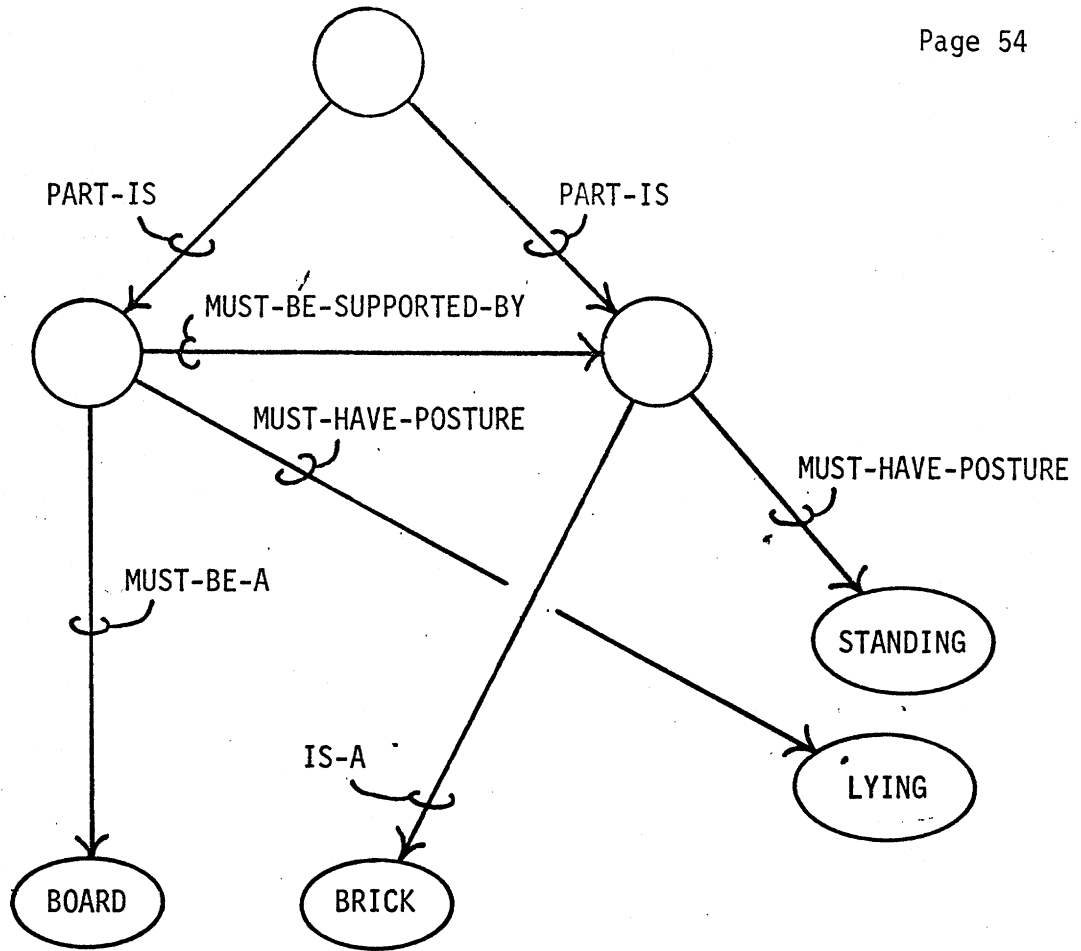


Figure 20
A pedestal model.

exploration of this detail through writing and experimenting with programs that gives the overall theory a crisp substance. Repeated cycles of refinement and testing of a theory, as embodied in a program, is an important part of an emerging artificial intelligence methodology.

Now the results of this approach on the difference ranking module itself include the following points:

First of all, if two differences are observed which are of the same nature and description, then they are assumed to contribute jointly to the failure of the near-miss and both are acted on. This handles the arch case where two support relations were observed to be absent in the near-miss. Since the differences are both of the missing pointer type and since both involve the same supported-by relation, it is deemed heuristically sound to handle them both together as a unit.

Secondly, differences are ranked in order of their distance from the origin of the net. Thus a difference observed in the relationship of two objects is considered more important than a change in the shape of an object's face, which in turn is interpreted as more important than an obscured vertex.

Thirdly, differences at the same level are ranked according to type. In the current implementation, differences of the missing pointer type are ranked ahead of those where a pointer is added in the near-miss. This is reasonable since dropping a pointer to make a near-miss may well force the

introduction of a new pointer. Indeed we have ignored the introduction of a support pointer between the lying brick and the table because the difference resulting from this new pointer is inferior to the difference resulting from the missing pointer. Finally, if two differences are found of the same type on the same level, then some secondary heuristics are used to try to sort them out. Support relations, for example, make more important differences than one expects from touch or left-right pointers.

Now these factors constitute only a theory of hypothesis formation. The theory does make mistakes, especially if the teacher is poor. I will return to this problem after completing the tour through the arch example. Recall that the machine learned the importance of the support relations. In the next step it learns, somewhat indirectly, about the hole. This is conveyed through the near-miss with the two side supports touching. Now the theory of most important differences reports that two new touch pointers are present in the near-miss, symmetrically indicating that the side supports have moved together. Here surely the reasonable conclusion is that the new pointers have fouled the concept. The model is therefore refined to have ~~must-not-touch~~ pointers between the nodes of the side supports. This dissuades identification programs, later described, from ever reporting an arch if such a forbidden relation is in fact present.

It is now clear how crucial information of the negative sort is introduced into models. They can contain not only information about what is essential but also information about what sorts of characteristics prevent a sample from being associated with the modeled concept.

So far I have shown examples of emphatic relations, both of the must-be and must-not-be type as introduced by near-miss samples. The following is an example of the inductive generalization introduced by the sample with the lying brick replaced by a wedge. Whether to call this a kind of arch or report it as a near-miss depends on the taste of the machine's instructor, of course. Let us explore the consequence of introducing it as an example, rather than a near-miss.

In terms of the description network comparator, the machine finds an is-a pointer moved over from brick to wedge. There are, given this observation, a variety of things to do. The simplest is to take the most conservative stance and form a new class, that of the brick or wedge, a kind of superset.

To see what other options are available, look in figure 21 at the descriptions of brick and wedge and the portion of the general knowledge net that relates them together. There various sets are linked together by the a-kind-of relationship. From this diagram we see that our first choice was a conservative point on a spectrum whose other end suggests that we move the is-a pointer over to object, object being the most distant

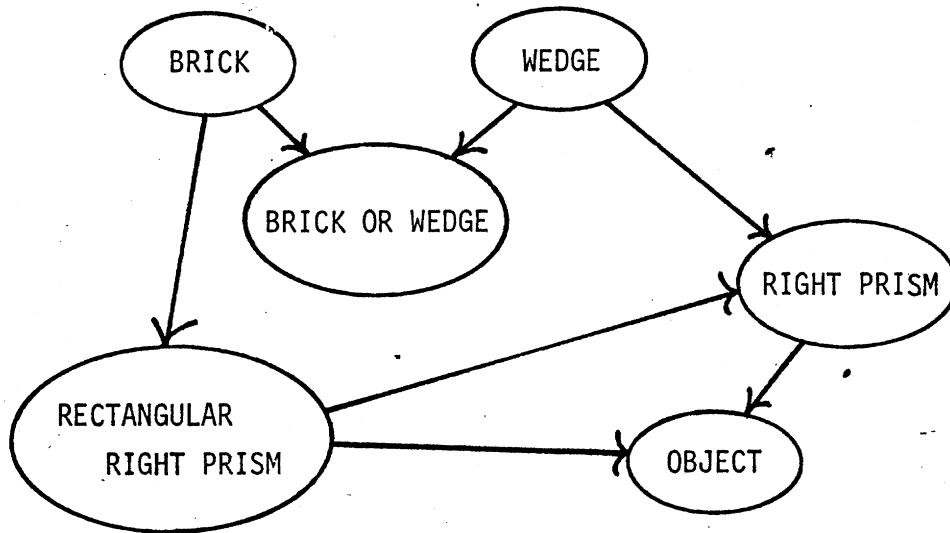


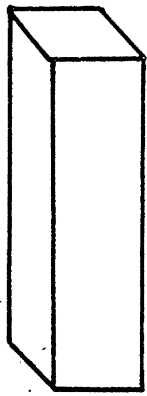
Figure 21
Relations between brick, wedge, and object. All pointers are a-kind-of pointers.

intersection of a-kind-of relations. We choose a conservative position and fix the is-a pointer to the closest observed intersection, in this case right-prism.

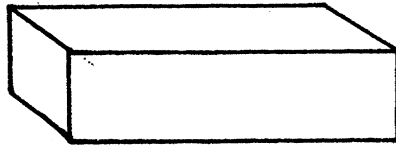
Again a hypothesis has to be made, and the hypothesis may well be wrong. In this case it is a question of difference interpretation rather than the question of sorting out the correct difference from many, but the effect is the same. There simply must be mechanisms for detecting errors and correcting them.

Errors are detected when an example refutes a previously made assumption. If the first scene of Figure 22 is reported as an example of concept X while the second is given as a near-miss, the natural interpretation is that an X must be standing. But an alternate interpretation, considered secondary by the ranking program, is that an X must not be lying. If a shrewd teacher wishes to force the secondary interpretation, he need only give the tilted brick as an example, for it has no standing pointer and thus is a contradiction to the primary hypothesis. Under these conditions, the system is prepared to back up and try an alternative. As the alternative may also lead to trouble, the process of backup may iterate as a pure depth first search. One could do better by devising a little theory that would back up more intelligently to the decision most likely to have caused the error.

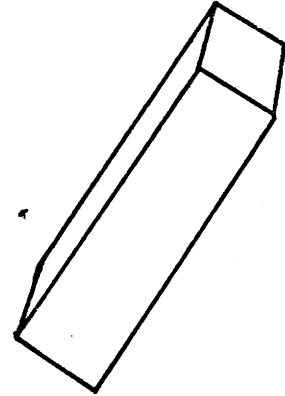
I mentioned just now the role of a shrewd teacher. I



AN X



NEAR MISS



AN X

Figure 22
A training sequence that leads to backup.

regard the dependence on a teacher as a feature of this theory. Too often in the past history of machine learning theory the use of a teacher was considered cheating and mechanisms were instead expected to self-organize their way to understanding by way of evolutionary trial and error, or reinforcement, or whatever. This ignores the very real fact that humans as well as machines learn very little without good teaching. The first attempt should be to understand the kind of learning that is at once the most common and the most useful.

It is clear that the system assimilates new models from the teacher and it is in fact dependent on good teaching, but it depends fundamentally on its own good judgement and previously learned ideas to understand and disentangle what the teacher has in mind. It must itself deduce what are the salient ideas in the training sequence and it must itself decide on an augmentation of the model which captures those ideas. By carefully limiting the teacher to the presentation of a sequence of samples, low level rote learning questions are avoided while allowing study of the issues which underly all sorts of meaningful learning, including interesting forms of direct telling.

Identification

Having developed the theory of learning models, I shall say a little about using them in identification. Since this subject is both tangential to the main thrust and documented elsewhere (Winston 1970), I shall merely give the highlights

here.

To begin with, identification is done in a variety of modes, our system already exhibiting the following three:

1. We may present a scene and ask the system to identify it.
2. We may present a scene with several concepts represented and ask the system to identify all of them.
3. We may ask if a given scene contains an instance of something.

Of course, the first mode of identifying a whole scene is the easiest. We simply insist that 1) all models must-be-type pointers are present in the scene's description and 2) all the models must-not-be-type pointers must not be present. For further refinement, we look at all other differences between the model and scene of other than the emphatic variety and judge the firmness of model identification according to their number and type.

When a scene contains many identifiable rows, stacks, or other groups, we must modify the identification program to allow for the possibility that essential relations may be missing because of obscuring objects. The properties of rows and stacks tend to propagate from the most observable member unless there is contrary evidence.

The last task, that of searching a scene for a particular concept is a wide open question. The method now is to simply

feed our network matching program both the model and the larger network and hope for the best. If some objects are matched against corresponding parts of the model, their pointers to other extraneous objects are forgotten, and the identification routine is applied. Much remains to be done along the lines of guiding the match contextually to the right part of the scene.

COPYING TOY BLOCK STRUCTURES

I here give a brief description of the system's higher level functions along with a scenario giving their interaction in a very simple situation. The main purpose is to illustrate the top down, goal oriented and environment dependent flavor of the system. Code samples are available elsewhere (Winston 1971)

Figure 23 shows the possible call paths between some of the programs. Note in particular the network quality that distinguishes the system from the earlier pass oriented metaphor.

Clarity requires that only a portion of the system be described. In particular, the diagram and the discussion omits the following:

- 1) A large number of antecedant and erasing programs which keep the blocks world model up to date.
- 2) A large network of programs which find skeletons and locate lines with particular characteristics.
- 3) A large network of programs that uses the group - hypothesize - criticize idea to find otherwise inaccessible properties of hidden objects.
- 4) A network of programs that jiggles an object if the arm errs too much when placing it.

The Functions

COPY

As Figure 23 shows, COPY simply activates programs that handle the two phases of a copying problem; namely, it calls for

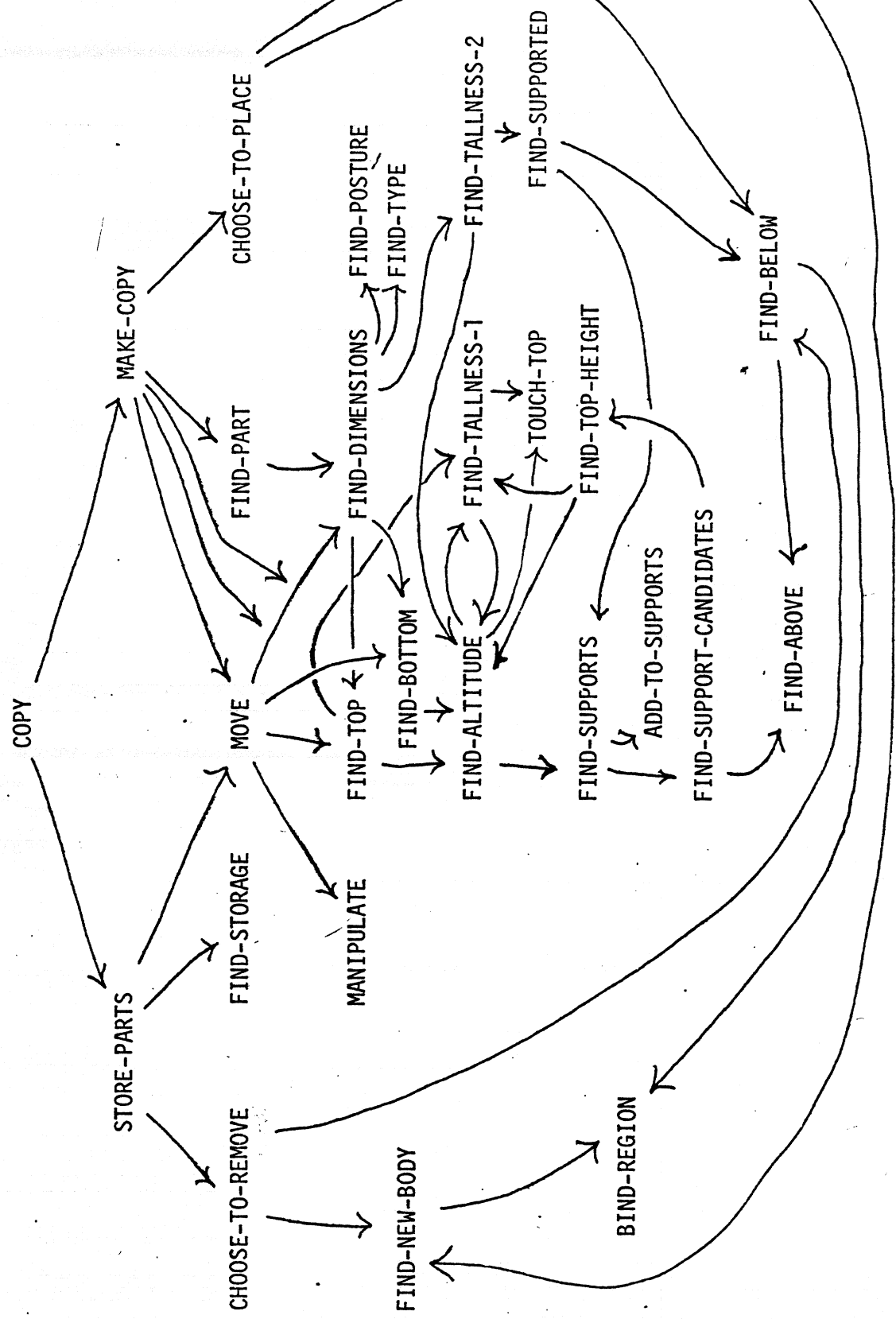


Figure 23
The vision system.

the spare parts to be found and put away into the spare parts warehouse area, and it initiates the replication of the new scene.

STOPE-PARTS

To disassemble a scene and store it, STOPE-PARTS loops through a series of operations. It calls appropriate routines for selecting an object, finding a place for it, and for eracting the movement to storage.

CHOOSE-TO-REMOVE

The first body examined by CHOOSE-TO-REMOVE comes directly from a successful effort to amalgamate some regions into a body using FIND-NEW-BODY. After some body is created, CHOOSE-TO-REMOVE uses FIND-BELOW to make sure it is not underneath something. Frequently, some of the regions surrounding a newly found body are not yet connected to bodies, so FIND-BELOW has a request link to BIND-REGION. (The bodies so found, of course, are placed in the data base and are later selected by CHOOSE-TO-REMOVE without appeal to FIND-NEW-BODY.)

FIND-NEW-BODY

FIND-NEW-BODY locates some unattached region and sets BIND-REGION to work on it. BIND-REGION then calls collection of programs by Eugene Freuder which do a local parse and make

assertions of the form:

(R17 IS-A-FACE-OF F2)

(B2 IS-A PCY)

These programs appeal to a complicated network of subroutines that drive line finding and vertex finding primitives around the scene looking for complete regions (Winston 1972).

FIND-BELOW

As mentioned, some regions may need parsing before it makes sense to ask if a given object is below something. After assuring itself that an adjacent region is attached to a body, FIND-BELOW calls the FIND-ABOVE programs to do the work of determining if the body originally in question lies below the object owning that adjacent region.

FIND-ABOVE-1 and FIND-ABOVE-2 and FIND-ABOVE-3

The heuristics implemented in Winston's thesis (Winston 1970) and many of those only proposed there are now working in the FIND-ABOVE programs. They naturally have a collection of subordinate programs and a link to FIND-REGION for use in the event an un bodied region is encountered. The assertions made are of the form:

(B3 IS-ABOVE B7)

MOVE

To move an object to its spare parts position, the

(B7 IS-A { CUBE
 BRICK
 STICK
 BOARD })

FIND-DIMENSIONS

This program uses FIND-TOP to get the information necessary to convert drawing coordinates to three-dimensional coordinates. If the top is totally obscured, then it appeals instead to FIND-BOTTOM and FIND-TALLNESS-2.

SKELETON

SKELETON identifies connected sets of 3 lines which define the dimensions of a brick (Finin 1971) (Finin 1972). It and the programs under it are frequently called to find instances of various types of lines.

FIND-TALLNESS-1

Determining the tallness of a brick requires observation of a complete vertical line belonging to it. FIND-TALLNESS-1 uses some of SKELETON's repertoire of subroutines to find a good vertical. To convert from two-dimensional to three-dimensional

coordinates, the altitude of the brick must also be known.

FIND-TALLNESS-2

Another program for tallness looks upward rather than downward. It assumes the altitude of a block can be found but no complete vertical line is present which would give the tallness. It tries to find the altitude of a block above the one in question by touching it with the hand. Subtracting the altitude of the lower block from that of the higher gives the desired tallness.

FIND-ALTITUDE

FIND-ALTITUDE determines the height of an object's base primarily by finding its supporting object or objects. If necessary, it will use the arm to try to touch the object's top and then subtract its tallness.

FIND-SUPPORTS

This subroutine uses FIND-SUPPORT-CANDIDATES to collect together those objects that may possibly be supports. FIND-SUPPORT-CANDIDATES decides that a candidate is in fact a support if its top is known to be as high as that of any other support candidate. If the height of a candidate's top is unknown but a lower bound on that height equals the height of known supports, then ADD-TO-SUPPORTS judges it also to be a valid support. At

the moment the system has no understanding of gravity.

FIND-STORAGE

Once an object is chosen for removal, FIND-STORAGE checks the warehouse area for an appropriate place to put it.

MAKE-COPY

To make the copy, MAKE-COPY, CHOOSE-TO-PLACE, and FIND-PART replace STORE-PARTS, CHOOSE-TO-REMOVE and FIND-STORAGE.

Assertions of the form:

(B12 IS-A SPAREPART)

(B2 IS-A-PART-OF COPY)

(B2 IS-ABOVE B1)

are kept up to date throughout by appropriate routines.

CHOOSE-TO-PLACE

Objects are placed after it is insured that their supports are already placed.

FIND-PART

The part to be used from the warehouse is selected so as to minimize the difference in dimensions of the matched objects.

A Scenerio

In what follows the scene in figure 24a provides the spare parts which first must be put away in the warehouse. The scene to be copied is that of Figure 24b.

COPY

COPY begins the activities.

STORE-PARTS

STORE-PARTS begins supervision of disassembly.

CHOOSE-TO-REMOVE
FIND-NEW-BODY
BIND-REGION

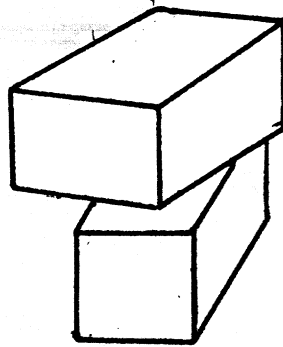
CHOOSE-TO-REMOVE parses a few regions together into a body, B1. A great deal of work goes into finding these regions by intelligent driving of low level line and vertex finding primitives.

FIND-BELOW
BIND-REGION
FIND-ABOVE

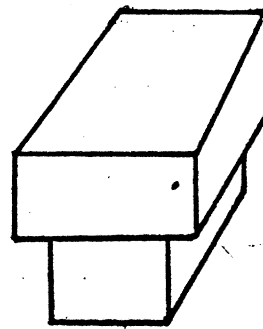
A check is made to insure that the body is not below anything. Note that B2 is parsed during this phase as required for the FIND-ABOVE routines. Unfortunately B1 is below B2 and therefore CHOOSE-TO-REMOVE must select an alternative for removal.

FIND-BELOW
FIND-ABOVE

B2 was found while checking out B1. CHOOSE-TO-REMOVE now notices it in the data base and confirms that it is not below anything.



a



b

Figure 24
A source of spare parts and a scene to be copied.

FIND-STORAGE

FIND-STORAGE finds an empty spot in the warehouse.

MOVE

MOVE initiates the work of finding the location and dimensions of B2.

FIND-TOP

FIND-ALTITUDE

FIND-SUPPORTS

FIND-SUPPORT-CANDIDATES

FIND-TOP-HEIGHT

FIND-ALTITUDE

FIND-SUPPORTS

FIND-SUPPORT-CANDIDATES

FIND-TOP-HEIGHT

FIND-TALLNESS-1

FIND-TALLNESS-1

FIND-BOTTOM proceeds to nail down location parameters for B2. As indicated by the depth of call, this requires something of a detour as one must first know B2's altitude, which in turn requires some facts about B1. Note that no calls are made to FIND-ABOVE routines during this sequence as those programs previously were used on both B1 and B2 in determining their suitability for removal.

FIND-DIMENSIONS

A call to FIND-DIMENSIONS succeeds immediately as the necessary facts for finding dimensions were already found in the course of finding location. Routines establish that B2 is a lying brick.

MANIPULATE

MANIPULATE executes the necessary motion.

CHOOSE-TO-REMOVE
 FIND-BELOW
 FIND-STORAGE

B2 is established as appropriate for transfer to the warehouse. A place is found for it there.

MOVE
 FIND-TOP
 FIND-DIMENSIONS
 MANIPULATE

The move goes off straightforwardly, as essential facts are in the data base as side effects of previous calculations.

CHOOSE-TO-REMOVE
 FIND-NEW-BODY

No more objects are located in the scene. At this point the scene to be copied, figure 24, is placed in front of the eye and analysis proceeds on it.

MAKE-COPY
 CHOOSE-TO-PLACE
 FIND-NEW-BODY
 BIND-REGION

B3 is found.

FIND-BELOW
 BIND-REGION
 FIND-ABOVE

B3 is established as ready to be copied with a spare part.

FIND-PART
 FIND-DIMENSIONS
 FIND-TOP

Before a part can be found, B3's dimensions must be found. The first program, FIND-TOP, fails.

FIND-BOTTOM

FIND-ALTITUDE
 FIND-SUPPORTS
 FIND-SUPPORT-CANDIDATES
 FIND-TOP-HEIGHT

FIND-DIMENSIONS tries an alternative for calculating dimensions. It starts by finding the altitude of the bottom.

FIND-TALLNESS-2
 FIND-SUPPORTED
 FIND-BELOW
 FIND-ABOVE
 FIND-SUPPORTS
 FIND-SUPPORT-CANDIDATES

FIND-TALLNESS-2 discovers B4 is above B3.

FIND-ALTITUDE
 TOUCH-TOP
 FIND-TALLNESS-1

FIND-ALTITUDE finds B4's altitude by using the hand to touch its top subtracting its tallness. B3's height is found by subtracting B3's altitude from that of B4.

MOVE
 MANIPULATE

Moving in a spare part for B3 is now easy. B3's location was found while dealing with its dimensions.

CHOOSE-TO-PLACE
 FIND-BELOW
 FIND-PART
 FIND-DIMENSIONS
 FIND-TOP
 MOVE
 MANIPULATE

Placing a part for B4 is easy as the essential facts are now already in the data base.

CHOOSE-TO-REMOVE
 FIND-NEW-BODY

No other parts are found in the scene to be copied.

Success.

CONCLUDING REMARKS

This essay began with the claim that the study of vision contributes both to artificial intelligence and to a theory of vision. Working with a view toward these purposes has occupied many years of study at MIT and elsewhere on the toy world of simple polyhedra. The progress in semantic rooted scene analysis, learning, and copying have now brought us to a plateau where we expect to spend some time deciding what the next important problems are and where to look for solutions.

The complete system, which occupies on the order of 100,000 thirty-six bit words, is authored by direct contributions in code from over a dozen people. This essay has not summarized, but rather has only hinted at the difficulty and complexity of the problems this group has faced. Many important issues have not been touched on here at all. Line finding, for example, is a task on which everything rests and has by itself occupied more effort than all the other work described here (Roberts 1967) (Herskovits and Binford 1970) (Griffith 1970) (Horr 1971) (Shirai 1972).

References

- Clowes, M., "On Seeing Things," Artificial Intelligence, Vol. 2, No 1, Spring, 1971.
- Ernst, H., "MH-1, A Computer-Operated Mechanical Hand," D.Sc. Dissertation, Department of Electrical Engineering, M.I.T., December, 1961.
- Finin, T., "Finding the Skeleton of a Frick," Vision Flash 19, Artificial Intelligence Laboratory, M.I.T., August, 1971.
- Finin, T., "A Vision Potpourri," Vision Flash 26, Artificial Intelligence Laboratory, M.I.T., June, 1972.
- Griffith, A., "Computer Recognition of Prismatic Solids," MAC Technical Report 73, Project MAC, M.I.T., August, 1970.
- Guzman, A., "Computer Recognition of Three-Dimensional Objects in a Visual Scene," MAC Technical Report 59, Project MAC, M.I.T., December, 1968.
- Herskovits, A. and Binford, T., "On Boundary Detection," A.I. Memo. 183, Artificial Intelligence Laboratory, M.I.T., July, 1970.
- Hewitt, C., "Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot," A.I. Technical Report 258, Artificial Intelligence Laboratory, M.I.T., April, 1972.
- Horn, F. K. P., "The Binford-Horn Line Finder," Vision Flash 16, Artificial Intelligence Laboratory, M.I.T., June, 1971.
- Huffman, D., "Impossible Objects as Nonsense Sentences," Machine Intelligence 6, pp. 295-323 (ed. Meltzer, F. & Michie, D.), Edinburgh University Press, Edinburgh, 1971.
- McDermott, D. and Sussman, G., "The CONNIVER Reference Manual," A. I. Memo. 259, Artificial Intelligence Laboratory, M.I.T., May, 1972.
- Minsky, M. and Papert, S., "Progress Report," A. I. Memo. 252, Artificial Intelligence Laboratory, M.I.T., January, 1972.

- Orban, R., "Removing Shadows in a Scene," A.I. Memo. 192, Artificial Intelligence Laboratory, M.I.T., August, 1970.
- Shirai, Y., "A Heterarchical Program for Recognition of Polyhedra," A.I. Memo. 263, Artificial Intelligence Laboratory, M.I.T., June, 1972.
- Simon, H., The Sciences of the Artificial, M.I.T. Press, M.I.T., Cambridge, 1969.
- Sussmar, G., Winograd, T., and Charniak, E., "MICRO-PLANEF Reference Manuel," A.I. Memo. 203A, Artificial Intelligence Laboratory, M.I.T., December, 1971.
- Rattner, M., "Extending Guzman's SFE Program," A.I. Memo. 204, Artificial Intelligence Laboratory, M.I.T., July, 1970.
- Roberts, L., "Machine Perception of Three-Dimensional Solids," Technical Report 315, Lincoln Laboratory, M.I.T., May, 1963.
- Waltz, D., "Shedding Light on Shadows," Vision Flash 29, Artificial Intelligence Laboratory, M.I.T., July, 1972.
- Waltz, D., Doctoral Dissertation and A.I. Technical Report in preparation, Artificial Intelligence Laboratory, M.I.T.
- Wichmar, W., "Use of Optical Feedback in the Computer Control of an Arm," A.I. Memo. 56, Stanford Artificial Intelligence Project, Stanford University, August, 1967.
- Winston, P. H., "Holes," A.I. Memo. 163, Artificial Intelligence Laboratory, M.I.T., August, 1968.
- Winston, P. H., "Learning Structural Descriptions from Examples," A.I. Technical Report 231, Artificial Intelligence Laboratory, M.I.T., September, 1970.
- Winston, P. H., "Wandering About the Top of the Robot," Vision Flash 15, Artificial Intelligence Laboratory, M.I.T., June, 1971.
- Winston, P. H., "Wizard," Vision Flash 24, in preparation, Artificial Intelligence Laboratory, M.I.T., 1972.
- Winston, P. H. and Lerman, J., "Circular Scan," Vision Flash 23, Artificial Intelligence Laboratory, M.I.T., March, 1972.

SHEDDING LIGHT ON SHADOWS

David I. Waltz

July 1972

ABSTRACT

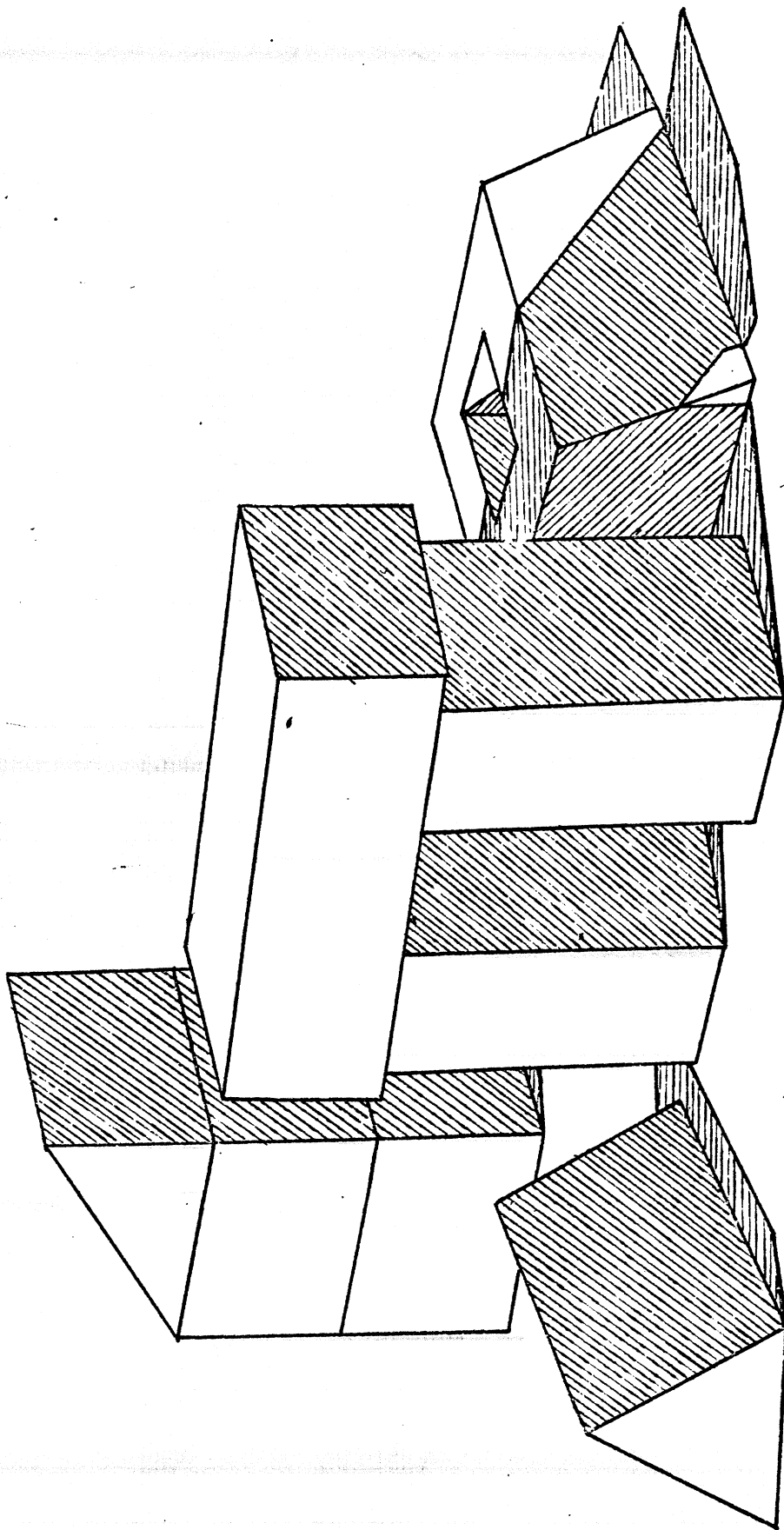
This paper describes methods which allow a program to analyze and interpret a variety of scenes made up of polyhedra with trihedral vertices. Scenes may contain shadows, accidental edge alignments, and some missing lines. This work is based on ideas proposed initially by Huffman and Clowes; I have added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

This paper was originally published as Vision Flash 29.

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, and use rote memory to assign shape to those objects when we



FIGURE

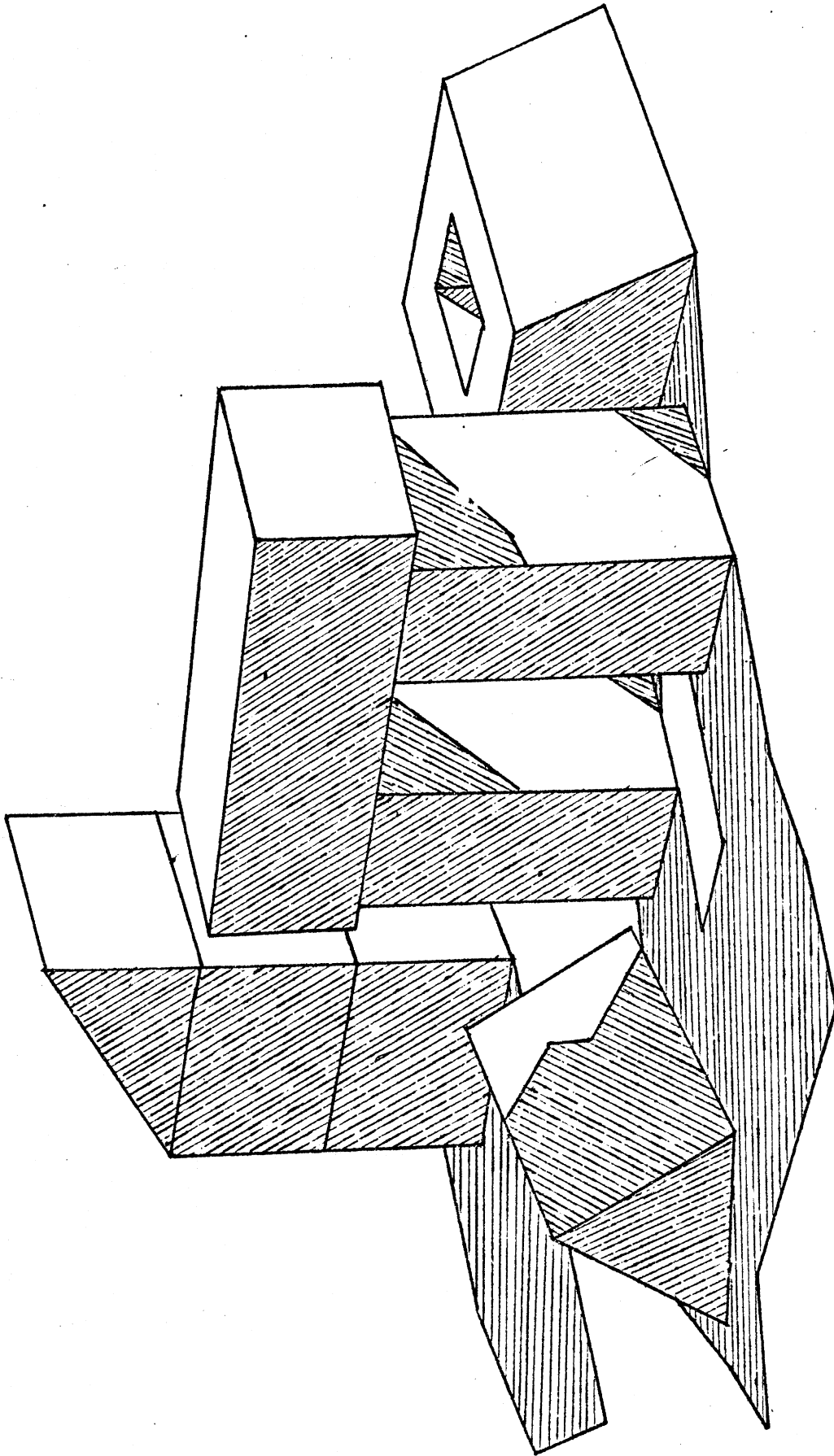


FIGURE 12

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 Cuzman 1968, Lerner 1970) and in building a system incorporating hand-eye coordination which could be used to help explore and disambiguate a scene (Caschnig 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 is to 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 of junction in a line drawing. While a natural language analogy to these labels could be misleading, I think that it helps in explaining the basic operation of this portion of the program.

If we think of each possible label for a line as a letter in the alphabet, then each junction must be labeled with an ordered list of "letters" to form a legal "word" in the language. 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 is interesting in its own right. While some basic edge geometries give rise to many dictionary entries, some give rise to very few. The total number of entries sharing the same edge geometry can be as low as three for some ARFCW junctions, including shadow edges, while the number generated by some FORK junction edge geometries is over 270,000 (including region orientation and illumination values).

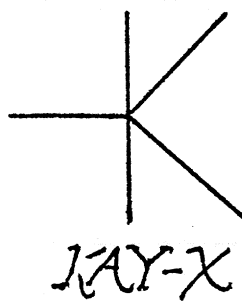
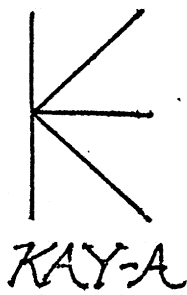
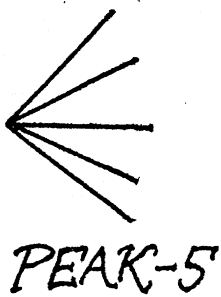
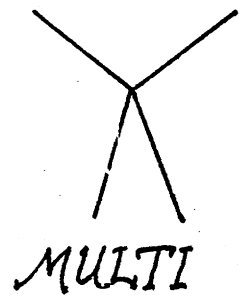
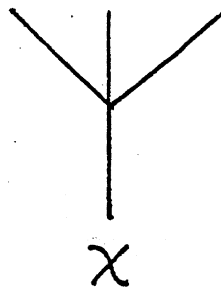
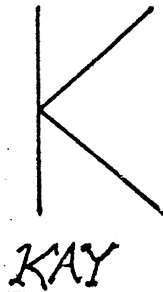
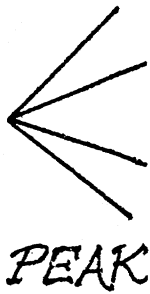
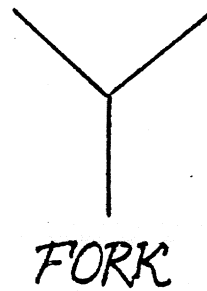
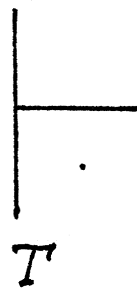
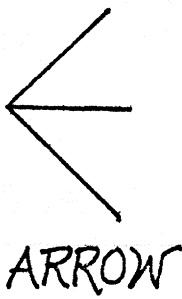
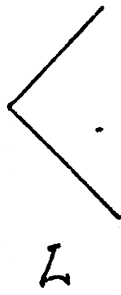
1.3 JUNCTION LABEL ASSIGNMENT

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 is already included implicitly in the idea of junction type. Junctions are typed according to the number of lines which make up the junction and the two dimensional arrangement of these lines. Figure 1.7 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 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

FIGURE 1.3

JUNCTION TYPES



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, 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.

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 LIST.

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, scene 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.

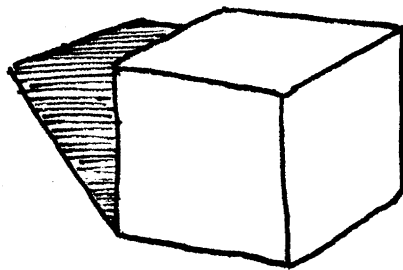
Fifth, because of this simplicity, I have been able to write a program which operates very rapidly. As a practical matter this is very useful for debugging the system, and allows modifications to be made with relative ease. Moreover, because of its speed, I have been able to test the

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.

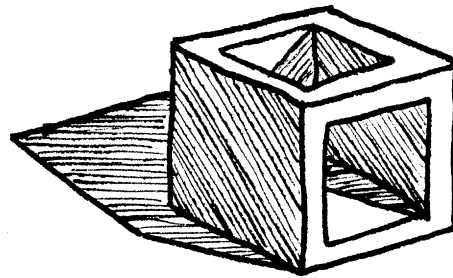
Sixth, as explained in the next section, the descriptive language provides a theoretical foundation of considerable value in explaining previous work.

1.7 HISTORICAL PERSPECTIVE

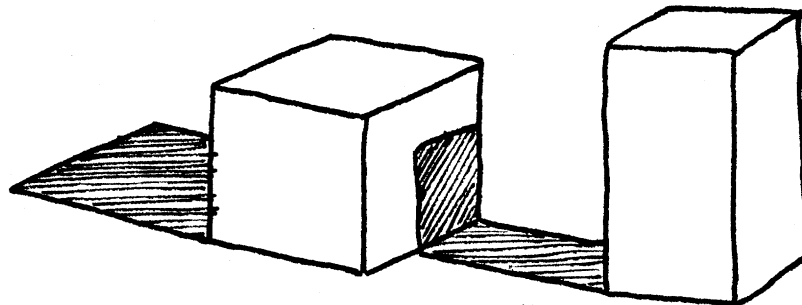
One of the great values of the extensive descriptive apparatus I have developed is its ability to explain the nature and shortcomings of past work. I will discuss in Chapter 9 how my system helps in understanding the work of Guzman (Guzman 1968), Rattner (Rattner 1970), Huffman (Huffman 1971), Clowes (Clowes 1971), and Urban (Urban 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



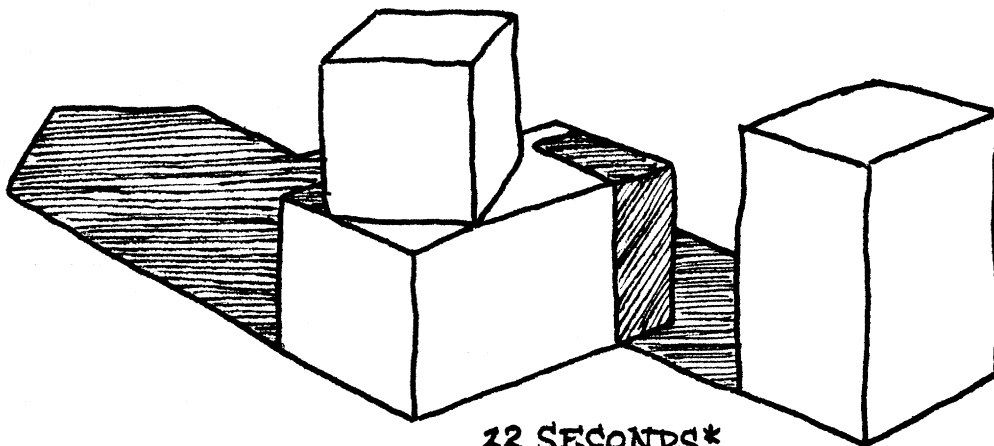
5 SECONDS*



15 SECONDS*



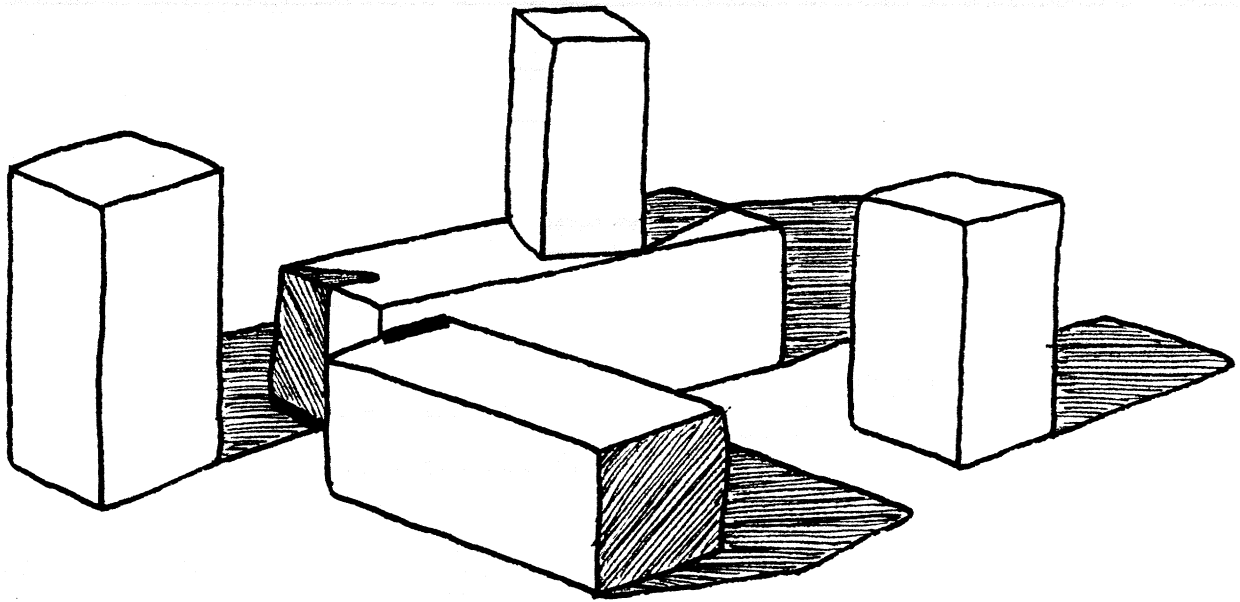
15 SECONDS*



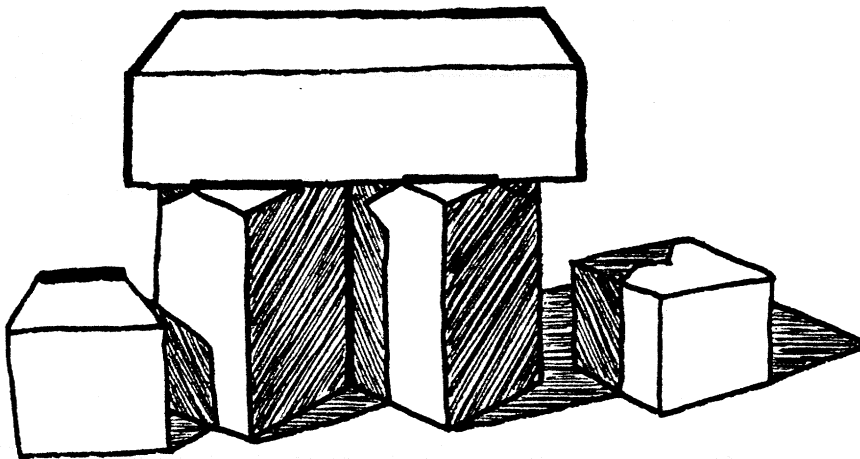
22 SECONDS*

FIGURE 1.4

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



39 seconds



48 seconds

FIGURE 1.5

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.

2.0 QUICK SYNOPSIS

This chapter provides a quick look at some of the technical aspects of my work. It provides a synopsis of work covered more fully in my thesis (A.I. TR-271).

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 is labeled as a shadow edge, line J2-J3 is labeled as a concave edge, line J3-J4 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

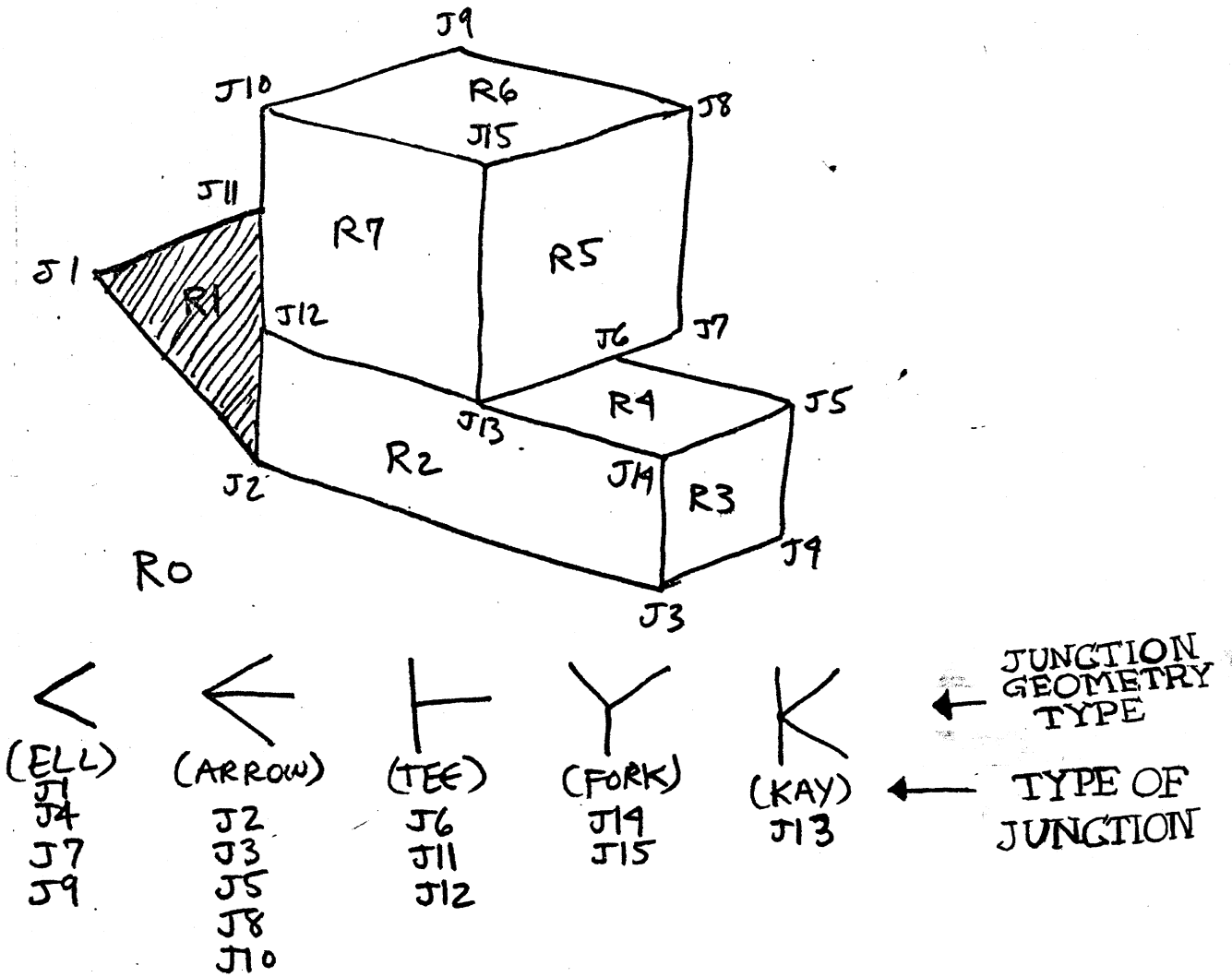


FIGURE 2.1

Preceding page blank

found in the three-dimensional scene.

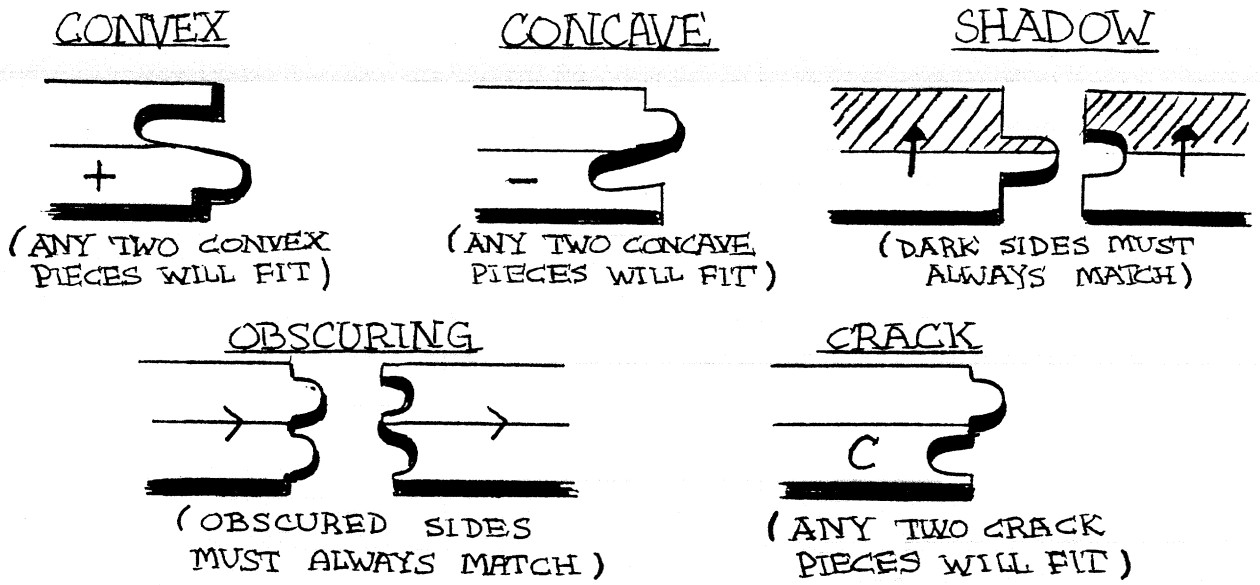
When we look at a line drawing of this sort, we usually can easily understand what the line drawing represents. In terms of a labeling scheme either (1) we are able to assign labels uniquely to each line, or (2) we can say that no such scene could exist, or (3) we can say that although it is impossible 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 several different ways to label any line segment. This implies that if all line labels could be assigned independently there would be $7^7 = 49$ 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 or 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 (triangular vertices) and to junctions of five or fewer lines. In Chapter 3 I explain how to obtain the 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.



ALL ELLS:

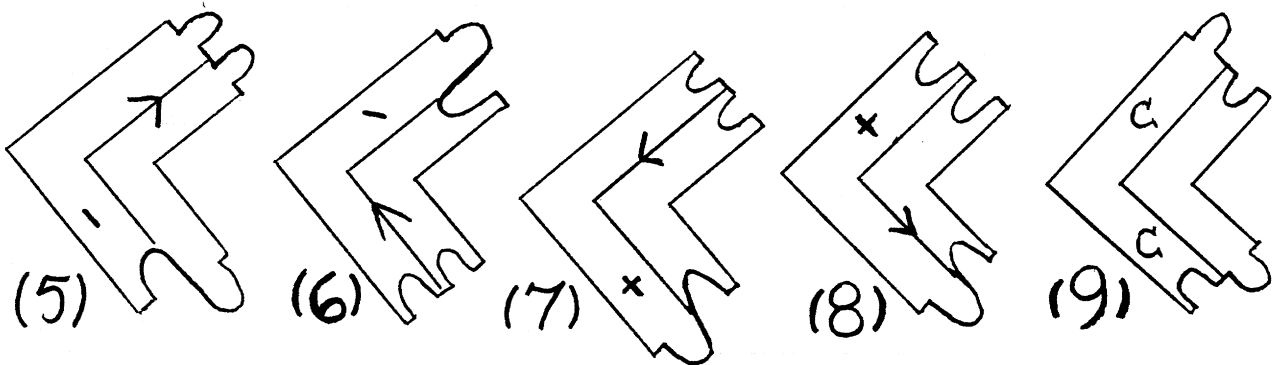
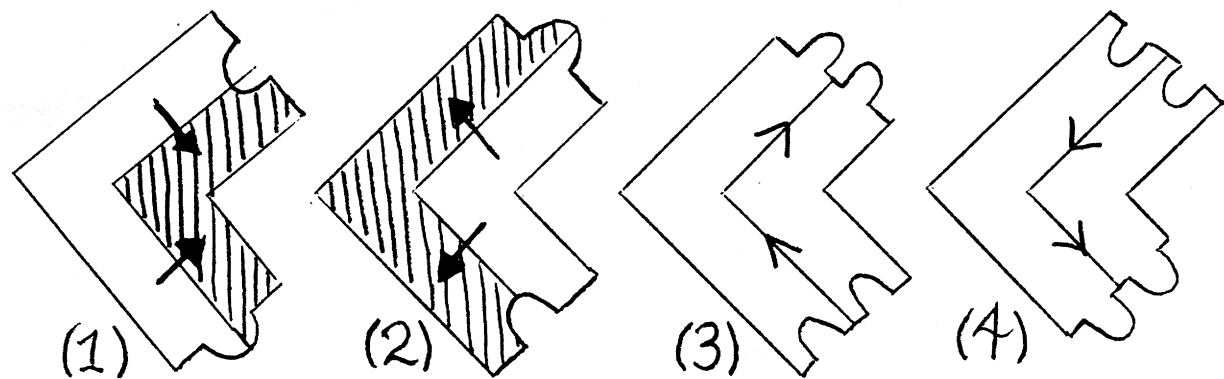


FIGURE 2.2

< PAGE ONE >

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

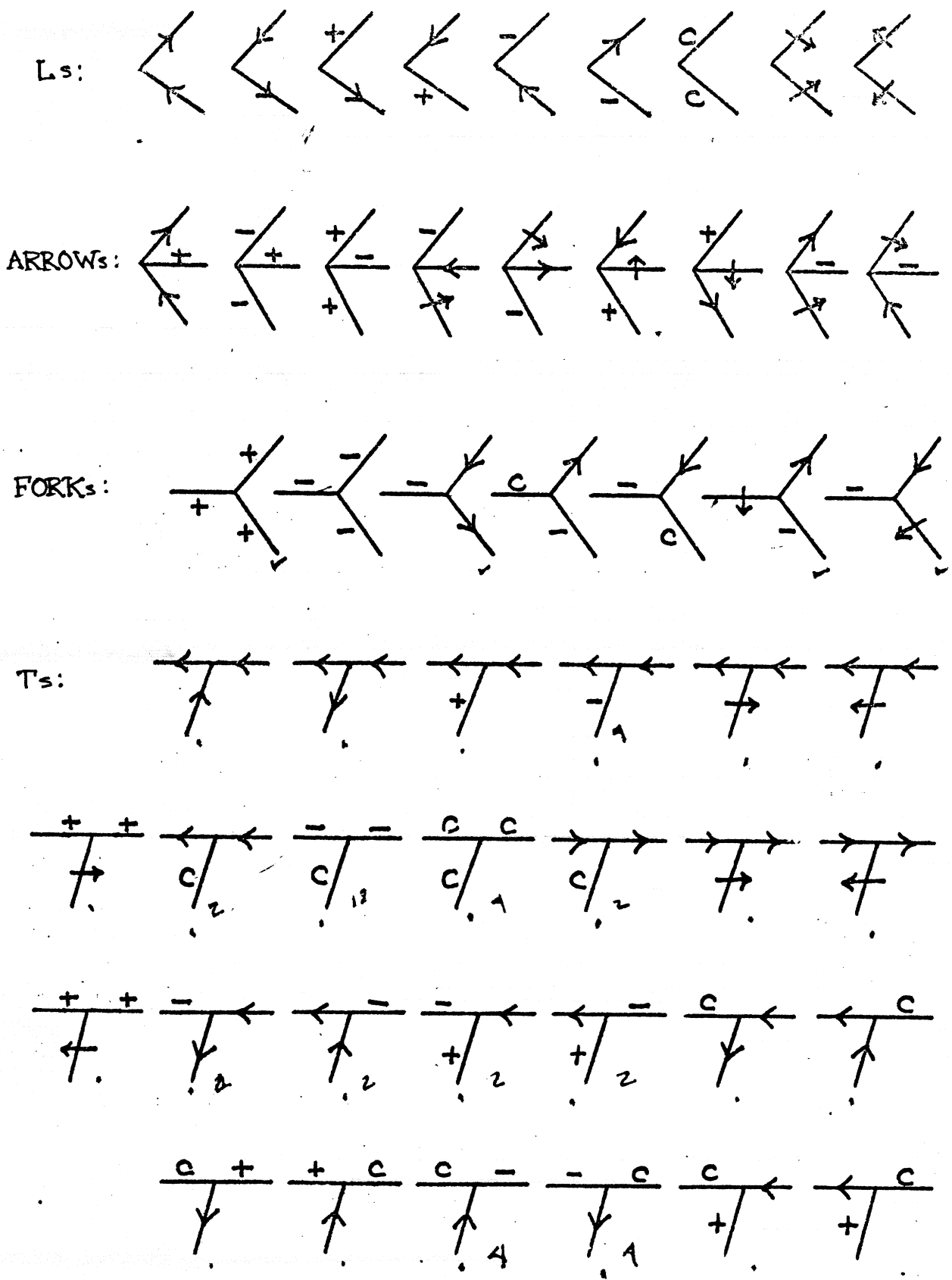
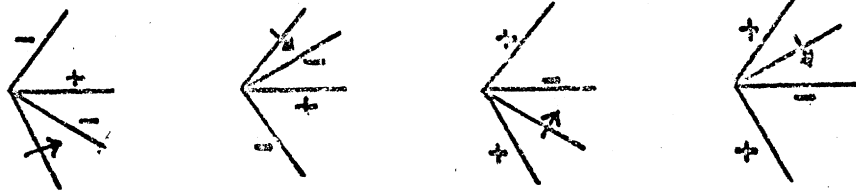
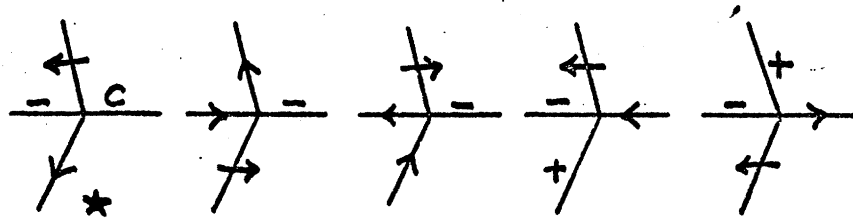
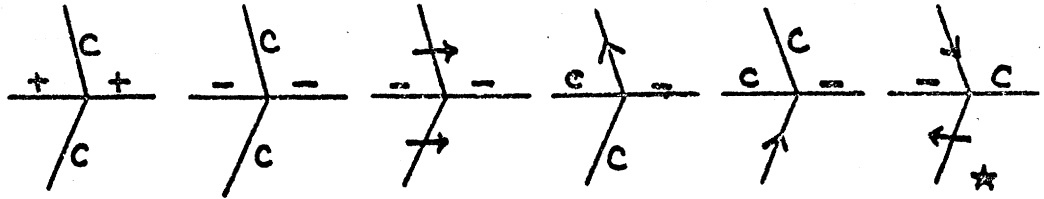


FIGURE 2.3
FIRST PAGE

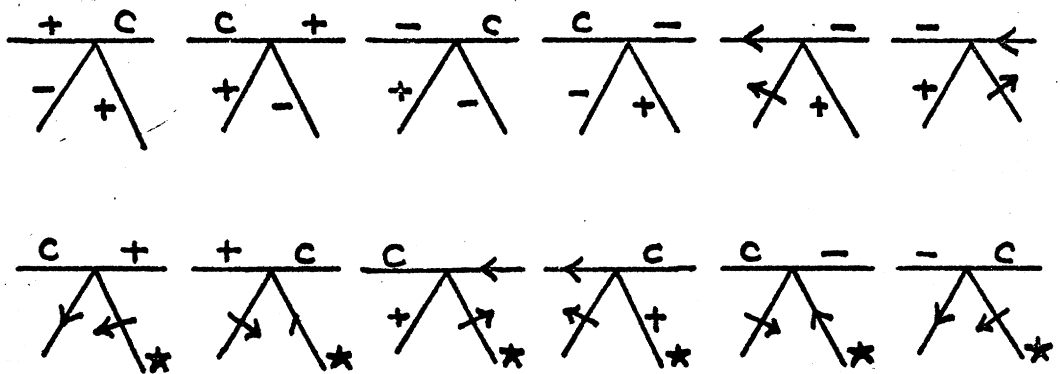
PEAKs:



Xs:



Ks:



Xs:
(SPECIAL)

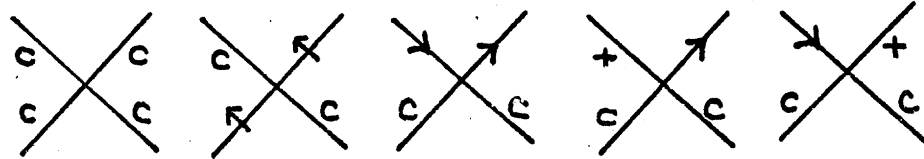
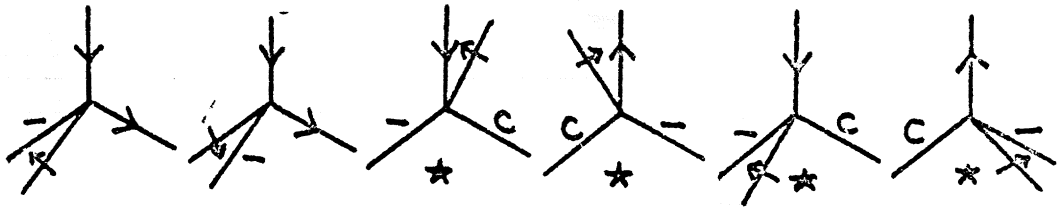
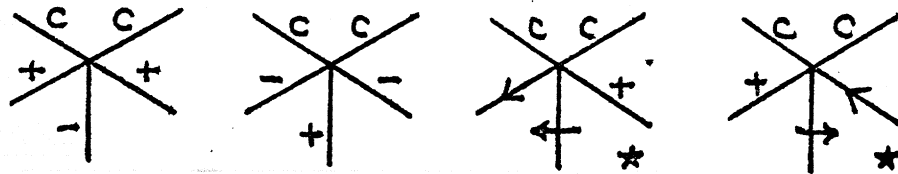


FIGURE 2.3
SECOND PAGE

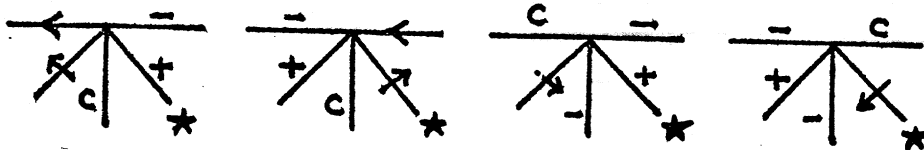
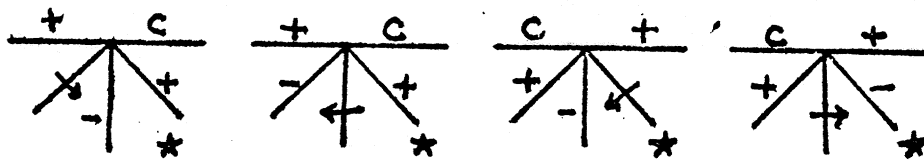
MULTIS:



K-Xs:
(SPECIAL)



K-As:



K-Xs:

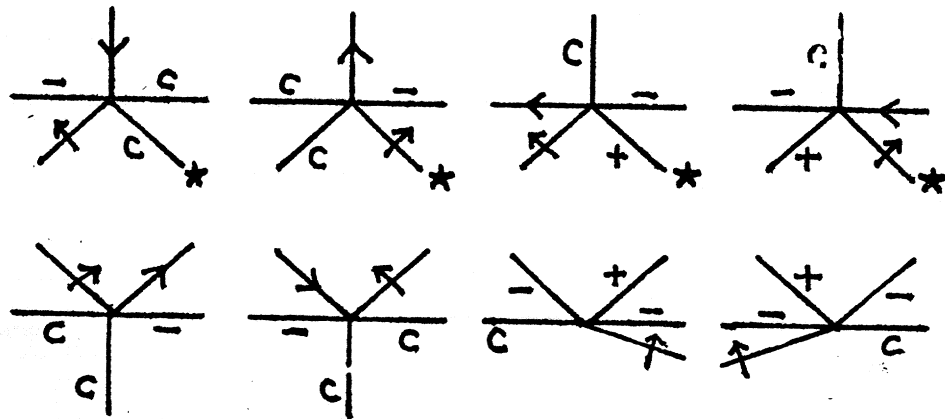


FIGURE 23
THIRD PAGE

[SPECIAL JUNCTIONS CONTAIN
TWO COLINEAR RELATIONS]

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

TABLE 2.1.

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 N_c will be used to refer to the current value of the counter. Thus if the counter is set to 6, then $J(N_c) = 6$.

Step 4: Try to place the top piece from the "untried labels" bin of junction $J(N_c)$ in board space $J(N_c)$. 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 $N_c < n$, increase the counter by one and repeat Step 4.

A2. If $N_c = n$, then the pieces row 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 $N_c = 1$, 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:

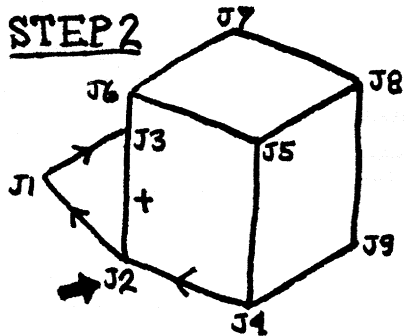
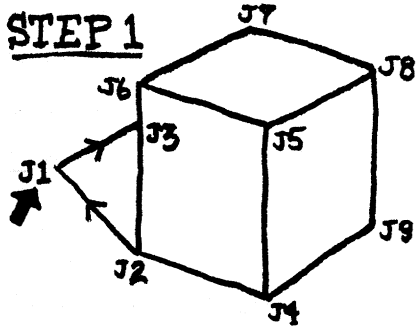
i. Transfer all the pieces from the N_c -th "tried labels" bin into the N_c -th "untried labels" bin, and

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

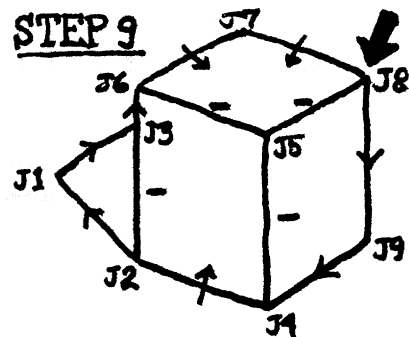
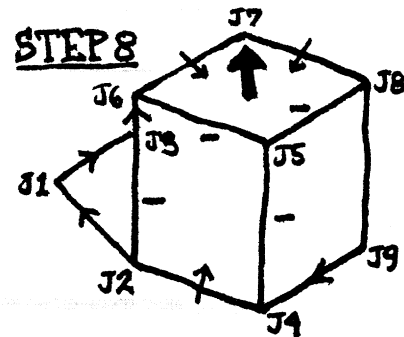
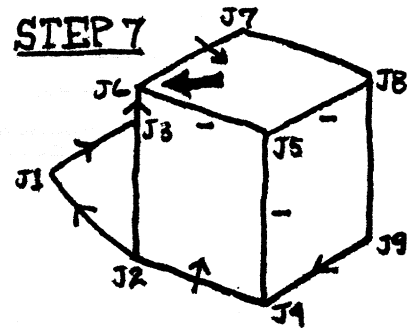
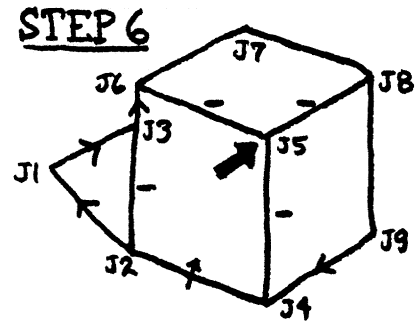
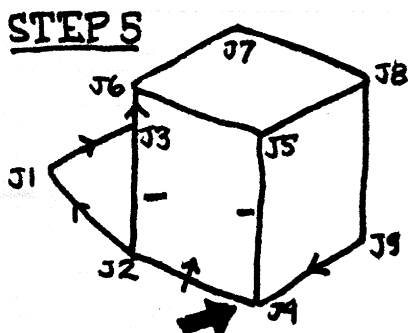
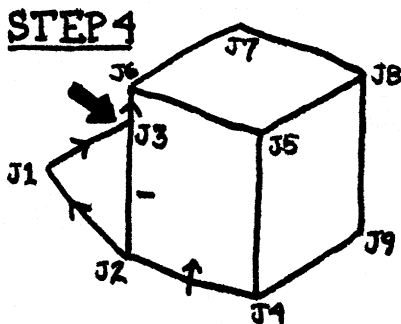
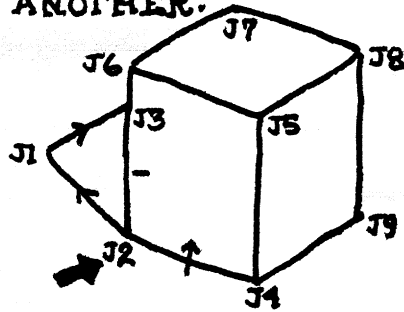
iii. Set the counter to N_c-1 , and go to Step 4.

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

FIGURE 1.1



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



STEP 10

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

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 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 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 2.5 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 (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.

OLD LABELING

NEW LABELING

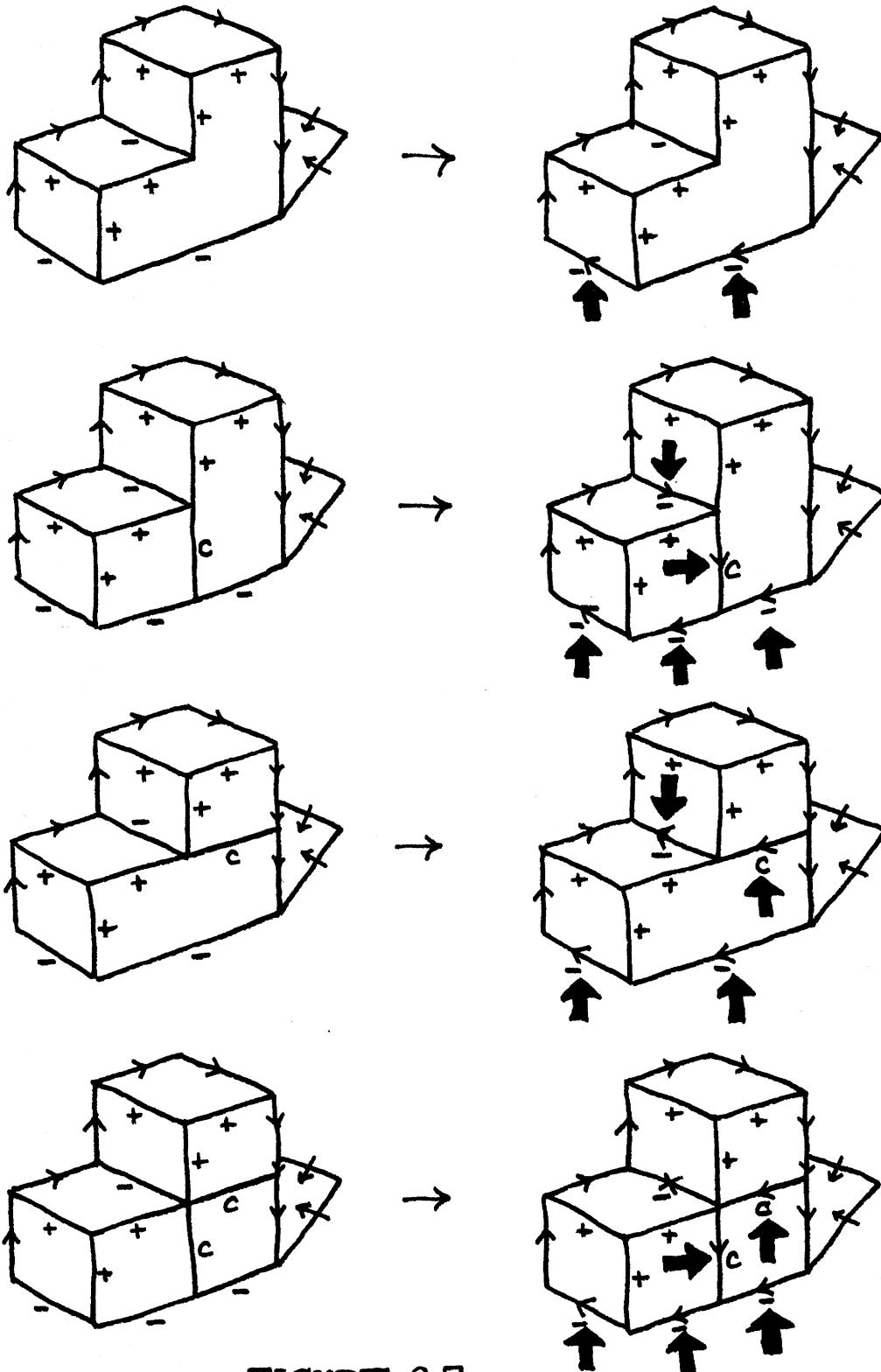


FIGURE 2.5
 <FIRST PAGE>

INTERPRETATION:

- $\frac{R1}{R2} \text{ ---}$ AN INSEPARABLE CONCAVE EDGE; THE OBJECT OF WHICH R1 IS A PART [OB(R1)] IS THE SAME AS THE OBJECT OF WHICH R2 IS A PART [OB(R1) = OB(R2)].
- $\frac{R1}{R2} \text{ ---} \leftarrow$ A SEPARABLE CONCAVE EDGE; IF R1 IS ABOVE R2, THEN OB(R2) SUPPORTS OB(R1).
- $\frac{R1}{R2} \text{ ---} \rightarrow$ SAME AS ABOVE; IF R1 IS ABOVE R2, THEN EITHER OB(R1) SUPPORTS OB(R2) OR OB(R2) IS IN FRONT OF OB(R1).
- $\frac{R1}{R2} \text{ ---} \times$ A 3-WAY SEPARABLE CONCAVE EDGE; NEITHER OBJECT SUPPORTS THE OTHER.
- $\frac{R1}{R2} \text{ ---} \rightarrow^c$ A CRACK EDGE; OB(R2) IS IN FRONT OF OB(R1) IF R1 IS ABOVE R2.
- $\frac{R1}{R2} \text{ ---} \leftarrow^c$ A CRACK EDGE; OB(R2) SUPPORTS OB(R1) IF R1 IS ABOVE R2.

SEPARATIONS:

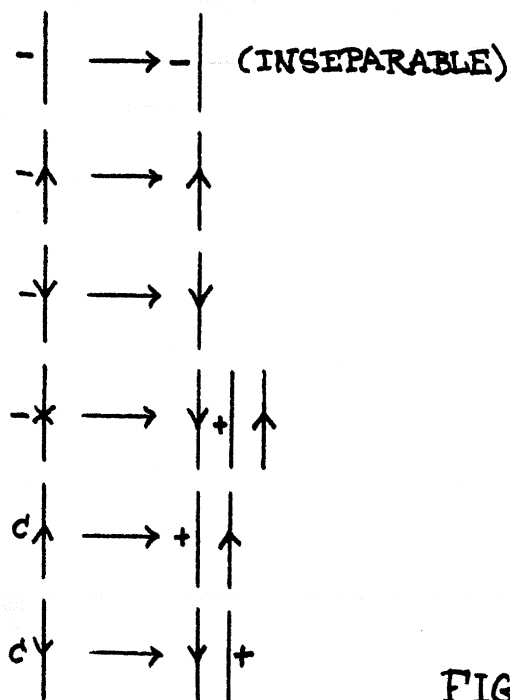
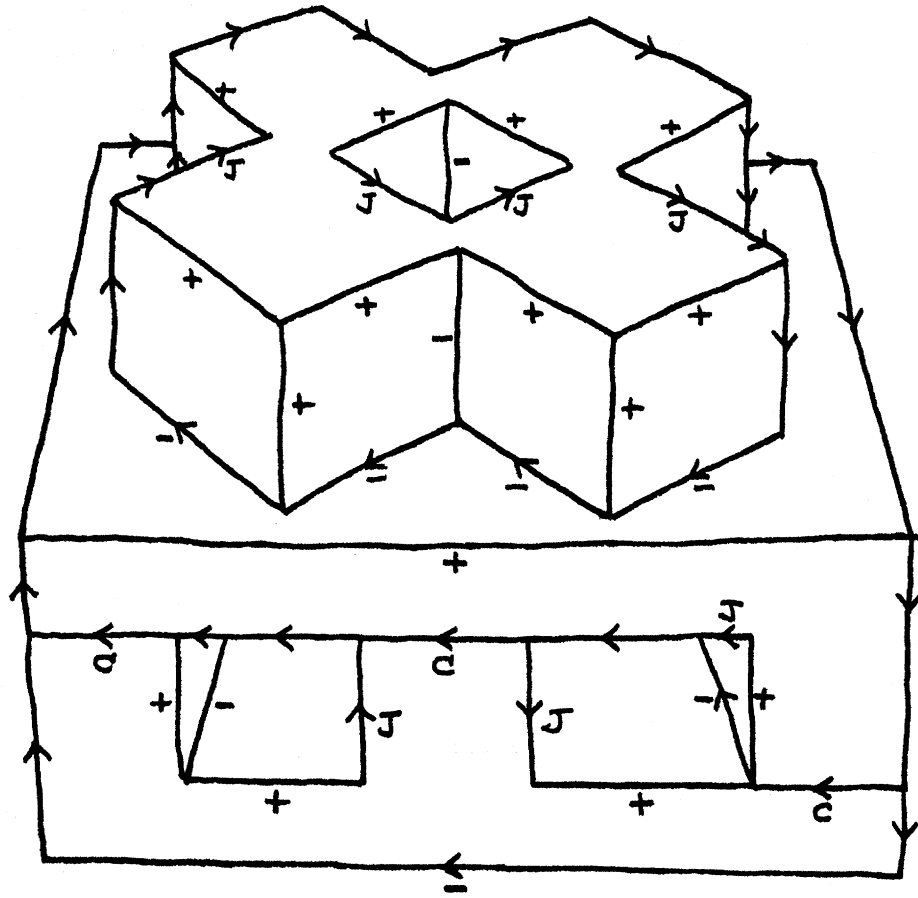


FIGURE 2.5
<SECOND PAGE>



DISTINCTION BETWEEN \rightarrow AND \rightarrow_J .

FIGURE 2.6

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

CONCAVE
1 | 2
- |
(ANY OF FOUR-FIG.1.5)

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

CONVEX
1 | 2
+ |

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

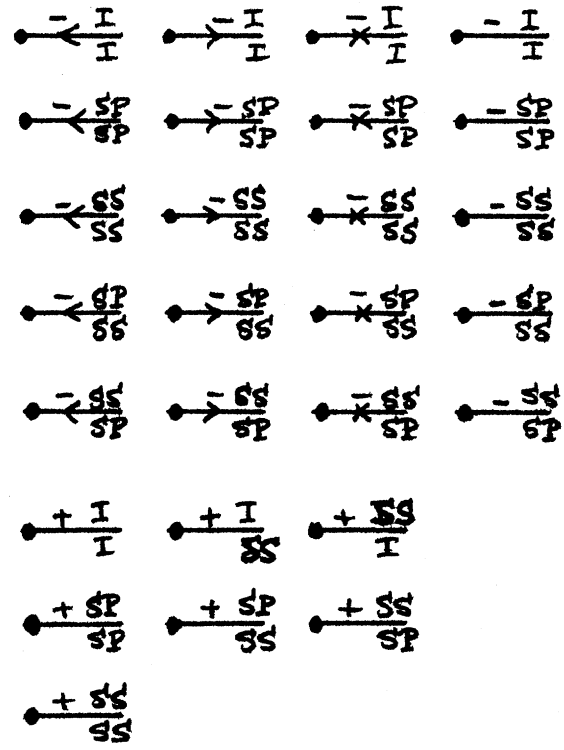
SHADOW
C.C.*
1 | 2
+ → |

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

SHADOW
C.L.*
1 | 2
← |

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

SET OF ALLOWABLE LABELS:



*C.C. = COUNTERCLOCKWISE;
C.L. = CLOCKWISE

FIGURE 2.7
< FIRST PAGE >

SET OF ALLOWABLE LABELS (CONT):

CRACK 1 2 c		1 \ 2	I	SP	SS
		I	<u>YES</u>	NO	NO
OBSCLURE 1 2 OR 1 2 (JOINED OR NOT)		1 \ 2	I	SP	SS
		I	<u>YES</u>	<u>YES</u>	<u>YES</u>
		1 \ 2	SP	SP	SS
		SP	<u>YES</u>	<u>YES</u>	<u>YES</u>
		1 \ 2	SS	SS	SS
		SS	<u>YES</u>	<u>YES</u>	<u>YES</u>

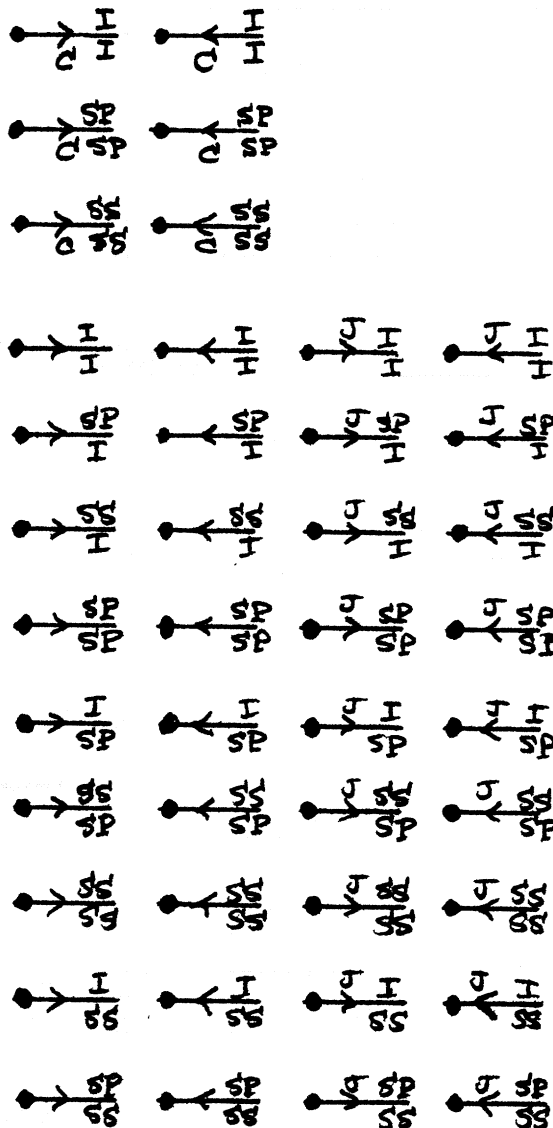


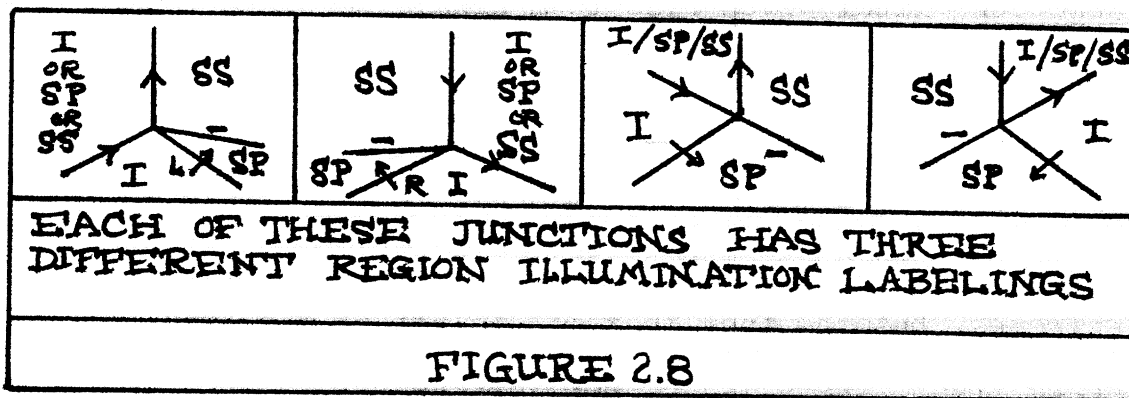
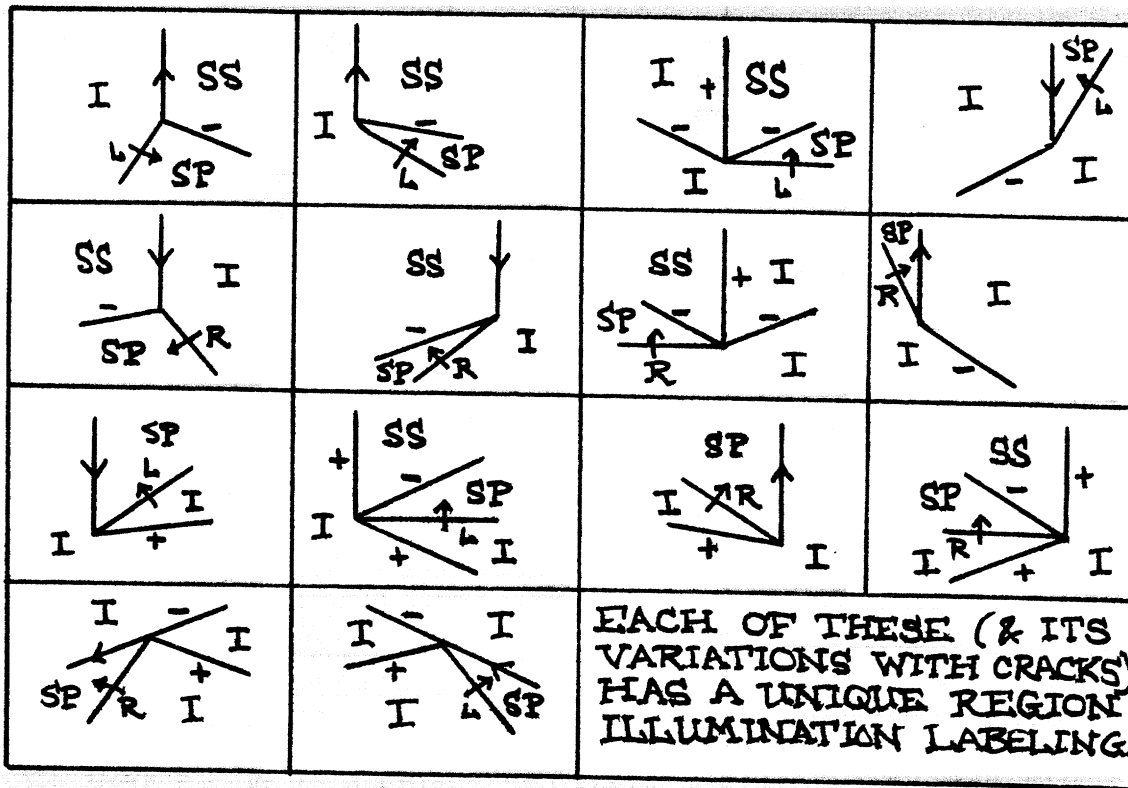
FIGURE 2.7
<SECOND PAGE>

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 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.

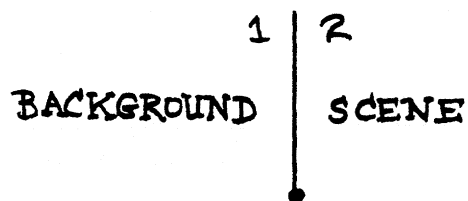
JUNCTION TYPE	NUMBER OF POSSIBILITIES IF EACH BRANCH WIRE LABELED AS INDEPENDENT	NUMBER OF OLD JUNCTIONS EXPANDED INDEPENDENTLY <small>($2^{10} \times 6 \times 3 \times 10^6$ / $2 \times 10^6 \times 10^5$) * CASE (1)</small>	OLD NUMBER OF JUNCTIONS	ACTUAL NUMBER OF NEW JUNCTIONS (APPROX.)	OLD PERCENTAGE (%)	NEW PERCENTAGE (%)
L	4624	1710	9	80	18.4	1.7
ARROW	314,432	12352	9	73	2.6	0.02
FORK	314,432	47382	17	~500	5.0	0.16
T	314,432	61896	26	~1000	7.6	0.32
PEAK	21,381,376	27280	4	8	0.2	~1 x 10 ⁻⁵
X	21,381,376	?	37	414	1.5	~2 x 10 ⁻³
MULTI	21,381,376	?	24	96	1.0	~5 x 10 ⁻⁴
K	21,381,376	?	12	~100	0.5	~5 x 10 ⁻⁴
K-A	145 x 10 ⁹	?	8	30	0.05	~2 x 10 ⁻⁶
K-X	145 x 10 ⁹	?	12	38	0.07	~2.5 x 10 ⁻⁶

*SEE TEXT, PAGE 46

TABLE 2.2

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

The value of a better descriptor 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.



CAN ONLY BE LABELED IN
ONE OF THE FOLLOWING WAYS:

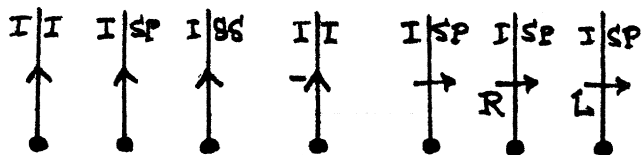


FIGURE 2.9

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 M 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 $N_c = 1$, and place the card so that it reads "no junction modified".

Step 3: Check the value of N_c :

A. If $N_c = n + 1$, and the card reads "no junction modified" then go to SUCCEEDED.

F. If $N_c = n + 1$, 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 $J(N_c)$:

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

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

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

E. If there are pieces left in the N_c -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 N_c :

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

P. If there are empty neighboring spaces, then

P1. If there are no more junctions in the neighboring "untried labels" bins which could fit with the piece in space N_0 , then that piece is not a possible label. Throw it away, and arrange the card to read "junction modified" if it doesn't already.

P2. 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.

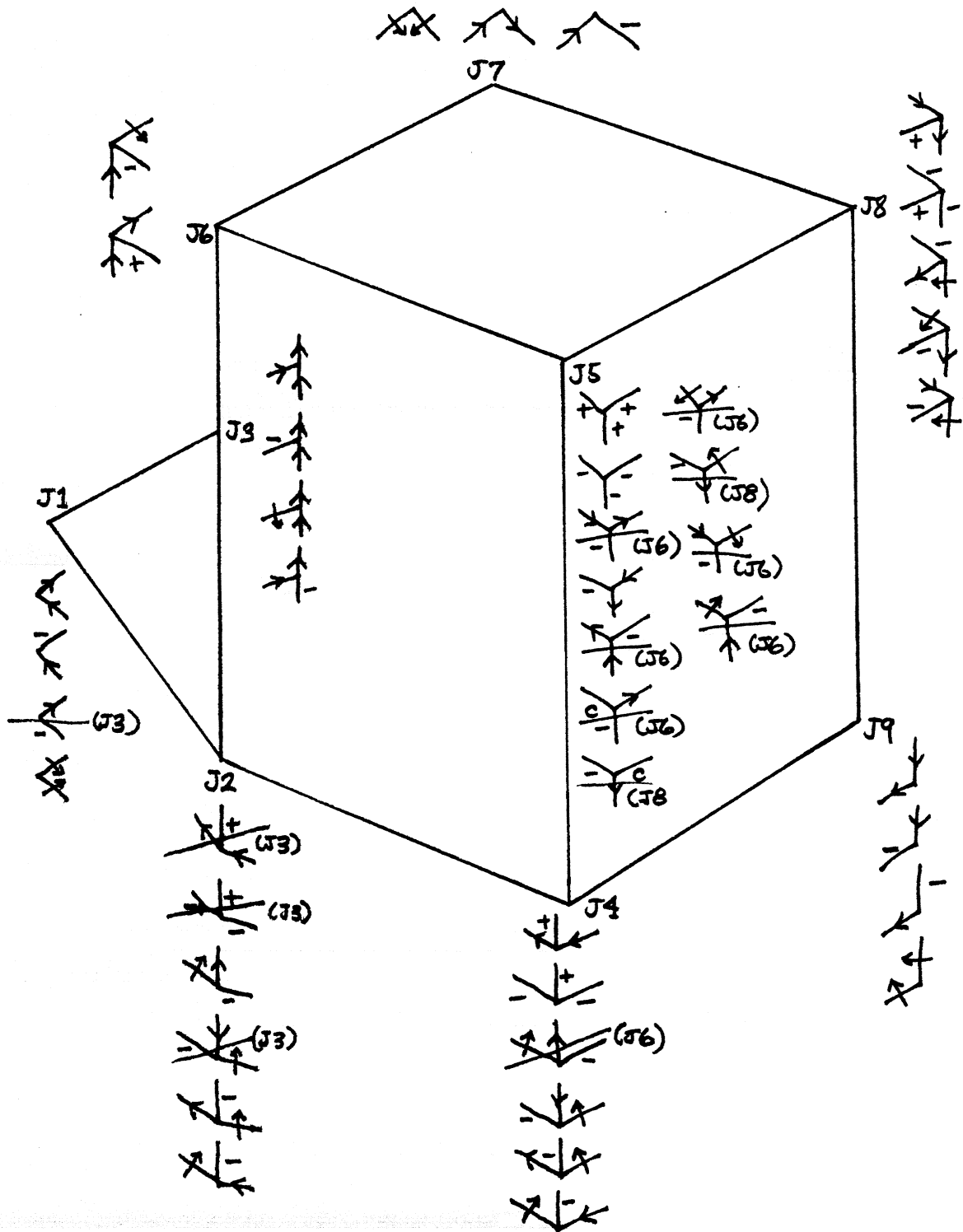
SUCCESS: The pieces in the "untried labels" bins of each junction have passed the filtering routine and constitute the output of this procedure.

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.

FIGURE 2.9



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 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 AFRCWs, FCRKs, etc.), nor is any use made of the length of line segments, microstructure of

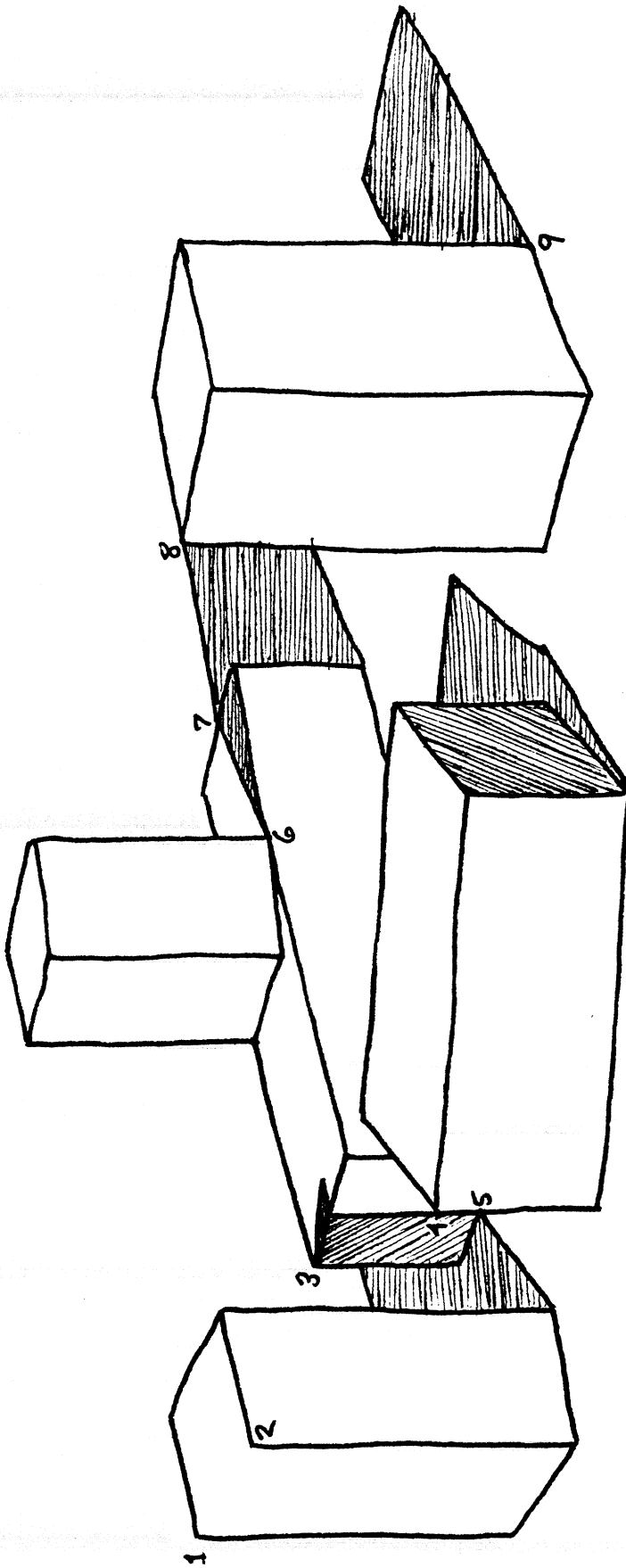
edges, lighting direction or other potentially useful cues.

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



1,2 LINE MISSING

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

4,5 ACCIDENTAL ALIGNMENT

6 NON-TRIHEDRAL JUNCTION FORMED BY ACCIDENTAL ALIGNMENT.

7,8 ACCIDENTAL ALIGNMENT.

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

FIGURE 2.10

of the scene. Even worse, the program may find a way of 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

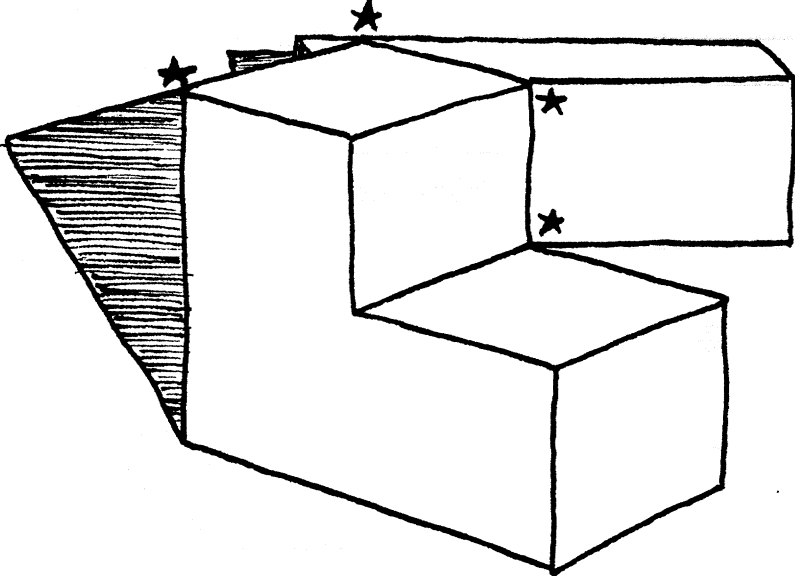


FIGURE 2.11a.

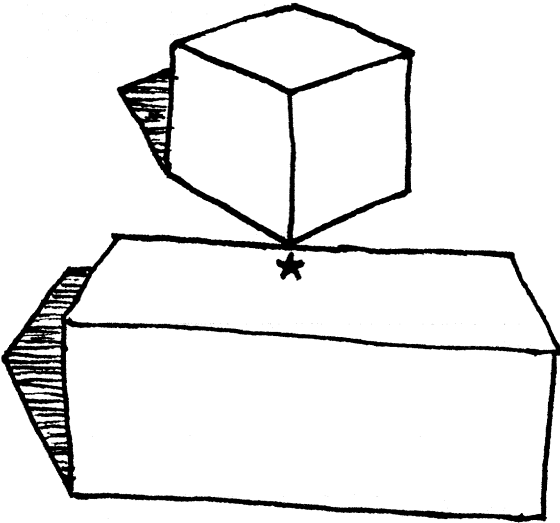


FIGURE 2.11b

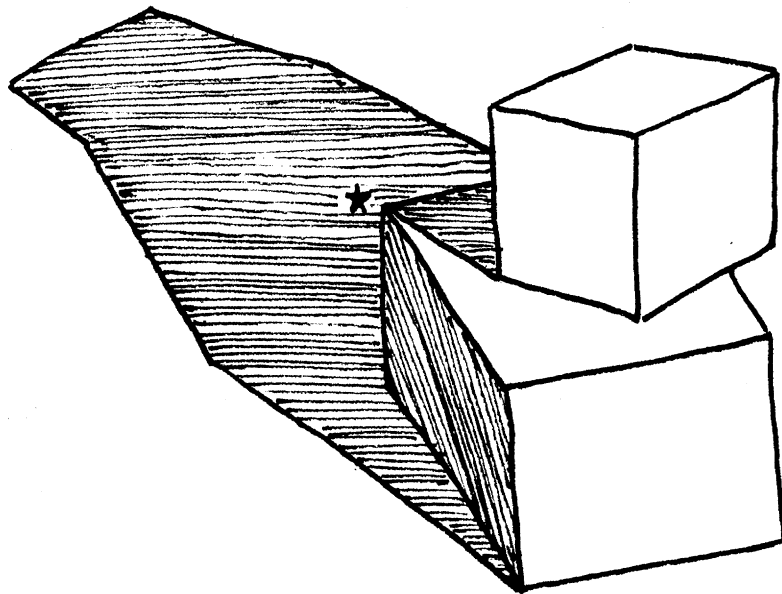


FIGURE 2.11c.

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 is unable 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 are included.

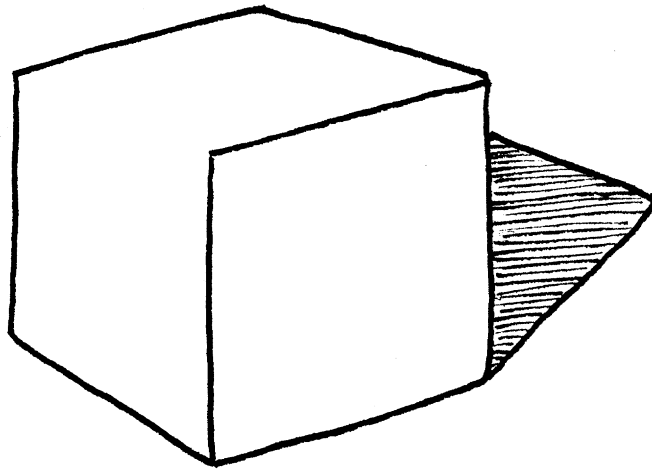


FIGURE 2.12

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.

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.

BIBLIOGRAPHY

- 1 Clowes H B On Seeing Things
AI Journal
Spring 1971
- 2 Lowsen M What Corners Look Like
MIT AI Vision Flash 13
June 1971
- 3 Lowsen M & Waltz D L
Shadows and Cracks
MIT AI Vision Flash 14
June 1971
- 4 Finin T Two Problems in Analyzing Scenes
MIT AI Vision Flash 12
June 1971
- 5 Finin T Finding the Skeleton of a Brick
MIT AI Vision Flash 19
August 1971
- 6 Guzman A Computer Recognition of Three-dimensional
Objects in a Visual Scene
MIT Project MAC Technical Report MAC-TR-59
December 1968
- 7 Huffman E A Impossible Objects as Nonsense Sentences
Machine Intelligence 6
1970
- 8 Mahabala H N V
Preprocessor for Programs Which Recognize Scenes
MIT Project MAC AI Memo 177
August 1969
- 9 Orban R Removing Shadows in a Scene
MIT Project MAC AI Memo 192
August 1970
- 10 Rattner M H Extending Guzman's SEE Program
MIT Project MAC AI Memo 204
July 1970

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

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

- 13 Winston P Learning Structural Descriptions from Examples
 MIT Project MAC Technical Report MAC-TR-75
 September 1970

A VISION POTPOURRI

Tim Finin

June 1972

ABSTRACT

This paper discusses some recent changes and additions to the vision system. Among the additions are the ability to use visual feedback when trying to accurately position an object and the ability to use the arm as a sensory device. Also discussed are some ideas and a description of preliminary work on a particular sort of higher level three-dimensional reasoning.

This paper was originally published as Vision Flash 26.

JIGGLING A BLOCK INTO PLACE

The vision system can now use visual feedback when trying to accurately position a block. This is done without a costly rescanning of a significant portion of the scene by using our knowledge of where the block should be to direct the eye. The basic idea is to determine the block's actual location by looking for certain key vertices using a circular-scanning vertex finder developed by Winston and Lerman < Vision Flash 24 >.

When placing a block the arm sometimes makes positional errors up to half an inch and rotational errors of about 10 degrees. These errors are caused by poor hand placement due to hysteresis and general slop in the arm's joints and by poor information about the brick's initial position and dimensions due to a distorted line drawing. Although these errors can be disastrous in delicate tasks such as stack-building, they are small enough to allow us to use the scheme described below.

The organization of the theorems is shown in figure 1. TC-JIGGLE, the top level theorem, first calls TC-FIND-BODY whose goal is finding the actual location of the just moved brick. This is done by locating a three-vertex 'skeleton' on either the top or bottom of the brick, examples of which are shown in figure 2. Candidate skeletons are suggested by the theorems TC-LOOKFOR-TOP, TC-LOOKFOR-BOTTOM, and TC-LOOKFOR-SKELETON which predict the locations of vertices and decide whether they should be visible. TC-FINDBODY then locates the three vertices comprising the

skeleton with the circular-scan vertex finder and calculates the true position of the brick. If it fails to find one of the vertices, it asks for another skeleton and tries again.

Once the location of the brick is found, TC-SHIFT-BODY calculates the positional and rotational errors and, if they are greater than a tolerance, corrects them through a call to TC-MOVE-GENTLY. This theorem differs from the usual TC-MOVE in calling the arm with GRASP and UNGRASP commands instead of PICKUP and DRCP. PICKUP and DROP raise the arm several feet above the table when moving to avoid obstacles, whereas GRASP and UNGRASP lift the hand less than an inch (using the wrist) and thus, hopefully, are less prone to error.

The most difficult part of this jiggling procedure is determining which vertices of a brick will be visible and not obscured by other objects. We must also avoid looking for vertices which are adjacent to others already in the scene, for example the vertices where two bricks are aligned. Such situations may confuse the vertex finder and cause it to find the wrong vertex. Since these theorems are written to work in the context of a copying task, they use information about the model scene that is being copied. For instance, before TC-LOOKFOR-TOP looks for any vertex on the top of a brick it must either find that:

1. The top of the matching brick in the model was completely visible, or

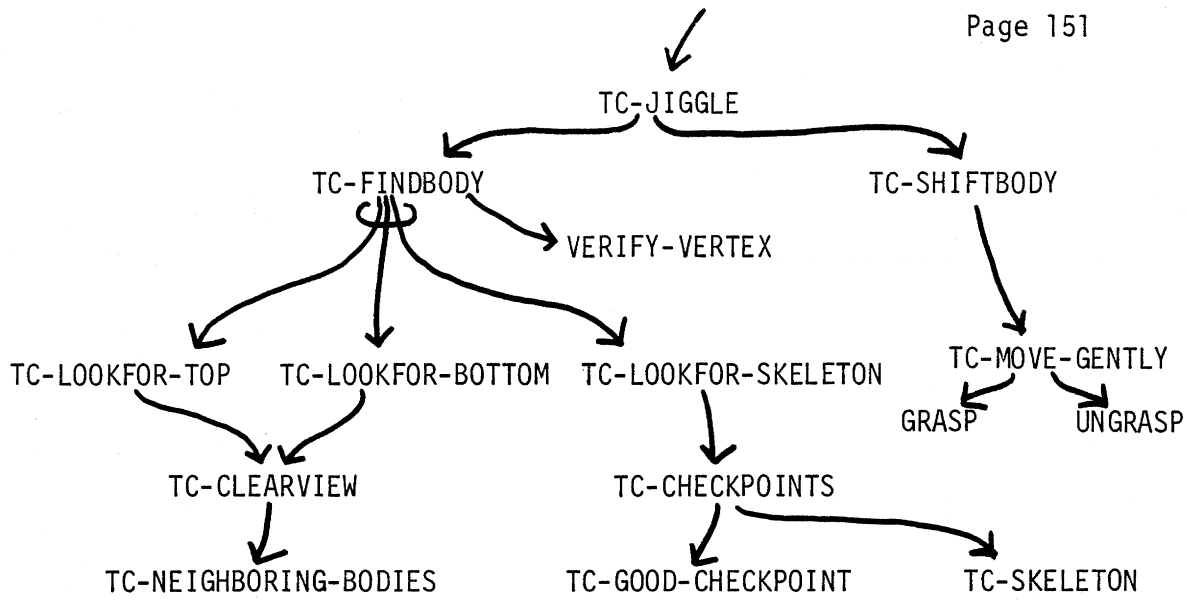


FIGURE 1

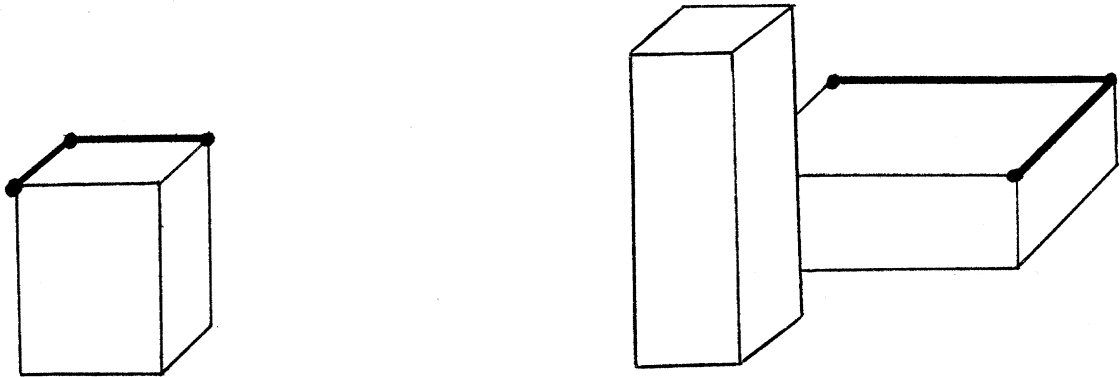


FIGURE 2

2. All bricks which could be adjacent to the one in question are either below it or have not yet been placed. The theorems lean toward conservatism in accepting vertices as good candidates to look for and will reject all of them in some cases.

One exciting possibility for further work is the incorporation of a model of the hand. With it we could adapt the system to avoid vertices occluded by it, doing away with the necessity to release the brick and withdraw the hand. This would result in a more dynamic and accurate feedback system.

OUR ROBOT HAS A HAND, TOO

Until now the vision system has made no use of its arm in getting information about the world. We now have a limited ability to reach out and touch in order to disambiguate some scenes, using a new arm primitive written by Jerry Lerman.

Sending (TOUCH X Y) to the arm causes it to position itself above the point (X,Y), slowly descend until it touches something, and report its final height. An optional third argument can specify a maximum height at which something is expected, allowing the arm to rapidly drop to this height and then more slowly feel its way downward.

A series of theorems have been written which actively use the arm as a sensor and other theorems have been taught to use them, resulting in the system network shown in figure 3. With these theorems we can now handle scenes such as the pedestal in figure 4. In this scene we can't determine the tallness of B1, since it could touch the bottom of B2 near the front, the back, or somewhere in between. As a result, we can't get the dimensions and location of B2 either.

We can however determine the location of B1 in the X-Y plane (through TC-FIND-LOCATION-BOTTOM). Moving the arm down over this spot until it touches the top of B2 gives the altitude of B2's top. With this information we can calculate the location and dimensions of both bricks.

Previously, when we wanted to find the location or

dimensions of a brick we had to find its altitude above the table. If it was not resting on the table, we had to find the dimensions of its supports, necessitating knowing their altitudes above the table, etc..... We recurse downward until we reach the table or fail by hitting a brick for which no tallness or altitude can be found. With these new theorems we have another alternative: recursing upward until we find a brick we can touch with the arm.

One problem is that we aren't working with a very good three dimensional model. TC-TOUCHTOP is the theorem which tries to touch the top of a brick. Checking first that there is nothing above the brick, it tries to touch it above the center of one of its supports. The brick could, however, not be above this spot (as in figure 4b) causing the arm to miss it. One precaution that TC-TOUCHTOP takes is calculating the minimum height to expect the top of the brick. If it touches something below this height, it assumes it missed.

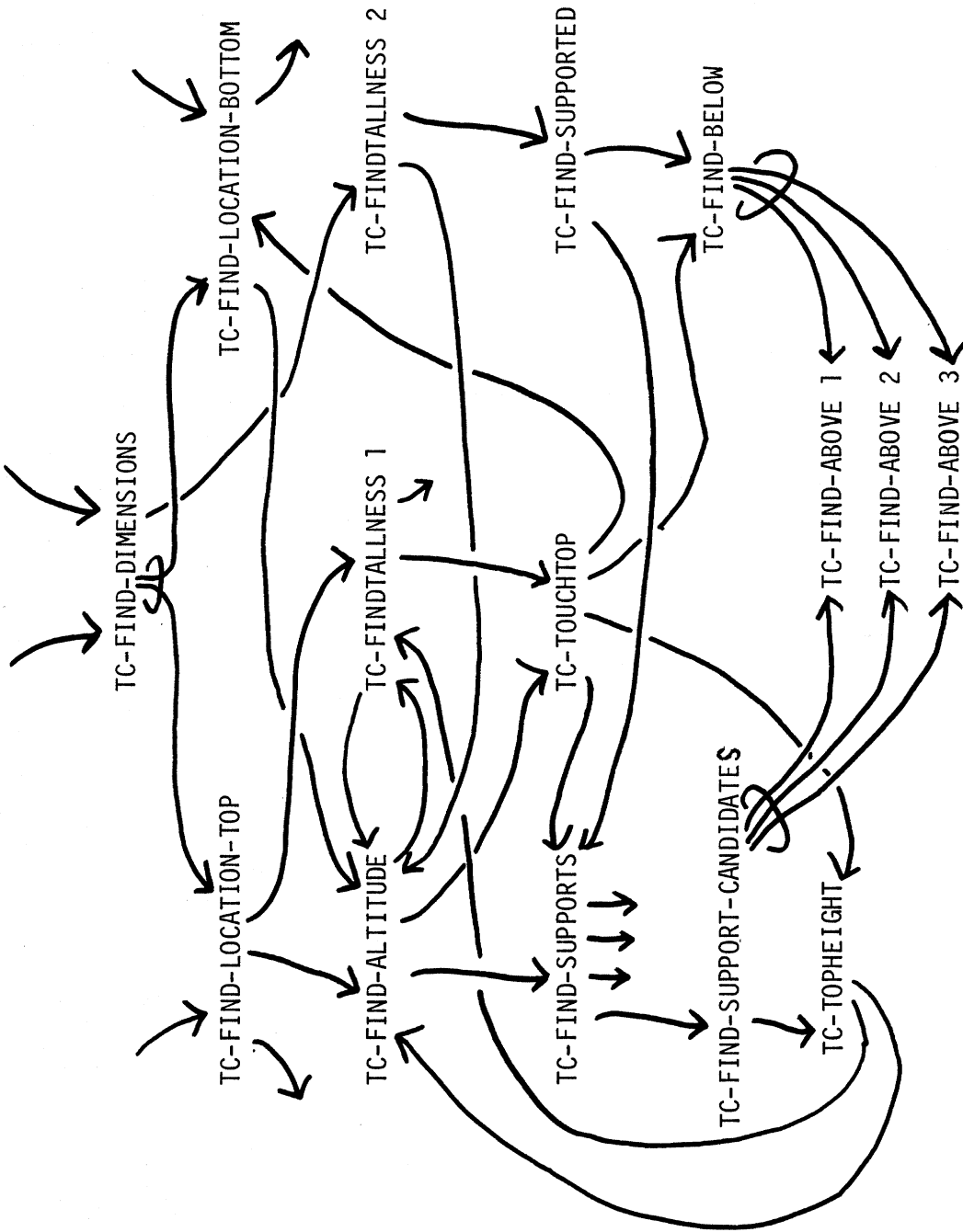
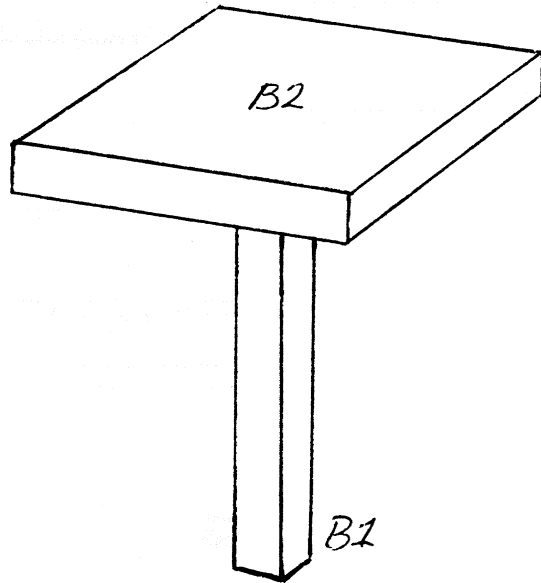
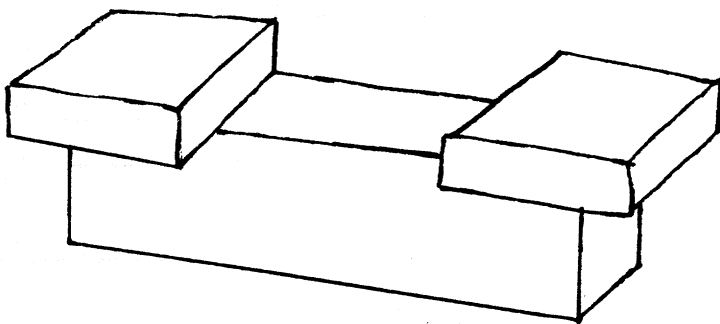


FIGURE 3



A

FIGURE 4



B

TC-FIND-SUPPORTS

TC-FIND-SUPPORTS and its related theorems have been modified to handle situations with which they previously could not cope. Figure 5 shows the new organization of this part of the system. The strategy of TC-FIND-SUPPORTS was to take each object below the brick in question (found through TC-FIND-ABOVE-1 & -2) as a support candidate. The altitude and tallness of each candidate were found and summed. The object or objects (if there were several with nearly equal combined altitude and tallness) with the largest sum were then taken as the actual supports. This sum was then asserted as the altitude of the supported brick. The theorem failed if it could not find an altitude or a tallness for one of the bricks below.

The new TC-FIND-SUPPORTS works in much the same way, but has been modified to handle many cases where the tallness or altitude of an support candidate can not be found. In such cases it determines the minimum height that the top of the candidate could have.

It will also yield useful information in cases where it is still ambiguous which objects support another. Before failing it makes assertions of the form:

(B1 may-be-supported-by B4)

(B1 may-be-supported-by B7)

(B1 has-minimum-altitude 4.12)

These assertions can later be used by other theorems with more

real world knowledge to clarify the scene. For example, we might call on a theorem which knows about stability or one which can recognize a table top and legs to decide who is doing the supporting.

Two auxilliary theorems are used, TC-ADD-TC-SUPPORTS-1 and -2, which contain some 3-D knowledge. TC-ADD-TC-SUPPORTS-1 looks for a marrys relation between the brick in question and a support candidate. If one is found, the theorem reports that it must be a support (assuming gravity and no glue). TC-ADD-TC-SUPPORTS-2 is explained below.

The capabilities of the new TC-FIND-SUPPORTS are best shown in the scenes in figure 5. For each of these scenes the old TC-FIND-SUPPORTS would simply fail, leaving no assertions in the data base. Figure 5e is particularly interesting, showing the application of some three dimensional reasoning. On this figure TC-FIND-SUPPORTS first calls TC-FIND-SUPPORT-CANDIDATES which reports that B2 and B3 are likely support candidates and that B1 must have an altitude of at least T. TC-ADD-TC-SUPPORTS-1 then finds that B2 marrys B1 along B1's bottom edge, implying that B2 must support B1 and that B1 has an altitude of T. TC-ADD-TC-SUPPORTS-2 is activated and notes that B1's altitude is now known to be T. Discovering that the minimum tallness of B3 is also T (within an epsilon) it asserts that B3 must also marry B1 and be a support.

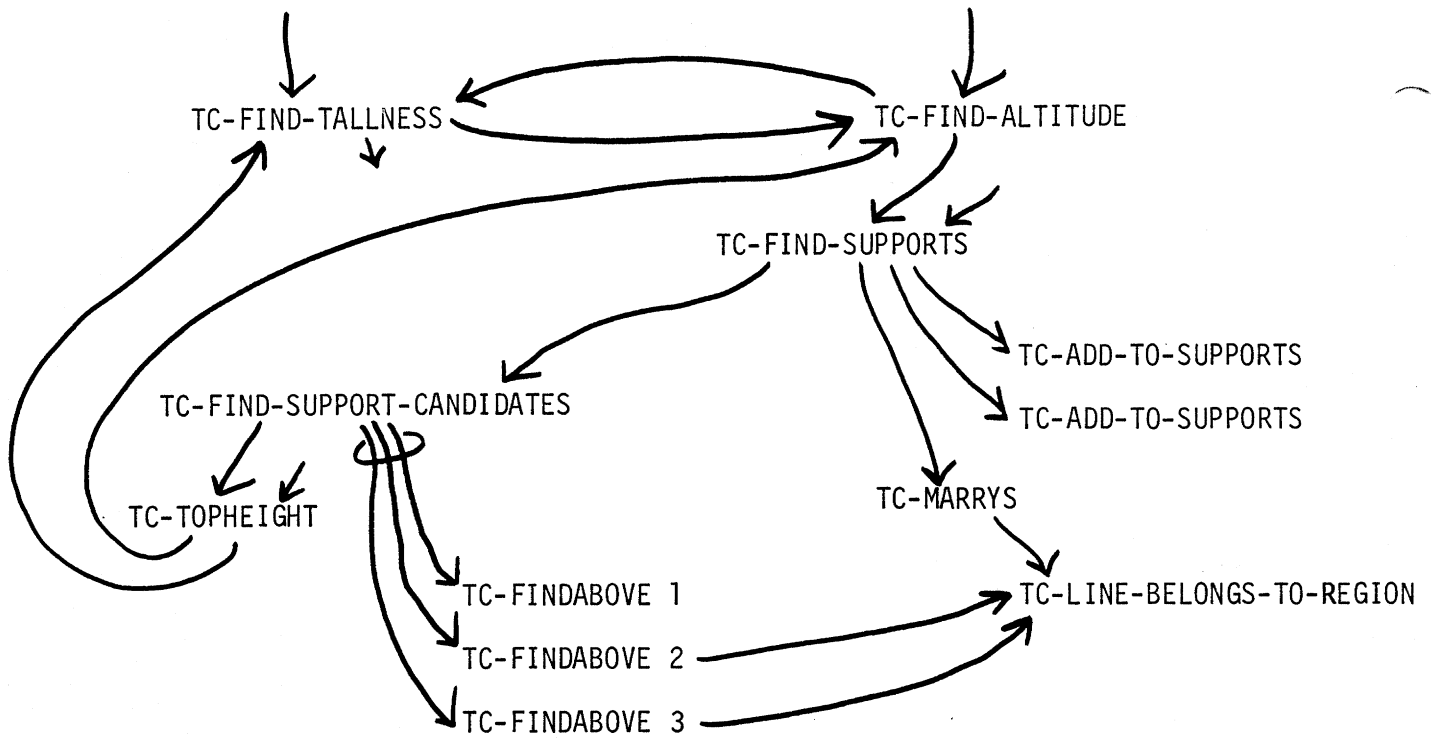
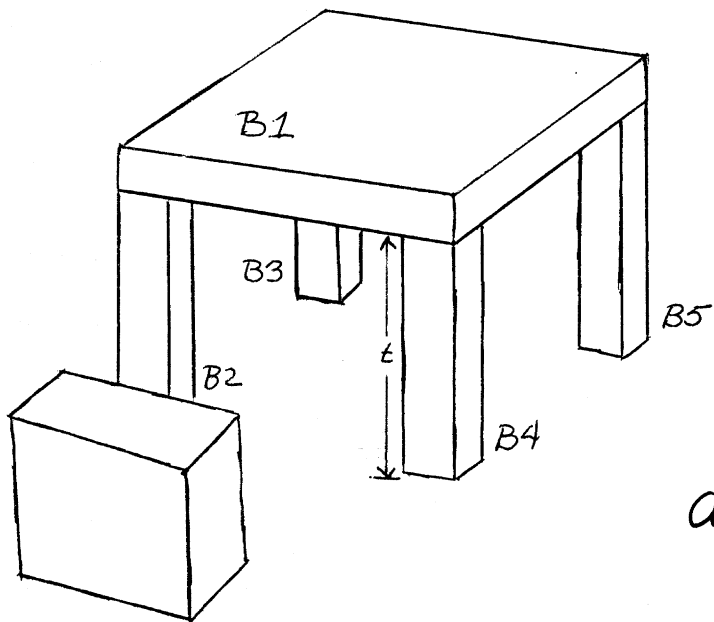


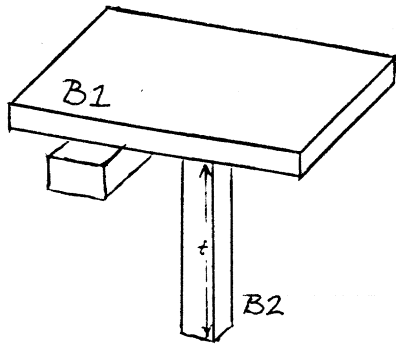
FIGURE 5



Assertions made by

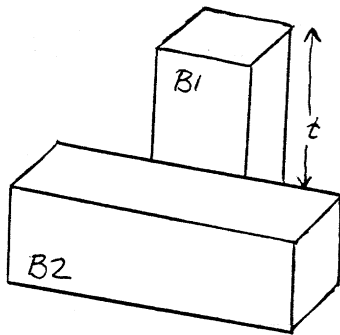
TC-FIND-SUPPORTS:

- (B1 is-supported-by B2)
- (B1 is-supported-by B4)
- (B1 is-supported-by B5)
- (B1 is-supported-by B3)
- (B1 has-altitude t)



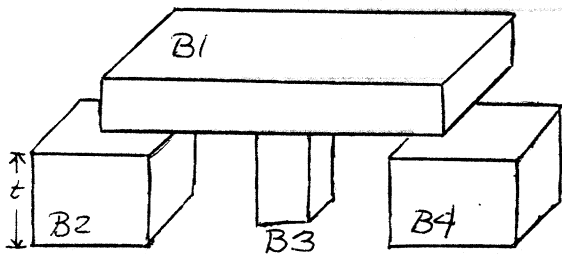
B

(B1 is-supported-by B2)
(B1 has-minimum-altitude t)



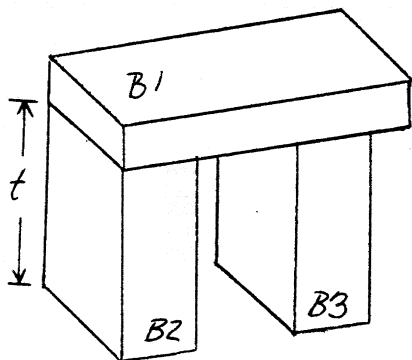
C

(B1 is-supported-by table)
(B1 has-minimum-altitude t)



D

(B1 may-be-supported-by B2)
(B1 may-be-supported-by B3)
(B1 may-be-supported-by B4)
(B1 has-minimum-altitude t)



E

(B1 is-supported-by B2)
(B1 is-supported-by B3)
(B1 has altitude t)

CHANGES TO TC-SKELETON

TC-SKELETON has been changed to return a little more information if it can't find a complete skeleton for a brick. When TC-SKELETON fails to find a line of a particular type it tries to find the longest line fragment of that type and makes partial-skeleton assertions of the form:

(B3 type-two * (V1 V2))

(B3 has-partial-skeleton V4 V1 * (V1 V2) V1 V14)

Figure 6 shows some examples of partial skeletons.

This partial skeleton information is used by other theorems which hypothesize what the rest of the brick may be like. From it we can get a handle on some three-dimensional information such as a brick's orientation, its minimum dimensions, and some idea of its location. Since other theorems make hypotheses that complete parts of the skeleton, an antecedent theorem has been added to keep the skeleton assertions up to date.

FIGURE 6

(B4 has-partial-skeleton * (V1 V2)*(V1 V3) V2 V4

(B1 has-partial-skeleton V1 V2 V1 V3 * (V3 V4))

'TC-LINE-BELONGS-TO-REGION

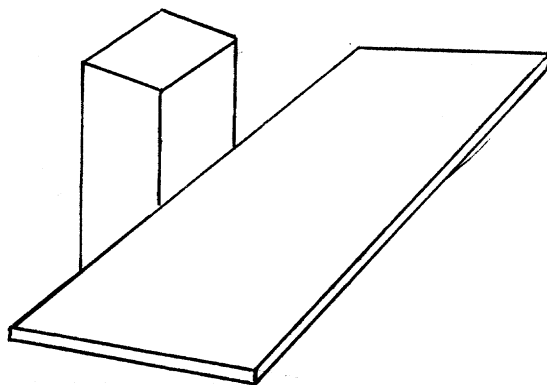
A new theorem exists which will determine whether a line is physically associated with a particular region in a picture. This question crops up in numerous places in the vision system: finding the skeleton of a brick, finding a brick's tallness, finding bottom lines, deciding whether a face of a brick is vertical or horizontal, etc. In each of these places several heuristics were employed to find lines which 'belonged' to a region. TC-LINE-BELONGS-TO-REGION is a collection of these heuristic and several new ones which can be called whenever needed. It makes assertions of the form:

(L-V1-V2 belongs-to R4)

One of the heuristics is that an interior line of a body belongs to both regions it bounds, so that TC-LINE-BELONGS-TO-REGION should not be used on self-occluding bodies. In general, the heuristics are applied conservatively, sometimes calling on the theorem TC-OCCLUSION-1 to look for supportive or contradictory evidence. Consequently some lines will not be recognized. Having this information at hand makes many tasks much easier. For example:

- * A region is a vertical face if there is a vertical line which belongs to it.
- * Two objects marry if we find a line which belongs to a region from each body.

M&M HACK



How many things can you find wrong with this picture?

In a picture such as the one above where one brick obscures the bottom of another, we can extract some information on the whereabouts and size of the obscured brick. Examine the scene in figure 7. B2 could be a very tall brick which was touching B1, or a shorter brick far behind B1, or anywhere in between. It's ambiguous. Knowing the range of possible heights and resulting locations of the obscured brick will be quite useful to other theorems which try to decide what the situation really is. To get quantitative three dimensional information we use the procedures described below.

To get the maximum tallness of B2 we assume that it is directly behind and touching B1. Assume for the moment that the ends of B2's three vertical edges (b, c & d) touch the edge a-e.

These three points would then have an altitude of h (the 3-D tallness of B1) from the table. From this we can get their 3-D coordinates and the resulting lengths of the three verticals. We then take the shortest of these lengths as the maximum tallness of B2. This corresponds to selecting the vertical ending in c as the only one which could touch P1.

To get the minimum tallness we assume that B2 is far behind B1 and its bottom edges just barely obscured. For the moment assume all three points are on the table. We get their 3-D tallness coordinates and calculate the lengths of the verticals. The maximum of these three lengths gives us a minimum tallness for B2. Taking the longest corresponds to selecting the point b as the only one which could actually be on the table.

The problem with the picture on the previous page is that, assuming the obscured object is a brick, its apparent minimum tallness is greater than its apparent maximum tallness.

A further refinement of this heuristic is shown in figure 8. In this picture, to get the minimum tallness of the obscured brick we assume that the points a and b rest not on the table, but on the regions R1 and R2 respectively. We must check of course, that these regions are not vertical, as in figure 8b.

The same situation exists for the inverted case, the pedestal in figure 9. If we assume that B1 supports B2, we can put upper and lower bounds on the height of B1 (and the resulting altitude of B2). We know that the top of B1 must touch the

bottom of B2 somewhere near the front, the back, or in between. Getting the minimum tallness is trivial, just measure the length of the three vertical edges of E1 and take the largest of these. This corresponds to a situation where B1 and B2 marry along their front edges. To get the maximum tallness we start off by locating the visible bottom edges of B2 and predicting where the others should be. We then extend each vertical until it intersects one of the back bottom edges of E2. After calculating the 3-D lengths of these verticals we take the shortest as being the upper bound on the height of B1.

Considering the stability of such structures would of course lead to more refined upper and lower bounds.

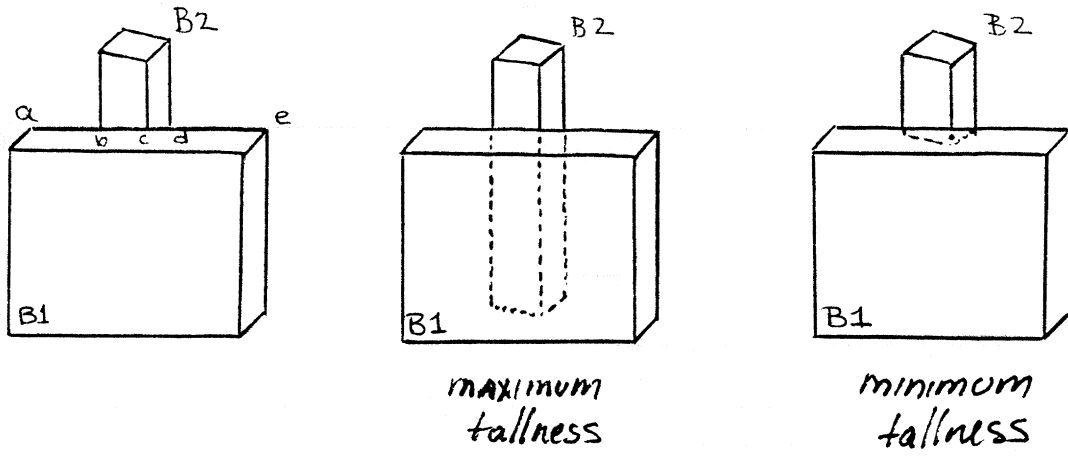


FIGURE 7

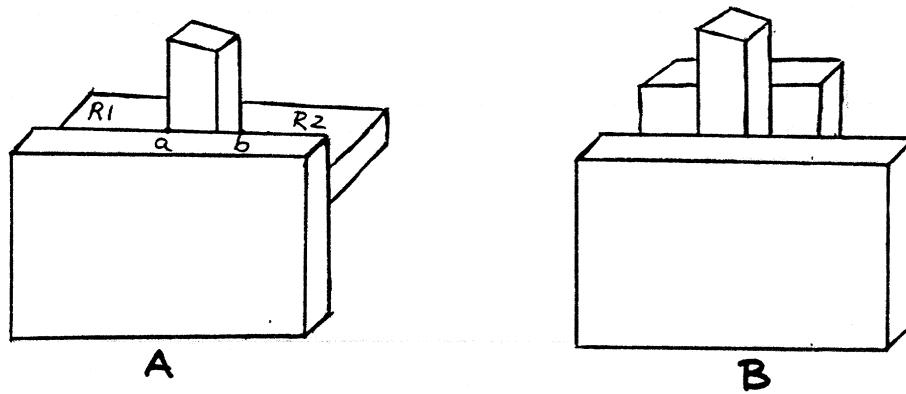


FIGURE 8

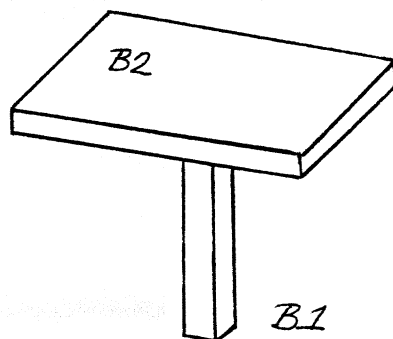


FIGURE 9

ONE KIND OF THREE-DIMENSIONAL REASONING

Everyone agrees that future advances in computer vision will come with the incorporation of 3D knowledge and reasoning. One type of 3D reasoning we could add is shown in figure 10. In figure 10a the average human viewer (trusting and non-cynical) sees two identical bricks supporting a third, even though B1 could be longer or shorter than B2. Because B1 and B2 serve the same function (supporting B3), have the same orientation, and as far as we can see, could be the same size, we easily assume that they are. Similarly, in figure 10b, we see B4 as being the same size as the other three standing bricks. Here the evidence is:

- * There is a preponderance of tall, thin standing bricks
- * B1, B2, and B3 form a row, which would include B4 if it were the same size
- * B1, B2, B3, and B4 all have the same orientation
- * B4 could be the same size as B1, B2 and B3

Humans do this hypothesizing and filling in of details to a great degree. It is an integral part of perception, as it would be impossibly costly (in time and effort) to try to disambiguate everything we see. Teaching the machine to do such hypothesizing would be a natural way to incorporate some three dimensional reasoning and to enable it to consider the global context of the scene.

A similar form of reasoning was pointed out by Winston in his thesis < Learning Structural Descriptions from Examples , AI TR-231 >. In figure 11, B2 appears to be a wedge, while we see B4 as a brick, even though they show the same arrangement of lines and faces. In both cases since the two objects form stacks and are exactly aligned we first assume that they are identical.

We might object that this presupposes an orderly scene , one which is carefully set up or contrived. However, the world we humans create, and which robots may inherit, is just such an orderly, contrived world. Even in the mini-world of plain white-faced blocks we use in our present research we tend to build little arches, stacks, and other structures containing identical, interrelated parts. In the larger world of human construction this orderliness is more apparent. We tend to build things which are symmetric and unsurprising in details. Complex objects such as buildings, electronic circuits, and cars are built using smaller identical parts (e.g. standard-sized windows, resistors, bolts). Who would suppose that the wheels on one side of a car are any different than those on the other? Long hallways usually have identically dimensioned doors uniformly spaced. The legs of a table or desk are nearly always the same.

We might also object that a robot would do better to spend his time trying to disambiguate a scene by removing some obscuring obstacles, by walking to one side, or by reaching out his hand and touching. In many cases however, our robot may find

it impossible to change his viewpoint or to interact with the scene. Even more importantly, the ability to do this sort of reasoning would allow him to have some expectation as to what will most probably be seen if obscuring objects are removed, or the viewpoint is changed. The robot can then quickly test his previously formed hypothesis. If this verification fails, he can flush the hypothesis and examine the scene more carefully.

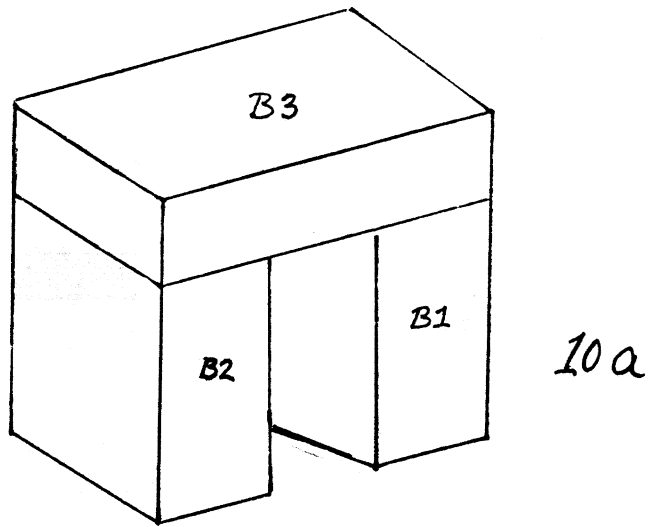


FIGURE 10

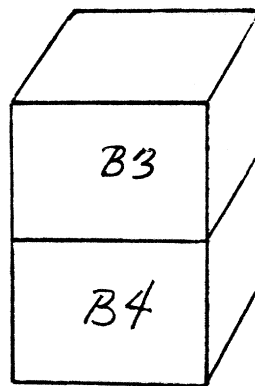
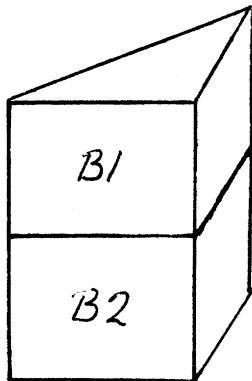
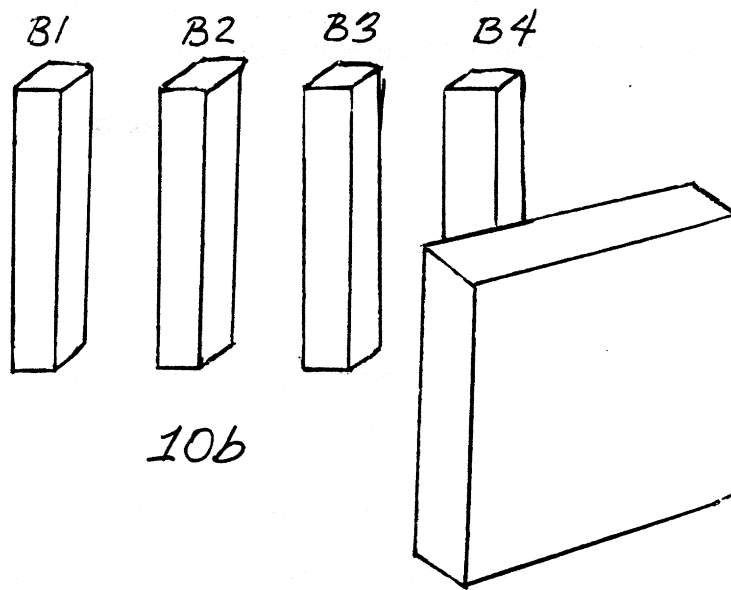


FIGURE 11

The following pages describe initial work in creating a system of theorems to do just this sort of reasoning. A skeletal system has been implemented which will handle a number of the simpler situations (such as those with fairly obvious group type relationships — rows, stacks tables, arches, etc.).

Whenever we can't find the complete dimensions of a badly obscured object, as in figures 10a and 10b, we check for evidence that this mystery object might be just like some other object in the scene. Typical of the evidence we look for are:

- * Chains of relations (rows, stacks, walls)
- * Identical relations to a third object, or functional similarities (e.g. supports of an arch, table legs)
- * A preponderance of identical objects in the scene
- * Exact alignment with another object
- * Other relations and properties such as attitude, nearness, placement, marrys, abuts, etc.

If we find such a candidate we check for contradictory evidence. Things we check are:

- * That the known dimensions of the obscured object agree with the corresponding dimensions in its supposed double.
- * The unknown dimension(s) lie within the

apparent maximum and minimum bounds
we have calculated.

- * That the hypothesis yields no colliding objects (i.e. those occupying common space).
- * That we do not contradict any previous assumptions we have made (e.g. support).

In some cases we need not require an object which might be an exact double, but only one which implies a similarity that might resolve the hidden dimensions. For example, in figure 12 we hypothesize that that the two stacked bricks have the same depth, even though their heights are different.

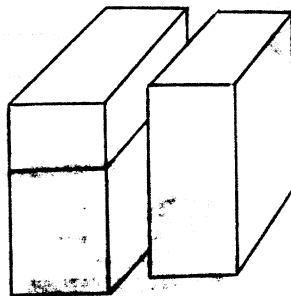


FIGURE 12

If we find any contradictory evidence, we reject the candidate and can look for another. If no contradictory evidence is found, then we make our calculations and tentatively assert the dimensions and location of the object given our hypothesis. This is done with pointers to the theorems which suggested this hypothesis so that we can reconstruct our reasoning if needed.

With this information we can proceed to analyze the scene

and hope that we have not been tricked. Alternatively we can try to verify the hypothesis by some other means. For instance, we could create a daemon theorem to lurk in the background waiting for some of the obscuring objects to be removed. When they are, the eye can be asked to verify critical lines or vertices suggested by our hypothesis. For our paradigm arch case, as soon as the top brick is removed, we can look for the top back vertices of the mystery brick. If the vertices aren't found near their proposed locations then the hypothesis is flushed. In such a case we could use the eye to more completely scan the area to resolve the problem. In other situations we might be able to use the hand as a sensor to verify the hypothesis or find out what has gone wrong.

RECOGNITION OF REAL OBJECTS

Eugene C. Freuder

October 1972

ABSTRACT

High level semantic knowledge will be employed in the development of a machine vision program flexible enough to deal with a class of "everyday objects" in varied environments.

This report is in the nature of a thesis proposal for future work.

This paper was originally published as Vision Flash 33.

1944

1945

The following table shows the number of persons who were employed in the various occupations in the United States in 1944 and 1945.

Occupation 1944 1945

Professional occupations 1,200,000 1,300,000

Foreword

This material is not presented as a discovery but rather as a journey on the path laid down by Professors Minsky, Papert and Winston in Vision. Please ignore any dogmatic tone which may appear in the efforts to overcome my natural tendency to the opposite "I think, perhaps, maybe..." school of rhetoric.

Abstract

A proposed program of research in Machine Vision is described. The first section, the Scerario, lays out succinctly and informally the aims of the project and summarizes the background of the work, the problems to be faced and the approach to be taken. An extended description of the Problem follows with its background and its formulation for the present work. We next discuss the Artificial Intelligence Issues involved, and the Approach we will take. Benchmarks are laid down for specific progress. Our research interests are once again carefully formulated in the Summary.

Scenario

Picture the following scenario.

We walk into the street, and hijack the first passerby. We have him go home, retrieve his toolbox and return to the AI Robot. He grabs a handful of tools and dumps them in front of the eye. The Robot identifies these objects.

The key element here is flexibility in dealing with real objects in real situations (surreal perhaps in this instance).

The flexibility to deal with reality will be the basic aim of our research.

Why has previous work in visual recognition often been sadly lacking in this regard? Why does this effort have a better chance of success?

Consider first some recent contributions.

1. Minsky and Papert—The heterarchical approach essential to the flexibility sought (4,5,6,7).

2. Winston—Network structure, data driven control, basic structural description, learning by failure, grouping, quantitative analysis of relational concepts (11,12,13).

3. Shirai—A.I., and vision research in particular, as an experimental science, as demonstrated most recently by Shirai(8).

4. Waltz—The extent to which completeness of description can replace flexibility of control (9).

5. Charniak—The role and extent of knowledge in

understanding (1).

6. Winograd—Semantic/syntactic integration. Frontal attack on global problem; large system structure. Visual/linguistic analogies (10).

7. Hewitt—The power of a problem oriented control structure. The ease of a pattern directed data base. Two way flow of control; flexibility of non destructive failure (3).

As a framework for these developments, we have the Vision System. This has provided the experience and facilities for a broad based background in vision research plus the specific technical and organizational results of the System, itself (Complemented by the more formal seminar structure).

Perhaps most importantly we want to deal specifically with the problem suggested in the first paragraph. This is very different from most previous efforts at visual recognition.

Often the descriptive problem has been dealt with separately as a pattern recognition issue. When real visual input has been used, the aim has been generally to transform this input, with a single low level predicate, into a "preprocessed" data base. Features could then be extracted from this data base and dealt with as a pattern recognition or tree search problem.

In fact, we maintain that there is no real problem, on the macroscopic level, in distinguishing a respectable set of real objects. In some respects, the more complex and unusual the objects the better. It should not be at all difficult to

"separate these objects in parameter space" with any number of redundantly distinguishing features.

The problem is to get some of these gross features:

- a. from amid all this redundant mass of information
- b. in particular, from the mass of "meaningless" low level data
- c. in differing:
 1. individual samples of objects (variations in size, shape, orientation, texture, etc.)
 2. scene contexts (occlusions, shadows, etc.).

On a local level these changing conditions create great differences. If we deal only from the "bottom up" we cannot hope to deal with such chaotic variation.

However, with higher level interaction we can hope for global guidance at levels that can deal with such variation. The old saw: "it's easier to find something if you know what you're looking for". We need to know what we are looking for to find the features.

I think my goals can be approached in two stages.

- 1) First I will pick a single object of some semantic richness, e.g. a hammer. I will then write the world's greatest hammer recognizer. This is not as trivial as it may sound.

The program will needlessly to say not merely reply "hammer" when faced with any object, on the grounds that hammer is the

only object it knows about. It will have to decide whether the object is a hammer or not a hammer.

It should be able to make this decision about your roommate's favorite hammer (or screwdriver). It should be able to decide after a stranger is allowed to place the object in the field of view and arrange the lights. It should decide correctly when visiting Cub Scouts come in and are allowed to empty their pockets over and around the object to be studied.

This is a hard problem.

2) Assuming Part I is not a thesis in itself, the next stage will be to integrate knowledge of several objects into a system. Hopefully the experience of part I will provide principles for encoding visual knowledge that will facilitate adding the initial ability to deal with several new objects.

We will want to design the analytical structure to take advantage of the hypothesis and verification techniques developed in Part I. Note that the problem domain of Part I encourages us to explore the practical implications of our philosophical bias: application of specific higher level knowledge.

Preceding page blank

Problem

Machine Vision involves both "low level" image processing and "high level" descriptive analysis. The failure of Machine Vision to date has been in adequately linking these levels.

Much of the work in this field has restricted itself rather strongly to one or the other of these levels. There are several reasons for this. In the beginning there was a tendency for image processing people to believe they could "do it all" with a clever enough set of Fourier Transforms. The descriptive analysts, on the other hand, tended to underestimate the problem of preprocessing a suitable data base for their descriptive schemes.

Problems in each of these areas have been formidable enough to perhaps demand single-minded concentration. But now that a groundwork has been laid we are in a position to take a more comprehensive approach. It has in fact become clear that such an approach is mandatory for continued progress in Machine Vision (4-7, 11-13). The pressures for thesis work at a single level of the problem have begun to lead away from the overall goal. Search for the most sensitive line finder sidesteps the natural context in which the problem is to decide which lines to ignore, not which obscure lines to find. Yet work continues on idealized "picture grammars" which assume the preprocessor has given them just the data base they need (without any knowledge of those

needs).

Part of the reason that work on these two aspects of the vision problem has often remained unnaturally isolated must be the distinction in interests and skills required for work in the two fields. We therefore think it might be advantageous for two students to pool their experience in these separate areas. In any event a "pincer" attack is called for specifically on the interface between high level and low level analysis.

This does not mean regarding present accomplishments in these areas as "black boxes" and wiring them together. Rather, processes at each level must be organized with the requisite cooperation in mind.

Seeing at any level beyond the feature point is a matter of organizing visual information according to some visual model of the world. In the past we arbitrarily used only the most elemental physical units of the model to process the visual data--the most basic physical regularities and anomalies such as lines and points. The rest of the higher level knowledge was stored separately to be "matched" against data that could hopefully be organized into meaningful structures without knowing which structures were meaningful. This knowledge must also be part of the visual machinery that organizes the raw visual input and thus "sees". Instead this information has been so independent of any inherently visual process that researchers have been able to point proudly to their ability to perceive and program this

"recognition" apparatus as instances of highly general abstract mathematical models.

In point of fact, higher level context assumptions (a parallelepiped environment, for example) have long been implicit, or even explicit, in much of our work. Experience has shown the utter necessity for such guidance in order to make sense out of a chaotic sensory reality. It is time to fully integrate our higher level processing into the organization of our visual data.

We have seen how difficult it is to obtain even a line drawing of a cube without suspecting that we are looking at a cube. How can we expect to obtain some idealized data representation of a horse and then recognize that representation as a horse. We cannot choose the ideal descriptive model of a horse analytically and expect some syntactic predicate to produce this description for us.

The conclusion we come to is that there is a need to eliminate the old distinctions between preprocessing, description and recognition. One cannot obtain a preprocessed result which is then analyzed to produce a description which is then matched with a model to be recognized. Rather seeing is a homogenized process, the "model" is built into the structure of the processor, the description of a data set is the protocol of its processing. Recognition is coincident with description.

Recognition is not a matching of templates against a processed data structure. Rather it is a many layered silk

screen process that begins with the raw input and leads to the richly patterned perceptual world we seek.

Issues

We feel that an approach which is predicated on the intimate interaction of descriptive apparatus and description predicates is the best hope for practical progress in Machine Vision. Beyond that we believe that such a project would provide an appropriate laboratory for the illumination of several current A.I. conceptual concerns.

1. Heterarchy

Problems in reconciling low level results with high level idealizations, even in a very simplified domain, provided one of the early motivations for the heterarchy concept (4-7,12). Return of recognition level advice to the preprocessor is perhaps the most salient example of heterarchy practice (4). It is significant that, except on a most general level, even this case study of heterarchy has not been implemented and its implications certainly not fully explored.

Our investigation of heterarchy in vision would emphasize the crucial role of recognition level knowledge.

We do not analyze heterarchy in terms of interaction or advice between major discrete processing modules or stages. We do not view "high level" knowledge as criticizing the descriptive structure resulting from low level "preprocessing". Rather we believe that in general no useful structure can be derived without high level knowledge directly involved in the

construction process from the beginning.

In any case, a working fully ramified case study should provide useful insight into the theory and practice of heterarchical interaction. In particular:

2. Sensory/perceptual systems

The heterarchical interaction between low level sensory systems and high level perceptual systems is of particular interest to A.I. research at present. Having worked "down" from chess-playing and integration, A.I. is now facing the far more obscure problem of enabling machines to duplicate the essential "automatic" processing of real world sensory data. Techniques and processes abstracted from our results in melding low level and high level visual processing should prove relevant when we provide our robot with other sensory apparatus. The possibility of higher level semantic intervention in the auditory analysis of speech, for example, is already a live issue.

3. Knowledge—"the size of infinity"

In developing a system that can deal with in a real world context we will face certain problems analogous to those faced by Charniak in his work on understanding childrens' stories. Previous successes in machine vision, or even machine pseudo-vision, i.e. analyses of hypothetical input, have dealt largely with a severely restricted domain. The recognition set has been either highly stylized or simply very small. There is an analogy here to syntactic uni-directional theorem-proving systems, which

break down on a non-restricted data base. First we will have to get a practical idea of what the size of infinity is in the visual domain, and then we will need techniques for organizing it. In particular:

4. Knowledge—as procedures

Our knowledge—about shapes as well as subjects—will be organized as procedures. This is more than a fashionable device in a system where knowledge will often consist of knowing the right questions to ask of some other module, preprocessing modules in particular. ("Preprocessing" is obviously a misnomer here, reflecting an organizational mode we hope to replace.)

Recognition will not consist of preparing a description of an input scene and matching it with a description of a model. Rather description and recognition will occur simultaneously during the processing of the scene. A description will be essentially the protocol of a process, not a result.

5. Parallel processing

If parallel processing techniques are to prove significant, this would certainly seem to be an area in which they might demonstrate their value. We envision a system in which processing would proceed simultaneously on visual subunits or on conceptual subprocesses. Proceedings would be dependent continually on the results of intermodule communications and questions, both as to the results being attained in different "higher" level investigations, and the results of additional low

level processing. It would seem plausible that dialogues of the following style could take place profitably:

a: I think these four things are legs. Is anybody looking at anything above them all?

b: Yes, I am.

a: Can you tell me if it's planar or bumpy?

b: Not yet.

a: Well, look, work on it; take your time and wake us when you have something.

a: (Yawn) Wazzat? Bumps huh? Probably an animal. We'll activate hoof, paw and knee searches. You send your stuff to an animal body parser.

c: I can't find any subdivisions on this thing for head or tail. Can anybody make head or tail out of any nearby blobs of relative size x and position n ?

.
. .

This particular dialogue may be vacuous. Investigation of many possible dialogues may reveal some essential usefulness of this type of organization.

6. Grouping

As even the brief dialogue above indicated, grouping problems will be central. Guzman-like techniques will not suffice, particularly for higher level organization. Rather than

very sensitive local predicates capable of distinguishing fine variations, we will require broader based cruder predicates capable of determining relative homogeneity. Contextual, conceptual cues will be needed to distinguish overlapping objects from functional groups. A program that knows about collars will not be dismayed to find a dog's head separated from its body; it will be capable of seeking the "severed head". Alternative hypotheses will be explored for conjoining parts of the picture, ignoring occlusions, or melding sequences into single units. The same subscene may be viewed in each of these ways: as linked units, overlapping distinct units, single textured units. Successful interpretation will direct the choice.

7. Organization

Direction will be available from both the lower level and the upper level. That is, knowledge about a shape as well as an object or class of objects, will be embedded in procedures which will in turn direct the flow of processing through other procedures. This flow will not be locked into any piece-by-piece or level-by-level decision tree or network search. The system can skip to high level hypotheses, plan an approach on that basis, fail, review what it found in the process and use this as a basis for some more "from the ground up" investigation. Details can be verified on a hoof before during or after the search for a head; whatever is expedient.

Approach

We expect our progress on this project to reflect the structure of the results. That is, both will proceed by a process of "successive approximation".

Professor Papert has characterized one approach to problem solving as "neglect all complication, try something". The results may serve as a "plan" on which to base further refinements or a basis for "debugging".

There is a distinction between simplifying the problem and simplifying the solution. One technique involves splitting the problem into a number of separate or successive simplified problems. For example: ignore shadows, texture, etc., and just consider ideally lighted ideal planes, then consider shadows, then consider texture, but no shadows, Etc. This approach can be very fruitful. However, it may leave us far from a solution to our total problem, particularly if the various aspects of the problem are related in a non-linear fashion. In this case it may prove necessary at times to approach the total problem with a simplified solution. We can then rework this solution successively to approach a more adequate solution. The intermediate results serve to indicate directions for improvement. We may "project" structural aspects of the problem from the solution, rather than relying on an a priori division of the problem.

We expect a continuing dialogue during the research that reflects and suggests the dialogue that will be built into the system:

>Is that side straight?

>I can't tell; it peters out in spots.

>Oh really? Maybe they're lined up wrinkles. Are there more of them?

>Could be.

>Can you characterize their profile?

>Yes, I can give you a "possible wrinkle" predicate; but that predicate will also work on soft corners.

>But the surrounding planes for a wrinkle will be similar?

>Yes, and aligned shadows often occur with wrinkles.

>Now can you ...

>No but I can ...

>Well then if I tell you ..., then could you ...

>...

The important thing is that many of the technical realities of the system will flow from real experience with the problems of processing real data. This does not mean that the results will lack theoretical content, but that the theory will bear a valid relationship to the reality of the vision problem. We want the theory to flow from and reflect the structure of visual experience, rather than attempt to distort visual data to conform to preconceived and inappropriate analytic theories.

Benchmarks

We might proceed in several phases. A possible sequence would be as follows.

As a first stage exercise, we could consider the development of a system that dealt with simple geometric models, e.g. cube, cylinder. This system would provide experience in melding high level knowledge with a suitably varied set of low level predicates. The idea is not to push any one predicate or approach to its limit but to allow the extent of higher level knowledge and variety of low level approaches to work back and forth, to zero in on the correct perception.

A brief example is a simple cube. We avoid working very hard to find the precise edges from a feature point analysis. Rather we obtain some rough regions with a crude homogeneity predicate (that averages out not only surface noise but textures). We massage these a bit and get a rough idea of their shape, if we suspect a parallelepiped, we may apply a finer test to verify this shape. We find enough to guess a cube or at least a planar object. We then apply a crude line verifier over the wide band between regions. Meeting success we may home in on the lines with a sensitive line verifier in a narrow region. (If some regions were lumped into one at the start we use our models to guide predicates in a parsing attempt (4).)

This example already illustrates several interesting points.

Unlike a pass oriented system, we do not have to apply all our predicates at once, but only as (and where) needed. We make use of broad based, region oriented predicates to avoid the problems of high sensitivity until we have some hypotheses to guide the search for finer features. By working down from the general to the specific we avoid losing the forest for the weeds. And even at the lowest level we are still dealing with a broad based predicate, a line verifier. We also avoid an initial commitment to a world model of straight line planar objects.

The next stage would provide experience in dealing with a complex of surfaces in non linear terms. A limited set of "real" objects, a set of tools for example, would provide a miniworld. The second stage system would be able to function in this world. Here we would have to incorporate a greater variety of data types and predicates or procedures.

Another simple example is a hammer. We find an initial region. We suspect and verify a roughly rectangular shape. Relative length versus width prompts a handle hypothesis (or vice versa). This initiates searches for regions at either end, either crosswise or colinear to the handle axis (we could have a screwdriver). We find a chunky region crosswise at one end. We hypothesize a hammer with handle and head. We move back down to verify a few fine details to assure our success.

Notice here some further principles emerging. Precisely what is required to see a hammer head varies depending on where

or from what direction you "enter" the observation. Think, for example, how carefully you would have to draw a hammer head to make it identifiable in the following contexts: at the end of a roughly drawn hammer handle, standing alone by itself, at the end of a carefully drawn hammer handle, in a picture of the head alone to be viewed only partially and in isolated pieces. The requirements are different if we move "down" from a knowledge, or hypothesis, of "something at the end of a hammer", or if we move "up" from a detailed picture of the contour of a piece of the head. Generally we have a range of redundant information that specifies an object; any particular set of details, e.g. of shape, are not always required for identification, and may be easier to verify than to obtain by "preprocessing".

A flexibly programmed "understanding" of hammer heads should be able to be "entered" at several levels. Processing should proceed in parallel, with mutual calls, modified by the current state of knowledge.

It may be appropriate here to initiate a Charniak-type study of "everything we need to know" about, e.g. hammers, in order to perceive them in varied contexts. Why should such extensive knowledge not be necessary to "understand" in visual contexts, just as it is required in verbal contexts?

Another miniworld would be established to provide experience in dealing with classes of objects. A set of saws, for example, or a set of doll house furniture, could form such a class. The

third stage effort would provide a system that could move easily through the perceptual space of this class.

Some of Winograd's ideas on organizing knowledge might be useful here. A systemic grammar might be a useful metaphor for at least one aspect of the organization. Also some convenient method of inputting knowledge would be useful at this point. Ideally the system would be such that someone could eventually hook up the output of a Winston learning program to the input of this system. (And the output of this system to the input of a Winston learning program, of course.)

Finally, a rich miniworld would be chosen to combine our previous experience in a more varied environment. A cutaway doll house, or a multipurpose workbench, for example. By the completion of this last stage our general principles for an integrated perceptual system should have been demonstrated, and new principles for implementing this integration should have emerged.

This is rather an ambitious program. We could only hope to lay the groundwork really, to build an instructive testimonial to the possibilities derivable from a proper conjunction of high and low level knowledge.

Summary

In summary, in order to make a quantum jump in vision research, a number of bold steps are required. The thinking of pre-eminent theoreticians in the field has long tried to push us in these directions. However, "practical" considerations have too long held us back.

Instead of studying the parts and then deriving the "glue", we must study the glue and then derive the parts.

It has for some time now been recognized that the real problems in vision lie in understanding the cooperation of the various subprocesses. In practice, it has been easier to define and deal with specific pieces of the spectrum of visual knowledge. However, we eventually end up sitting around bemoaning the difficulty of putting the pieces together.

The solution is not simply to learn more about more pieces. Neither is it to treat the pieces we have as "black boxes" to be tied together. Aside from the theoretical arguments that could be mounted against this approach, it has proven rather barren so far in practice. To learn something about the mysterious "binding energies" as it were, we must simply grit our teeth and attack the problem directly. This approach will often mean messy, tentative, ad hoc progress. We must be willing to pay that price. We may utilize our hard won piecemeal knowledge, of course. However, when we have finally won our way to some

understanding of the interactive process of vision, we may find ourselves in a better position to identify the essential subprocesses involved.

Instead of generating data types and predicates to handle an idealized data base we must generate idealized data types and predicates to handle a real data base.

Becoming absolute experts on line drawings will only qualify us as experts in graph theory. The humor of this approach is evident when we consider that "real" physical data is manufactured at great cost and effort to match this idealized data base, and even then the predicates derived for the line drawing model cannot succeed in producing a satisfactory translation of the physical objects into line drawing data types.

Our line drawing studies have provided us with some useful methodologies and results that should provide guidance and submodules for a more general study. However it is time to return for inspiration and guidance to more realistic data: not to set up another panacea predicate, but to extract whatever information is required to organize the sensory input; not to organize simply and dogmatically as line or corners, but as recognizable perceptions, as is appropriate for the data.

Most fundamentally, the easy but artificial distinction between seeing, describing, and recognizing must be broken down. We cannot merely organize the visual input according to some "neutral" data base. We must not partition out part of our

knowledge to function as a template to "match" our results. The only way we can hope to organize the visual data to "match" a high level form is to use that high level knowledge to perform the organization. The concept of a "model" as such is outdated. The medium is the message. Recognition is the process of description. Description is the process of recognition.

Bibliography

AIMIT = Artificial Intelligence Laboratory, M.I.T.

- 1) Charniak, E., "Toward a Model of Children's Story Comprehension," D.Sc. Dissertation, Department of Electrical Engineering, M.I.T., August, 1972.
- 2) Freuder, E., "Views on Vision," Vision Flash 5, AIMIT, February, 1971.
- 3) Hewitt, C., "Description and Theoretical Analysis (Using Schemata) of Planner: A Language for Proving Theorems and Manipulating Models in a Robot," A.I. Technical Report 258, AIMIT, April, 1972.
- 4) Minsky, M. and Papert, S., "Status Report II: Research on Intelligent Automata," Project Mac, M.I.T., 1967. (Abstracted in "Progress Report IV" Project MAC, M.I.T., 1967.)
- 5) Minsky, M. and Papert S., "Proposal to ARPA for Research on Artificial Intelligence at MIT 1970-1971," A.I. Memo. 185, AIMIT, December, 1970.
- 6) Minsky, M. and Papert S., "1968-1969 Progress Report, A.I. Memo. 200, AIMIT.
- 7) Minsky M. and Papert S., "Progress Report," A.I. Memo. 252, AIMIT, January, 1972.
- 8) Shirai, Y., "A Heterarchical Program for Recognition of Polyhedra," A.I. Memo. 263, AIMIT, June, 1972.
- 9) Waltz, D., "Generating Semantic Descriptions from Drawings of

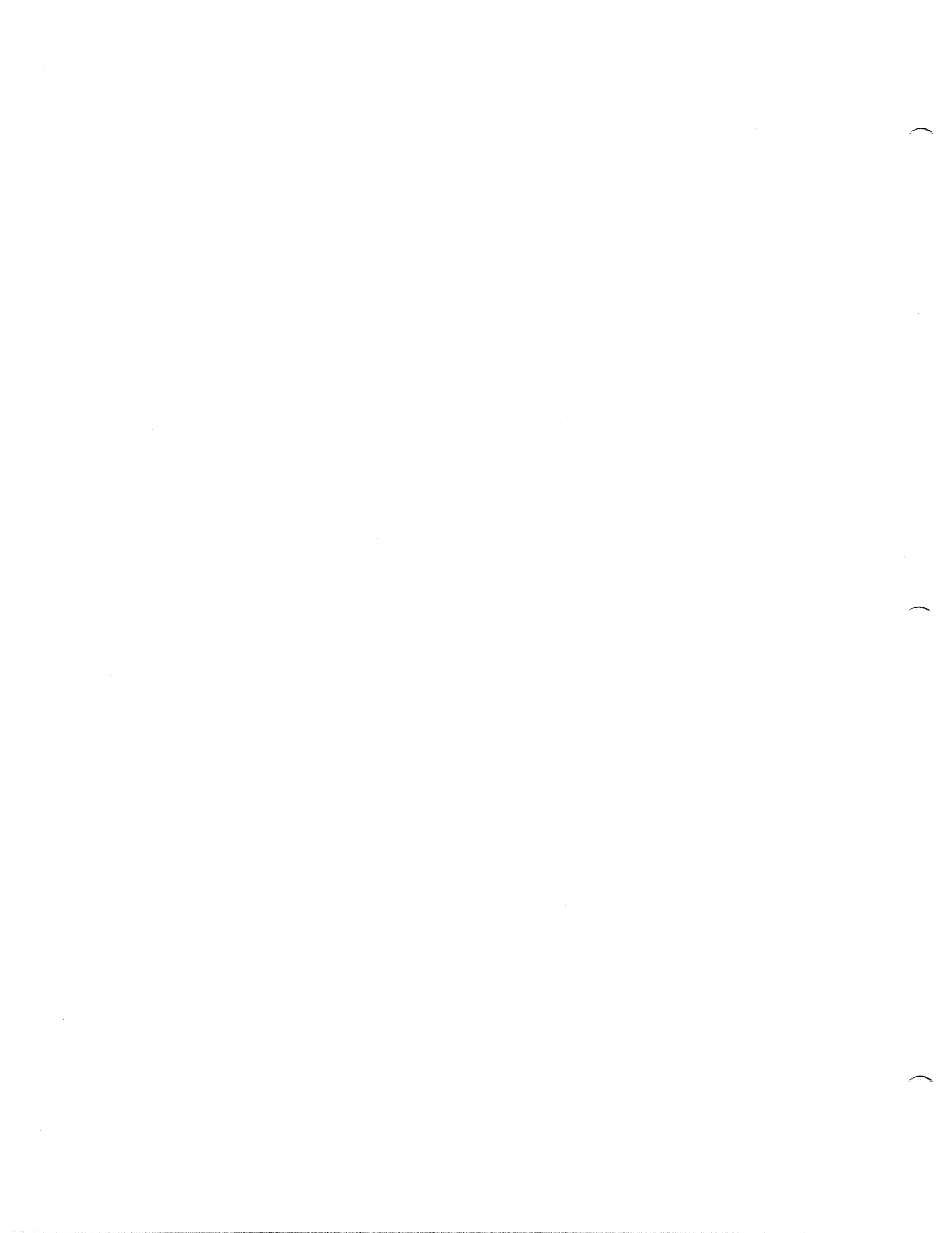
Scenes with Shadows," Ph.D. Dissertation, Department of Electrical Engineering, M.I.T., September, 1972.

10) Winograd, T., "Procedures as a Representation for Data in a Computer Program for Understanding Natural Language," A.I. Technical Report 235, AIMIT, February, 1971.

11) Winston, P. H., "Learning Structural Descriptions from Examples," A.I. Technical Report 231, AIMIT, September, 1970.

12) Winston, P. H., "Heterarchy in the M.I.T. Robot," Vision Flash 8, AIMIT.

13) Winston, P. H., "Summary of Selected Vision Topics," Vision Flash 30, AIMIT, July, 1972.



A HETERARCHICAL PROGRAM FOR RECOGNITION OF POLYHEDRA

by Yoshiaki SHIRAI

ABSTRACT

Recognition of polyhedra by a heterarchical program is presented. The program is based on the strategy of recognizing objects step by step, at each time making use of the previous results. At each stage, the most obvious and simple assumption is made and the assumption is tested. To find a line segment, a range of search is proposed. Once a line segment is found, more of the line is determined by tracking along it. Whenever a new fact is found, the program tries to reinterpret the scene taking the obtained information into consideration. Results of the experiment using an image dissector are satisfactory for scenes containing a few blocks and wedges. Some limitations of the present program and proposals for future developments are described.

1. INTRODUCTION

We do not know how to make a program to recognize objects visually as well as a human being. One of the shortcomings of many computer programs is, as Minsky has pointed out, their hierarchical structure. A human may recognize objects in the context of the environment. The environment may be recognized based on his a priori knowledge. The recognition procedure is, however, well programmed so that the simple obvious parts are recognized first and the recognition proceeds to the more complicated details based on the previous results.

The work in this paper studies an example of a heterarchical program to recognize polyhedra with an image dissector. Most previous works begin by trying to find feature points in a entire scene and make a complete line drawing. It is very difficult to get a complete line drawing without knowledge about a scene. If the line drawing has some errors, the recognition by a theory based on the assumption of the complete line drawing such as Guzman's might make still more serious mistakes. Our work is an attempt to recognize objects step by step, at each time making use of the previous results.

We assume in this paper that the difference in brightness between objects and the background is large enough to detect the boundary approximately. At present, this program works for recognizing moderately complicated configurations of blocks and

wedges. The limitations and proposals for future development are described later.

2. GENERAL STRATEGY

2.1 Priority Of Processing

For convenience, we set up the edges of the objects in a scene as falling into 3 classes. A line formed at the boundary between the bodies and the outer background is a contour line of the bodies. In Fig.1, lines AB, BC, CD, DE, EF, FG, GH, HI, IJ, JK, KL, LM, MN, NO, AO, VW, WX, XY, YZ and ZV are contour lines. A boundary line is a line on the border of an object. Contour lines are boundary lines. In Fig.1, the boundary lines are the contour lines and lines on the boundary between two bodies, i.e. CP, PH, IQ, QR, and RM. An internal line occurs at the intersection of two planes of the same body. Lines JS, LS, QS, PT, NT, AT, FU, GU, DU and XV are internal lines.

The global strategy is shown in Fig.2. At first, the contour lines are extracted (because we assume a priori enough contrast between the objects and the background). If more than one contour is found, as in Fig.1, one contour for bodies B1, B2, B3 and another for body B4, then the boundary lines and internal lines are searched one by one for each contour. The global strategy in block 2 in Fig.2 is as follows.

- A) Find boundary lines before finding internal lines because boundary lines often give good cues to guess internal lines. Note that to find boundary lines implies to find bodies.
- B) In searching for lines, different situations require

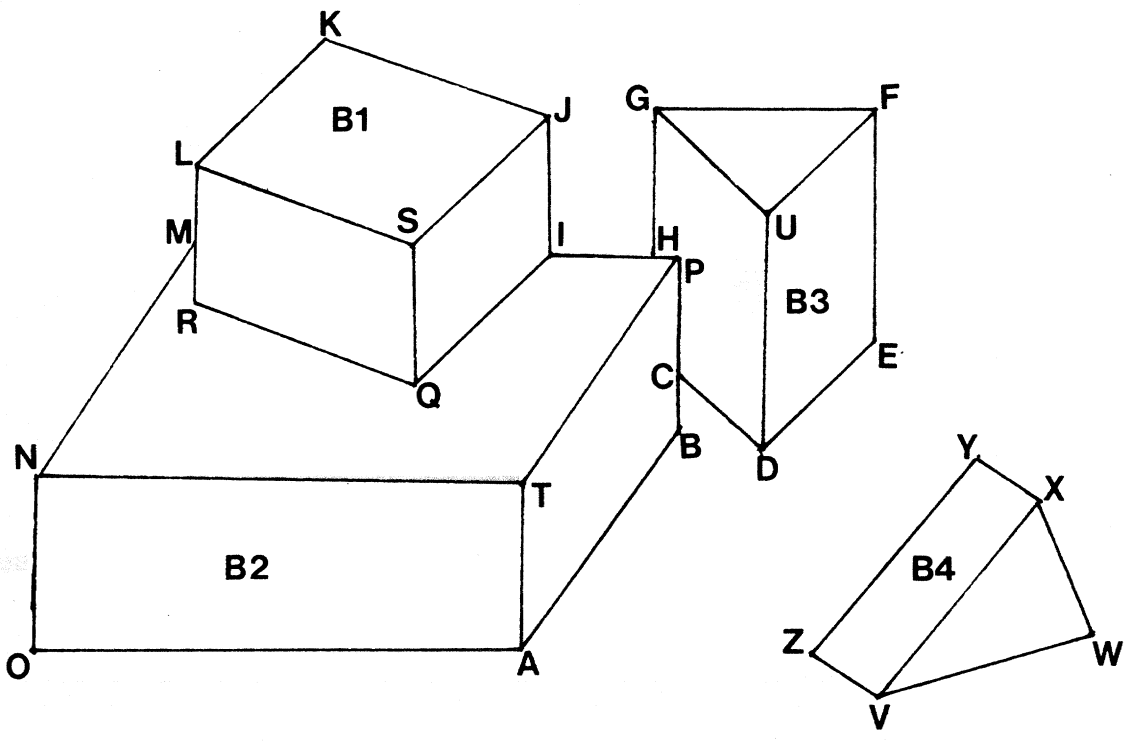


Fig. 1. Example of Scene

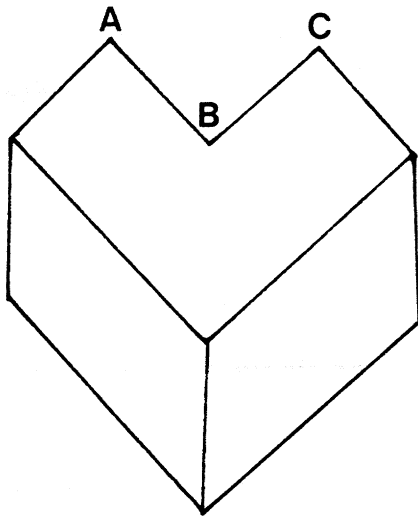
examination of larger or smaller areas. In our strategy, the smaller the area required to search lines, the higher priority we give to that search. In Fig.1, for instance, to determine the existence of an extension of line BC, it is enough to search a small area whose center is on the extension of the line. To find line IC, however, we should consider all possible directions of a line between IP and IJ. Thus the former search has priority over the latter.

The priority to extract the most obvious information first is in the following order.

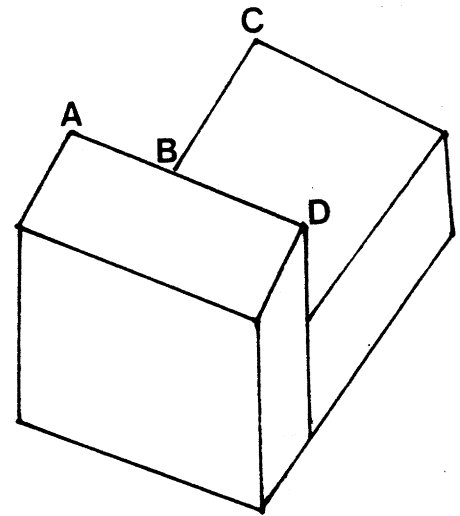
1. If two boundary lines make a concave point (such as point B in Fig.3), try to find an extension of them. If only one extension is found, track along this line. Most of such cases are like in Fig.3 (b) where one body hides the other. We can determine to which side of body this line belongs.

2. If no extensions of two concave lines are found, try to find another line which starts from the concave point. If only one line is found, track along this line. Most of these cases are as in Fig.3 (c) where it is not clear locally to which body this line belongs. In Fig.3 (c), line ED belongs to the upper body, but this is not always true. That is the lines AB, BC, BD are not sufficient to decide the relation.

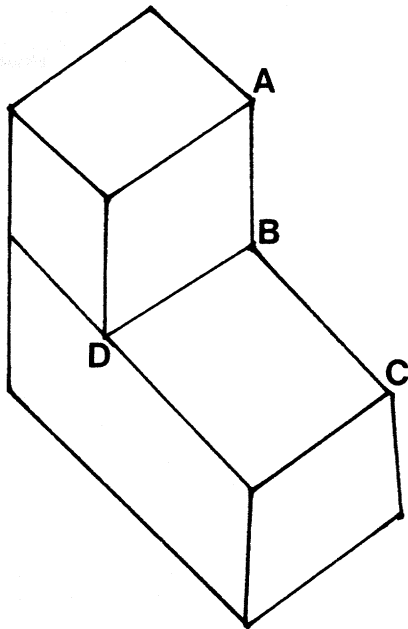
3. If both extensions of two lines are found at a concave point, try to find a third one. If only one line is found, track along this line. This is the case as shown in



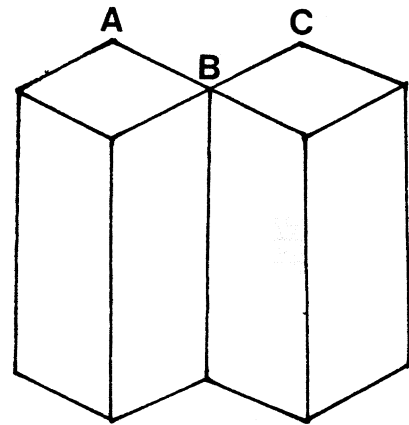
(a)



(b)



(c)



(d)

Fig. 3. Examples of concave boundary lines

Fig.3 (d) where the third line is the boundary line.

Whenever tracking terminates, an attempt is always made to connect the new line to the other lines that were already found. If more than one line segment is found in (1), (2) or (3), the tracking of those lines is put off hopefully to be clarified by the results of knowledge obtained in simpler cases. Fig.4 illustrates two extensions found at concave point P. The interpretation of the two lines is put off to treat simpler cases first. That is, one would continue examining the contour and lines AB and CD might be found next; then, by a circular search at points B (which is explained later), line BP would be found. At this stage it is easier to interpret lines AB and BP as boundary lines which separate two bodies. Then line DP might be found similarly and interpreted correctly.

4. If an end of a boundary line is left unconnected as PQ in Fig.5, try to find the line starting from the end point (Q in this example) by circular search. If multiple lines are found, try to decide which line is the boundary. If a boundary line is determined, track along it. In Fig.5, the dotted lines are found by circular search and the arrows show the boundary lines to be tracked.

5. If no line is found in the case (4) as stated above, extend the line (PQ in this example) by a certain length and test if the line is connected to other lines. If not, then

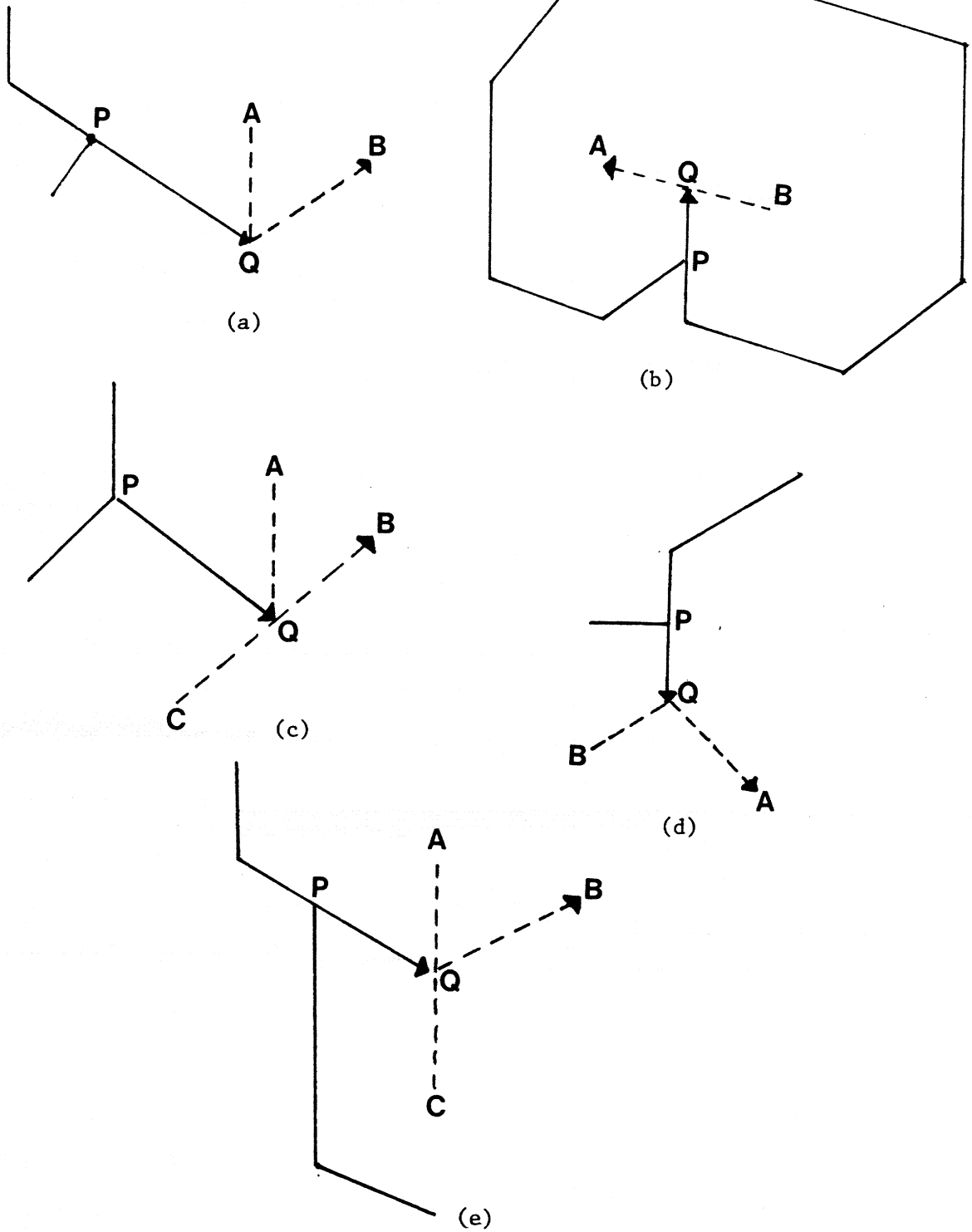


Fig. 5. Examples of line configuration found by circular search.

apply circular search again as in (4). This is necessary because the termination point of the tracking is not always precise.

Note that this process can be repeated until successful (that is either the line is connected to other lines or line segments are found by circular search).

6. If the boundary lines of a body are known, select the vertices of the boundary that might have internal lines starting from them. The selection of vertices is based on heuristics such as selecting upper right vertex rather than lower right vertex. At each vertex, try to find an internal line which is nearly parallel to other boundary lines. If one line is found, track along it. In Fig.1, for example, internal line JS is parallel to the boundary line KL or IQ, and QS is parallel to RI or IJ. Line FU is parallel to FD and XV is parallel to XZ. Thus it is often useful to find internal lines parallel to boundary lines of the same body. Note that search for parallels has small area.

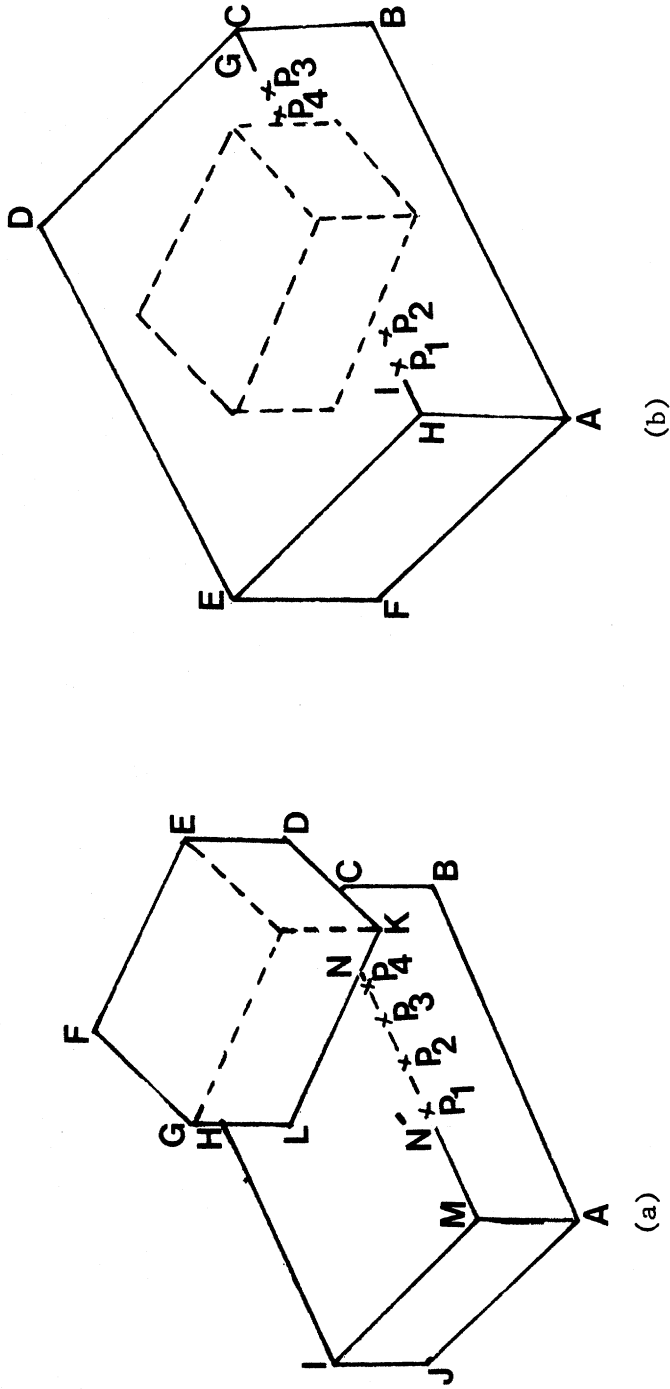
7. If no line is found in (6), try to find one by circular search between adjacent boundary lines. When one line is found, track along it. In Fig.6, circular search between BA and BC is necessary to find the internal line BF.

8. If two internal lines meet at a vertex, try to find another internal line starting at the vertex. This process is used in two cases. One is where no internal line was

found in (7) because of little difference in brightness between adjacent faces. Suppose in Fig.1, that the internal line SJ was not found at vertex J, but that LS and QS were found. Then try to find an internal line starting at S toward J. If there is enough contrast near S, a line segment is found. The other case is where a body is partly hidden by other bodies. In Fig.6, the triangular prism is partly hidden. After BE and CE are found, EF is searched for. In both cases, the direction of the line is sometimes predictable and sometimes not. If it is predictable, then try that direction. If it is unpredictable or if the predicted direction failed, then apply circular search between the two internal lines. If one line is found, track along it.

9. If an end of an internal line is not connected to any line, try to find lines starting from the end by circular search. If lines are found, track along them one by one.

10. If no line is found in (9), extend the line by a certain length as in (5) and test if it is connected to other lines. If not connected, try circular search again as in (9). This process can also be repeated until successful. Fig.7 illustrates this process. In Fig.7 (a), line MN' is not connected to others at N', thus step (9) is tried at N' and fails. The line is extended to P and (9) is again applied. This process is repeated until the line is



(dotted lines are not yet found in this stage)

Fig. 7. Examples of line verifying by circular search.

connected to line KL at N. Fig.7 (b) shows that line HI is extended by this process to P where a new line is found by circular search. Similarly line CG is extended to P. This process is useful so as to not miss a new body sitting on an obscure edge.

At each stage when an above step is finished, the obtained information is interpreted as shown in Fig.2 (block 3). For instance, if tracking along a line terminates, a test is made whether the line is an extension of other lines and/or the line is connected to other lines at a vertex. If a boundary line is connected to another boundary line, the body having the lines is split into two bodies and the properties of both lines and vertices are stored in an appropriate structure. In Fig.8, for example, line N'P' is obtained by tracking starting at point N. This line is interpreted as an extension of HN, and HN and N'P' are merged into one straight line using the equations of these two lines. Then, it is connected to CO and Fig.8 (b) is obtained. Before the line was connected to CO, there were two bodies B1 and B2 as in Fig.8 (a). Now body B2 is split into two bodies B2 and B3. We can interpret line NO as the boundary of B3 which hides a part of B2. The other properties of lines and vertices are obtained similarly at this stage.

2.2 Example

We illustrate the entire line-finding procedure with the aid of the example shown in Fig.9. At first, the contour lines AB, BC, CD, DE, EF, FG, GH, HI, IJ, JK and KA are obtained as shown in Fig.9 (a). Step (1) described in the previous section is tried for the concave points G and J. In this example, the position of G is not precise enough to find the extension of FG. On the other hand, a line segment is found as an extension of the line KJ. KJ is extended by tracking as far as L. Because there is no other point to which step (1) is applicable, step (2) is tried for point G. One line segment is found and extended till tracking terminates. Thus a line G'M' is obtained as in Fig.9 (b). This line is interpreted as an extension of FG and connected to JL. Then the position of point F, G, L are adjusted to as shown in Fig.9 (c). Now two bodies B1 and B2 are created by the boundary lines GL and JL. It is important to notice this, for it means that step (1) is again applicable (to point L) at this stage. Thus line FL is extended as far as M in Fig.9 (d) (Note that line MN' has not yet been found). LM is interpreted as an extension of FL but the end point M is not connected to any other lines. Thus vertices F, G, L and end point M are adjusted considering the new line LM. Here neither step (1), (2) nor (3) is applicable, so that (4) is now applied to M. Three lines are found by circular search as Fig.9 (d). MN' is determined as a boundary line and extended by tracking. When it terminates, the

line is connected to boundary line BC at vertex N as in Fig.9 (e). Body B1 splits into body B1 and B3. It is known at this stage that B1 is hidden by B3 and B2 is hidden partly by B3 and partly by B1. Next, step (6) is applied to each body one by one at each time selecting the easiest body for proposing the internal lines (in this example, the order is B3, B1, B2 because B3 hides B1 which hides B2). Internal lines CO and MO are found and connected at vertex O, but no line segment is found using step (6) and (7) applied to vertex E (this stage is shown in Fig.9 (e)). Step (8) is applied to vertex O and a line segment toward E is found. This is extended by tracking as far as E' as in Fig.9 (f). Line OE' fails to be connected to any other lines which activates step (10). After a few trials, OF' is extended to connect to vertex E. Similarly, internal line AM is obtained for body B1 and line IP is obtained for B2. When every step has finished, three bodies are known together with the relationships between them.

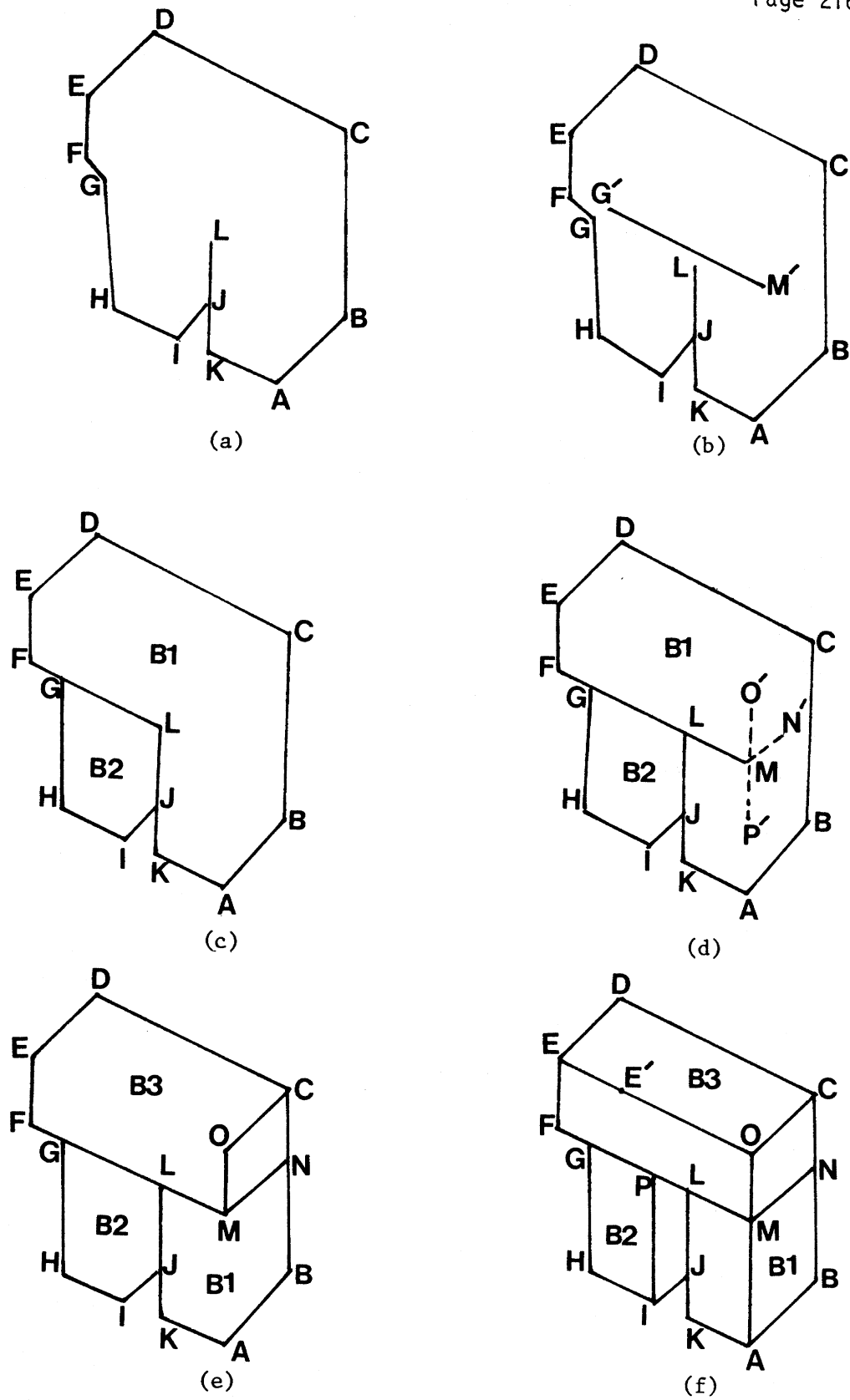


Fig. 9. Illustrates the procedure to find lines.

3. ALGORITHMS

This section describes the details of the algorithms that are used in finding contours and in the steps stated in section 2. Some of them, such as tracking and circular search are used in more than one step. An algorithm used in more than one step may be slightly different in each step but its essential part is not changed. In the tracking algorithm, for example, some changes occur depending on whether tracking is used for boundary lines or internal lines.

3.1 Contour Finding

Fig.10 shows the outline of the procedure to find contour lines. The picture data obtained with an image dissector usually consists of a large number of points (say about 100,000) each of which represents light intensity level. To speed up the processing, one point for every 8×8 points is sampled. This compressed picture data consists of $1/64$ the number of points in the original picture. To find the contour, this data is scanned till a contour point is found. The judgement of contour point is based upon the simple assumption that there is enough contrast between the background and objects. It is then checked whether or not the point is a noise point. If it is a real contour point, trace along the contour. Thus a set of contour points are found. Then, the picture data is again scanned until a new

contour point is found. This process is repeated for all the picture data. When all the sets of contour points have been found, each set is separately analysed.

Suppose a certain set of contour points is to be analysed. We now return to the original high-resolution picture. We can guess approximately the position of the boundary point in the original picture data which corresponds to the first point found in the sampled picture. The precise boundary point is searched for near this point. A set of contour points is obtained by tracing from this point in the same way as in the sample picture. A polygon is formed after we connect contour points one by one. To classify the points of this 'curve' into segments, the 'curvature' of the polygon is used. This curvature in a digital picture is defined here for convenience as shown in Fig.11. Each cell in the figure represents a contour point. The curvature of a point P is defined to be the difference in angle between PR and PQ (α), where Q and R are a constant number of points away from P (6 points in this case). If we plot the curvature along the contour as shown in Fig.12, we can tell what part is near a vertex and what part lies in a straight line.

Note that curvature is not very sensitive to noise or digitization error. If we integrate the curvature in part of a straight line, the result is nearly zero despite the effect of noise. If we sum up the curvature of consecutive points whose absolute value is greater than some threshold, we can determine

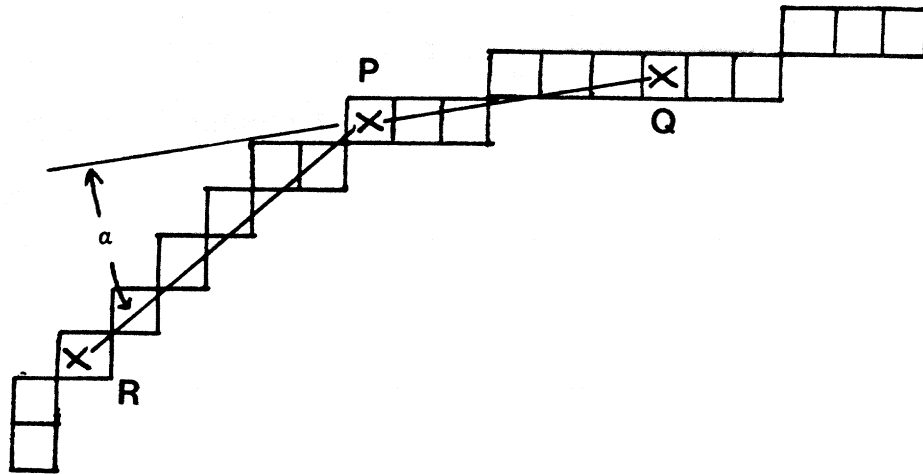


Fig. 11. Illustrates the definition of curvature

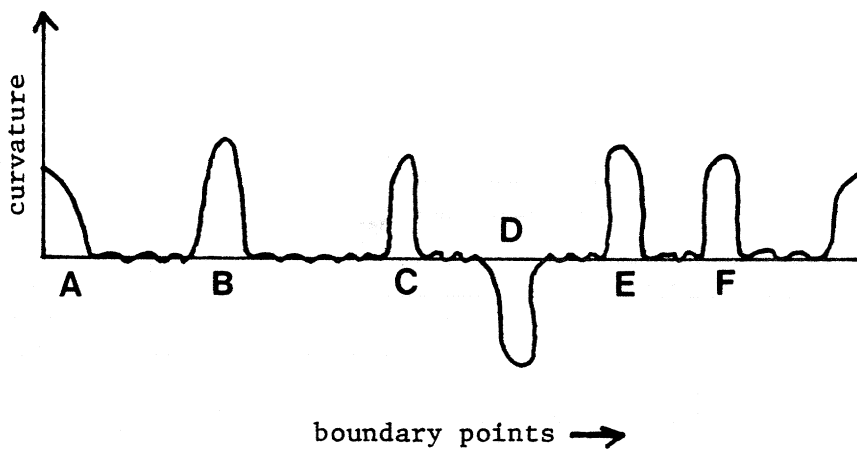
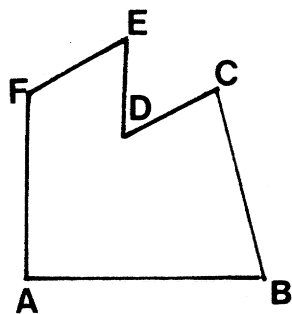


Fig. 12. Example of curvature.

the existence of a vertex. That is if this sum of the curvature of such points exceeds a certain threshold, there is a vertex near those points. Thus every contour point is classified to be either in the straight part of a line or near a vertex. Using points which belong to the straight part of a line, the equation of the line is calculated. Then each vertex is decided as an intersection of two adjacent lines.

3.2 Line segment Detection

A line Segment is detected given its direction and starting point. This procedure is used in most of the steps stated in section 1. The procedure consists of two parts. One is to detect the possible feature points which are to be regarded as elements of the line. The other is to test whether or not obtained feature points make a line segment.

In detecting feature points, we should consider various types of edges. Herskovits and Binford classified the light intensity profiles across an edge into 3 types, namely step, roof and edge-effect, and proposed 3 types of boundary detectors. In this paper, a roof type detector is not considered because roof type edges can be detected by a step detector or an edge-effect detector. In addition, most roof type edges are accompanied by step or edge-effect types. We set up local Cartesian coordinates $U-V$ such that U is the direction of the line segment to be detected. Let $I(u,v)$ denote the light intensity at point (u,v) ,

and define the contrast function $F(v)$ at (u,v) as

$$F(v) = \sum_{j=1}^{j_m} \sum_{i=-i_m}^{i_m} \left\{ I(u+i, v+j) - I(u+i, v-j) \right\}$$

Suppose we have an intensity profile as shown in Fig.13 (a), $F_u(v)$ at P in Fig.13 (b) is the difference of summed intensity between area A_2 and A_1 . $F_u(v)$ for a typical step type profile (Fig.13 (a)) is shown in Fig.13 (c) in which the edge is detected as the peak. The typical profiles of $F_u(v)$ for other types are shown in Fig.14 where the edge is detected as the middle point between positive and negative peaks.

The basic procedure, therefore, is to detect the peak of $F_u(v)$ and its position. The necessary properties for a peak are as follows (see Fig.15).

- (a) If $F_u(v)$ ranges from v_L to v_R , there must exist the maximum of $F_u(v)$ at v_m other than v_L or v_R .
- (b) $F_u(v_m) > f_m$ where f_m is threshold.
- (c) There must exist a minimum of $F_u(v)$ at v_1 between v_L and v_m and a minimum of $F_u(v)$ at v_2 between v_m and v_R such that

$$F_u(v_m) - F_u(v_1) > f_d$$

$$F_u(v_m) - F_u(v_2) > f_d \quad \text{where } f_d \text{ is a threshold}$$

If such v_m is found, the left of the peak (v_3) and the right of the peak (v_4) are determined as the intersection of $F_u(v)$ with the line $F_u(v) = f_t$ as shown in Fig.15. The value of f_t depends on $F_u(v_m)$ and is represented as

$$f_t = c_1 F_u(v_m) + c_0$$

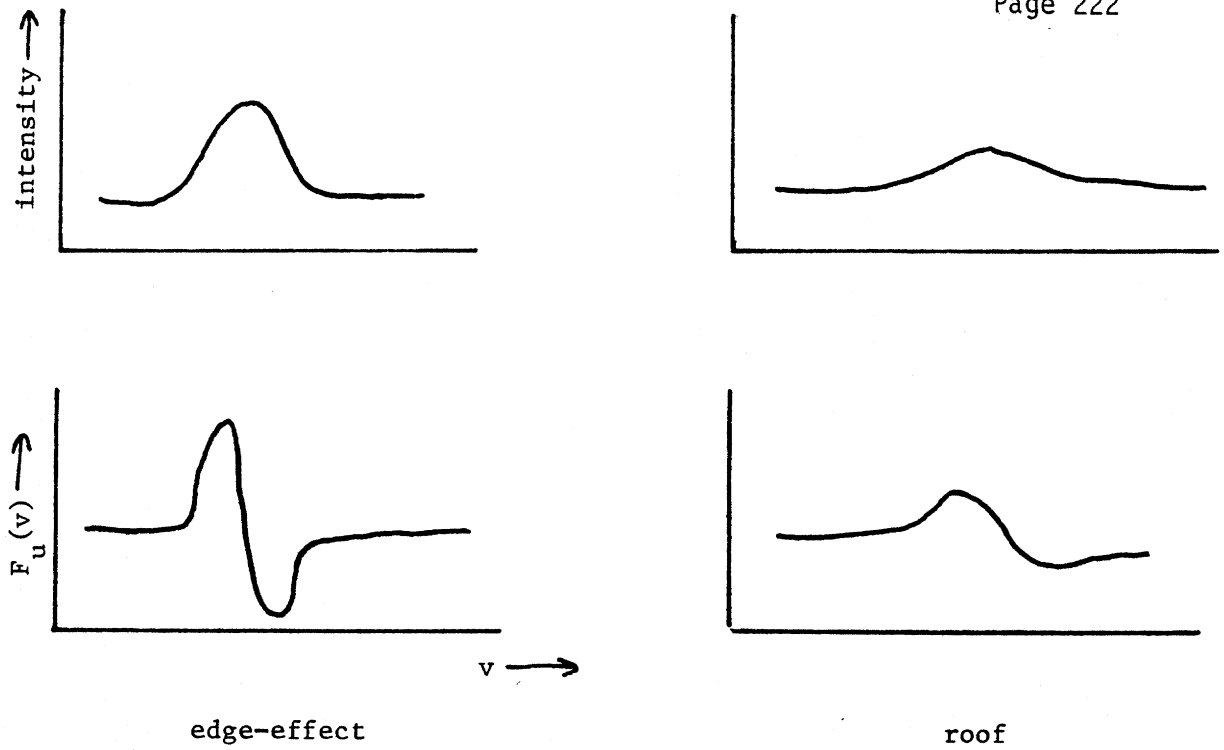


Fig. 14. Typical profile of $F_u(v)$.

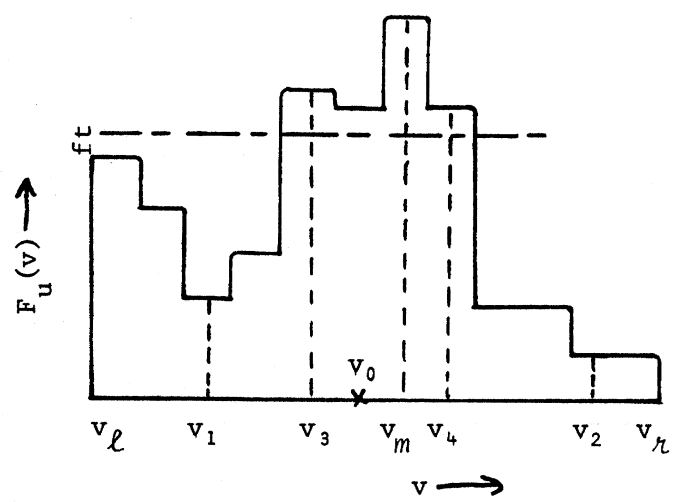


Fig. 15. Peak detection.

where c_1 and c_0 are constant and $0 < c_1 < 1$, $c_0 > 0$

The position of the peak v_0 is obtained as the middle of v_3 and v_4 . If more than one peak is found between v_l and v_r , the point v_0 which is nearest to the middle of v_l and v_r is adopted. A negative peak is similarly detected. A feature point for an edge-effect or roof is obtained as the middle of the positive and negative peaks, if both are found, (although the threshold f_m is not the same as in the simple positive or negative peak detection). This method for the detection of peaks and positions is not appreciably affected by noise.

The other part of the line segment detection is a test of the co-linearity for the detected feature points. Suppose concave boundary lines L_0 and L_1 meet at P_0 and suppose the line segment extending L_0 is tested as shown in Fig.16. Feature points are detected in a rectangular search area with given length and width whose direction is equal to that of $L_0 (= U)$, at an appropriate place where the detection of feature points is not affected by the edge corresponding to L_1 . Feature points are detected along the direction v at the center points P_1, P_2, \dots, P_m sequentially. If positive peaks are found at M_1, M_2, \dots, M_n as shown in the figure, the linearity of the points are tested as follows.

(a) The number of the feature points must exceed a threshold number n .

(b) The deviation σ^2 of the points in line fitting with the

least square method should be less than a threshold δ_t^2 .

- (c) let U' denote the direction of line segment obtained by line fitting,

$$|U' - U| < U_d$$

where $|U' - U|$ denotes the difference in directions U' and U

Similar tests are made for the different types of feature points. If more than one type of line segment is found, the selection depends on the following criteria.

- (a) If an edge-effect type is found, then it is selected.
 (b) For the line segment with δ^2 and U' , let the criterion

function C be

$$C = \delta^2 + w_u |U' - U| \quad \text{where } w \text{ is a constant}$$

The line segment selected is the one with smaller C .

3.3 Circular Search

Circular search is used to search for lines starting at a given point. The direction of the lines to be searched for is not known. The range of directions in circular search depends upon the particular case. Suppose two known lines L_1 and L_2 meet at P as in Fig.17 (a) and suppose we wish to search for lines lying between them. The search range α is between two lines L'_1 and L'_2 whose directions are slightly inside of L_1 and L_2 respectively. If lines starting at point P of line L_0 are searched for as in Fig.17 (b), L'_1 and L'_2 are similarly set inside

of L_0 . The center point P of the circular search is not always precisely determined, especially when tracking along a line has terminated at point P as shown in Fig.17 (b). Therefore circular search should not be too sensitive to the position of the center point.

It might be natural to try to detect feature points, as defined in section 3.2, based upon $F_u(v)$ along arcs around the center. The difficulty with this search is the classification of feature points into line segments if there is more than one as shown in Fig.18. To avoid this difficulty, a simple algorithm is used in this paper. Its basic method is to apply line segment detection successively in various directions. This is illustrated in Fig.19, where successive line segment detections toward u_1, u_2 and u_3 are applied. The step of direction change and search area (A_1, A_2 and A_3 in the figure) are determined so that line segments of any direction near the center point can be found. Thus successive circular search along a line as shown in Fig.7 can find lines starting at points between two adjacent center points (e.g. line L in Fig.20 starting between P_1 and P_2). The algorithm for line segment detection is the same as described in 3.2 except with respect to thresholds and search area. Because the search areas for different directions overlap each other, the same line segment may be found in different searches. Each time a line segment is found by line segment detection, a check should be made whether or not it is the same as the one

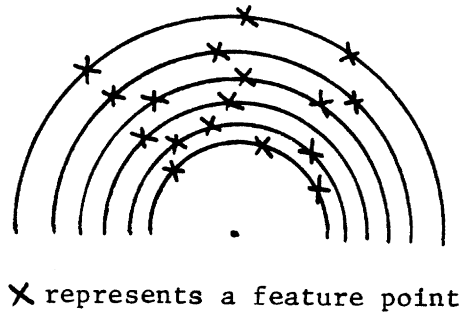


Fig. 18. Illustrates difficulty in classification of feature points.

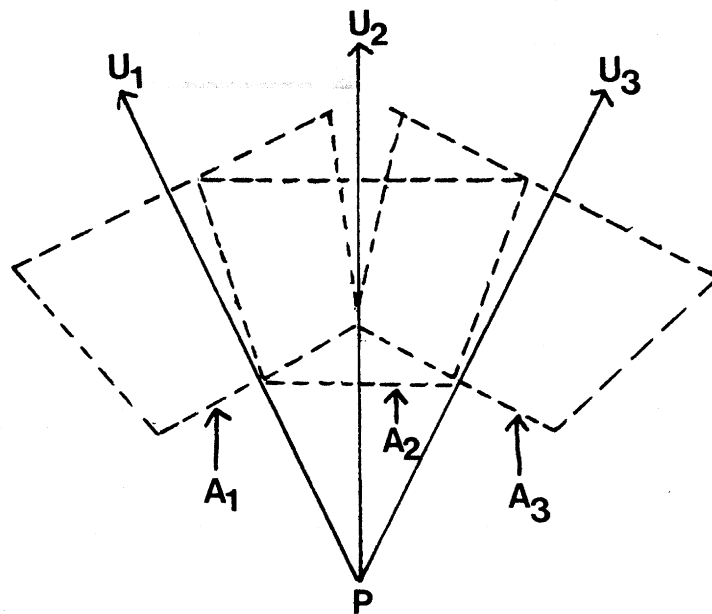


Fig. 19. Successive line detection in circular search.

obtained by the previous detection.

If the center point of circular search has not been determined precisely, it is not always possible to find all the lines starting at the given point. In Fig.21, for example, line L_2 might be missed in circular search at P_0 . To avoid this inconvenience, when line segments are found (such as L_1 and L_3 in the figure), a new center point P_1 is calculated based on the known line (L_0) and the obtained line segments (L_1 and L_3). Then circular search is applied again at P_1 .

3.4 Tracking

Tracking is used when a line segment is given, to track along it until it terminates. The requirements for a tracking procedure are 1) the line should not be lost due to the effect of other lines or noise, and 2) the procedure should terminate as precisely as possible at the end of the line. These requirements are contradictory in that the termination condition should be strict to satisfy the second requirement which makes it difficult to satisfy the first. The following algorithm is a compromise between these requirements.

The basic procedure is to predict the location of a feature point and to search for it near the point using line segment detection. The result of the search is classified into the following 4 cases.

- (a) there is no feature point.

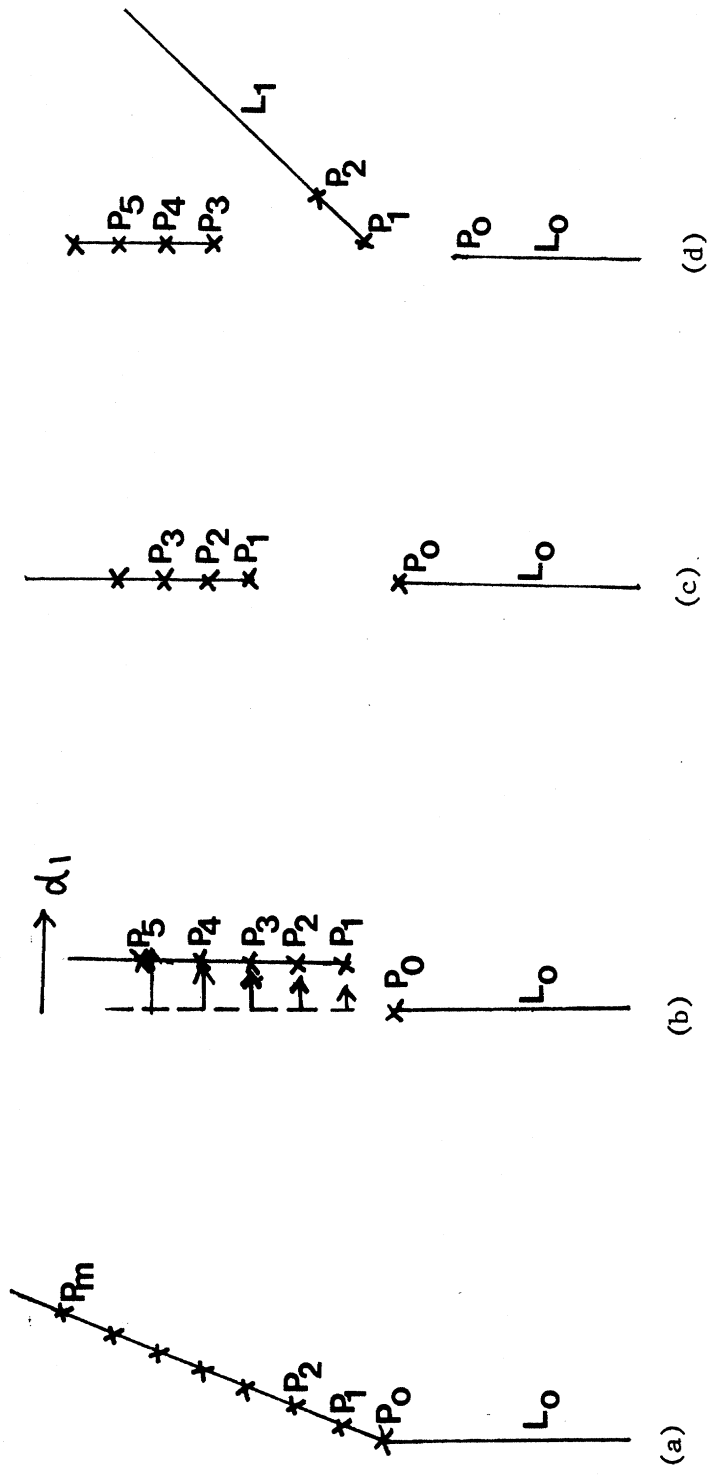


Fig. 22. Examples of tracking.

- (b) a feature point is on the line.
- (c) A feature point is not on the line.
- (d) It is not clear whether or not a feature point is on the line.

In case (a), the detection of a feature point is similar to line segment detection except that the type of edge is already known so that the thresholds stated in 3.2 can be adjusted based on the average peak of $F_u(v)$. The decision between cases (b), (c) and (d) is made using the distance d between the point and the line. That is

If $d \leq d_1$ then case (b)

If $d > d_2$ then case (c)

If $d_1 < d \leq d_2$ then case (d)

The threshold d_1 changes depending on the state of tracking. The state of tracking is represented by two integers m_1 and m_2 which are set initially to 0. The value of m_1 and m_2 are changed for each case (a), (b), (c) and (d) as follows.

(a) $m_1 = m_1 + 1$

(b) If $m_1 > m_2$, $m_1 = m_1 - 1$, (where m_1 is a constant)

Otherwise, $m_1 = 0$, $m_2 = 0$, and classify those feature points into (b) which have been classified into case (d) in the previous steps of tracking. Adjust the equation of the line with these points and the present feature point

(c) If $d \leq d_3$ and $m_1 > m_2$, $m_1 = m_1 - 1$

(where d is a constant)

Otherwise no change

(d) $m_2 = m_2 + 1$, and if $m_1 > m_d$, $m_1 = m_1 - 1$

The threshold d_1 is represented as

$$d_1 = d_0 + w_m m_2 \quad (\text{where } d_0 \text{ and } w_m \text{ are constants})$$

This procedure is repeated and tracking proceeds step by step extending the line until the termination condition is satisfied.

The termination condition of tracking is either

$$m_1 > m_n \text{ or } m_1 + m_2 > m_t \quad (\text{where } m_n \text{ and } m_t \text{ are constants})$$

The terminal point is defined as the last point classified into case (b). Fig.22 illustrates how this algorithm works. In Fig.22 (a), two lines cross at P_0 . Tracking might finish at some point beyond P_0 (P_m in the figure) which satisfies the termination condition. The terminal point of tracking is, however, determined more precisely near P_0 (P_1 or P_2). In Fig.22 (b), P_1, P_2, P_3, P_4 are classified into case (d) increasing the value of m_2 which classifies P_5 into case (b). Then the line is adjusted with these points which are now classified into case (b) and tracking proceeds.

Fig.22 (c) and (d) illustrate that even if a part of the intensity profile is disturbed by noise or other lines, tracking does not terminate there. In Fig.22 (d), however, if the light intensity of the right side of L_0 changes across L_1 , the type of feature points might change across L_1 . Thus feature points P_3, P_4

,... might not be obtained and tracking might terminate at P_1 . When tracking terminates, the line segment detection is applied at the extension of the line to see if another type of line segment is found. If found, we adjust the line equation and tracking proceeds. If not found, tracking finally terminates at point P_1 and the position of P_1 is adjusted with the line equation. The above procedure often extends the line across other lines when it terminates temporarily at their crossing as in Fig.9 (b) where tracking along G'M' crosses many vertical lines.

4. EXPERIMENTAL RESULTS AND COMMENTS

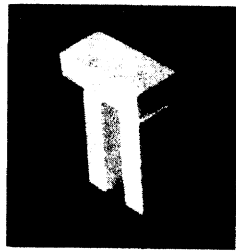
To test the program, experiments are made with cubes and wedges having relatively uniform white surfaces placed on a black background. The image dissector camera, used as an input device, dissects the scene onto a 20000 x 20000 (octal) grid. In this experiment, one point for every 8 x 8 block of grid elements is sampled. Thus, the scene is represented by 1024 x 1024 grid points. Objects occupy only a part of the scene. In the typical scene, the rectangular area which includes the objects of interest may consist of about 400 x 400 points. This area is divided into blocks each of which is made of 64 x 64 points and stored in disk memory. When a light intensity at some point is required, a block containing the point and adjacent blocks are stored in core memory. The core memory is accessed for the input of the light intensity until a point outside of those blocks is referenced.

Video input is at first converted into a 10 bit digital number which is an inverse linear measure of the light intensity. It is again converted into 10 bit logarithmic measure. Some intensity level resolution is lost in the logarithmic conversion. In this experiment the light intensity is represented by a little less than 100 levels. The input data for a clear bright edge in the dark background is blurred due to some limitations (mostly defocusing). If the intensity change is a step function, there

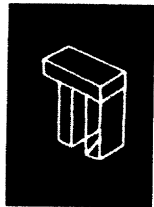
is a transient area in the input data about 10 points wide. Thus the resolution of the picture is regarded as 10 points. The parameters used in line segment detection and tracking are based upon this resolution. Features of the picture involving resolution of less than 10 points are not usually found.

Some results are shown in Fig.23. The difficulty or processing time of the recognition depends not only on the complexity of the object but also on the information extracted at each stage. In Fig.23 (c), for example, boundary lines SJ, KS and QS are easily proposed as the extension of contour lines. On the other hand, it is not easy to find boundary lines KM or LM in Fig.23 (c). That is, after DK and HL are found, circular search is necessary at K and L respectively. Circular search is less reliable in finding a line segment, and more time consuming. Once the boundary lines are determined, all the internal lines are proposed in both cases. But tracking along VW in Fig.23 (c) and EN in Fig.23 (c) terminates in the middle. Then step (10) stated in section 1.1 is applied. This is the most time consuming process (about 10 times more than the simple tracking process).

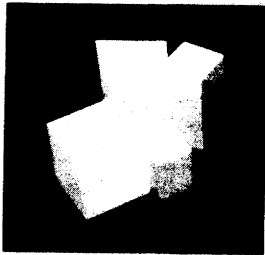
Some examples of the result of a hierarchical program are shown in Fig.24. Hierarchical programs may look at the whole scene homogeneously and pick up feature points. Lines are found with those feature points obtained in the previous stage. It is very difficult to determine a priori the various thresholds for



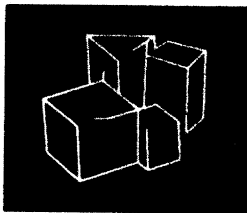
(a₃)



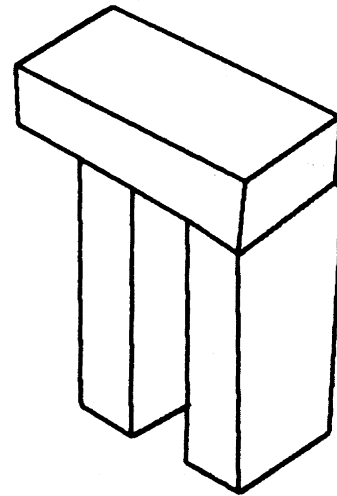
(b₃)



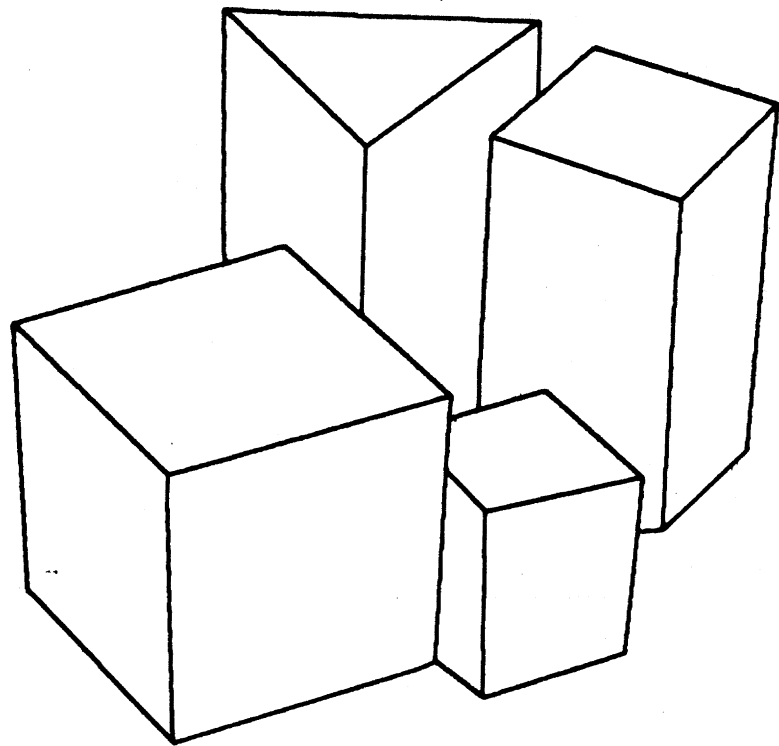
(a₄)



(b₄)



(c₃)



(c₄)

(a_i): Objects, (b_i): Result of Hierarchical Program
(c_i): Result of this program

Fig. 24. Examples of Comparison Between Hierarchical and Heterarchical Program.

detection of feature points, line fitting and connection of lines. In this heterarchical program, it is possible to adjust various thresholds with the context of the information obtained previously. Furthermore the algorithm itself can be modified case by case. (For instance, tracking algorithm is changed depending on whether the line is a boundary or internal.) The results of experiments with moderately complex scenes are mostly satisfactory. Because of the many checks for consistency of lines and vertices, the program has small probability of finding false lines.

However, there are some limitations of this program at present. One of them is that bodies may be missed in some cases. A simple example is shown in Fig.25. The boundary lines AB and BC in Fig.25 (a) are not proposed though the other contour lines and internal lines are found, because the resulting regions are so 'neat' that no conceive vertices activate step (1). In such a case when bodies are neatly stacked, it is necessary to search for boundary lines which start from some points on the boundary line. In Fig.25 (b) body B2 is not found. To find a body that is included in a face of another body, it is necessary to search for line segments inside the region. Though these two kind of search (search along the boundary line and search in the region) are required to find all the bodies in the scenes as shown in Fig. 25, they are still more effective than the exhaustive search in the entire scene. Besides, it is simpler to

Preceding page blank

interpret the scene when a line is found by those searches. This procedure, however, is left to future work.

The other limitation of the present program is, as stated in the introduction, that it is not always applicable to concave objects. Fig.26 (a) shows a simple example. Line BD is found as an extension of line CB. If all the bodies are convex, line BD is interpreted as the boundary line as shown in Fig.26 (b). This does not hold for concave bodies. In this program, line ED is regarded as a boundary line, and then line DE can be found by circular search at D. At this stage, however DE should be interpreted as an internal line of the same body instead of the boundary line which separates the body into two. If DE is interpreted correctly, then line BD can be determined as an internal line. This procedure should also be implemented in the present program.

CONCLUSION

A heterarchical program to recognize polyhedra is presented. The program is based upon the strategy of recognizing objects step by step, at each time making use of the previous results. The order of the lines to be detected is 1) contour lines (boundary of bodies and the background), 2) boundary lines which are the boundary between two bodies, 3) internal lines (intersection of two faces of the same body. Among boundary lines or among internal lines, the 'most plausible lines' are proposed at each stage and an attempt is made to find the line. To find a line, the range where a line segment may exist is proposed and it is detected in a suitable way for the proposed range. If a proper line segment is found, the end of the line is determined by tracking along the line. When the line is determined, the program tries to understand the scene taking this line into consideration. Because lines are mostly proposed instead of found by exhaustive search in the scene, the program is relatively effective. Results of the experiment using an image dissector are satisfactory for scenes including a few blocks and wedges. Although the present program has limitations, some of them may be overcome by developments proposed here for future work.

REFERENCES

- 1) M. Minsky and S. Papert, 'Proposal to ARPA for research on artificial intelligence at M.I.T., 1970-1971', MIT Project MAC Artificial Intelligence Memo No. 185, 1970.
- 2) A. Guzman, 'Computer Recognition Of Three-Dimensional Objects In A Visual Scene', MIT MAC-TR-59 (Thesis), 1968.
- 3) A. Herskovits and T. Binford, 'On Boundary Detection', MIT Project MAC Artificial Intelligence Memo No. 183, 1970.
- 4) E. Horn, 'The Image Dissector 'eye' ', MIT Project MAC Artificial Intelligence Memo No. 178, 1969.

VISNEM: A BAG OF "ROBOTICS" FORMULAE

B. K. P. Horn

December 1972

ABSTRACT

Here collected you will find a number of methods for solving certain kinds of "algebraic" problems found in vision and manipulation programs for our AMF arm and our TVC eye. They are collected here to avoid the need to regenerate them when needed and because I wanted to get rid of a large number of loose sheets of paper in my desk. Documented are various methods hidden in a number of old robotics and vision programs. Some are due to Tom Einfeld and Bill Gosper.

This paper was originally published as Vision Flash 34

TABLE OF CONTENTS:

1.	LINE AND LINE-SEGMENT MANIPULATION	
1.1	Line and line-segment representation and calculations	243
1.2	Projection of a rectangular corner	250
1.3	Relations between sides and angles in triangles	253
1.4	Proximity determination and multi-entry buckets	254
2.	LEAST SQUARES SOLUTION METHODS	
2.1	Solution of overdetermined sets of equations	255
2.2	Least square curve fitting	256
2.3	Fitting a straight line	258
2.4	Fitting a polynomial	259
2.5	Fitting exponentials, discrete fourier transform	260
2.6	Fitting sines and cosines not harmonically related	261
3.	OTHER NUMBER CRUNCHING TECHNIQUES	
3.1	Solving sets of simultaneous linear difference equations	264
3.2	Stability of a system of two difference equations	265
3.3	Multi-dimensional Newton-Raphson zero finding	266
3.4	Interpolation of functions of two variables	267
3.5	Finding the parameters of an ellipse	268
3.6	Approximation to $n!$ and other randomness	270
3.7	Fast matrix inverse and bit-reversing table generation	271
4.	FOURIER TRANSFORM AND RELATED MATTERS	
4.1	Some transform methods for images	272
4.2	Fourier transform using a lens and monochromatic light	275
4.3	Some transform heuristics	276
4.4	Modulation transfer functions and square wave response	277
4.5	Convolutions of pill-boxes	278
4.6	What a defocused edge looks like	279
4.7	A linear theory of feature point marking	280
4.8	Fast Fourier transform	284

5. VISION AND OPTICS

5.1	Contrast in a rectangular corner	285
5.2	Scatter in an image dissector	287
5.3	Virtual images of a point source in a sphere	290
5.4	Tracking an intensity glob	291
5.5	An analog glob tracker	292
5.6	The surveyors mark and its friends	293
5.7	Object rotation matrix and stereo image projection	294
5.8	Exposure guide for the DEC 340 display	295
5.9	Lens formulae	296
5.10	Focal length calibration	298

6. ARM TO EYE TRANSFORM AND THE LIKE

6.1	Determining the transform from arm to eye space	300
6.2	Relationship between simplified and real coordinates	303
6.3	Vertical predicate for image lines	304
6.4	Going from image dissector to arm space	305
6.5	Typical arm-eye transform	306
6.6	Absolute orientation methods	308

7. MANIPULATOR DETAILS

7.1	Geometry of the AMF arm and the Alles hand	311
7.2	Compensation for grip and tilt motion	312
7.3	Conversion from rectangular to pseudo-polar	313
7.4	Maintaining a constant hand orientation	314
7.5	Measuring the inertia of a link in the arm	315

LINES, LINE-SEGMENTS, REPRESENTATIONS AND CALCULATIONS:

There are many ways of representing a line by an appropriate equation. A minimum of two parameters is required. Unfortunately these parameters tend to become singular near certain angles. Special purpose tests must be made to handle these cases. The more redundant 3 and 4 parameter representations not only solve this problem but simplify the operations required to manipulate lines and points.

Let θ be the inclination of the line to the x-axis and ρ its perpendicular distance from the origin.

Let (x_0, y_0) , (x_1, y_1) and (x_2, y_2) be points on the line. Then some equations for a line are:

$$y = mx + c \begin{cases} y - (\tan \theta)x - \frac{\rho}{\cos \theta} = 0 \\ x - (\cot \theta)y + \frac{\rho}{\sin \theta} = 0 \end{cases}$$

$$\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1} \longrightarrow (x-x_1)(y_2-y_1) - (y-y_1)(x_2-x_1) = 0$$

$$\text{Let } l = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2} \quad \cos \theta = \frac{x_2-x_1}{l} \quad \sin \theta = \frac{y_2-y_1}{l}$$

$$(x-x_0) \sin \theta - (y-y_0) \cos \theta = 0$$

$$x \sin \theta - y \cos \theta + \rho = 0 \quad \rho = -(x_0 \sin \theta - y_0 \cos \theta)$$

This last formulation is nice from a number of points of view. It is always non-singular, easy to use and allows the least-squares equations to be formulated neatly. Note that we have a choice of polarity, by multiplying the whole equation by -1 . We can choose a preferred direction along or across the line in this fashion. This allows us to represent directed lines.

Now we get on to using this representation. First we note that the equation of a line at right angles through a point (x_3, y_3) is:

$$(x-x_3) \cos \theta + (y-y_3) \sin \theta = 0$$

Our line and one point (x_0, y_0) on it implicitly define a coordinate transformation:

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} \cos \theta & \sin \theta \\ \sin \theta & -\cos \theta \end{vmatrix} \begin{vmatrix} x-x_0 \\ y-y_0 \end{vmatrix}$$

The inverse is:

$$\begin{vmatrix} x-x_0 \\ y-y_0 \end{vmatrix} = \begin{vmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{vmatrix} \begin{vmatrix} x' \\ y' \end{vmatrix}$$

The separation (perpendicular to the line) of two points:

$$(x_2-x_1) \sin \theta - (y_2-y_1) \cos \theta \quad (\text{A})$$

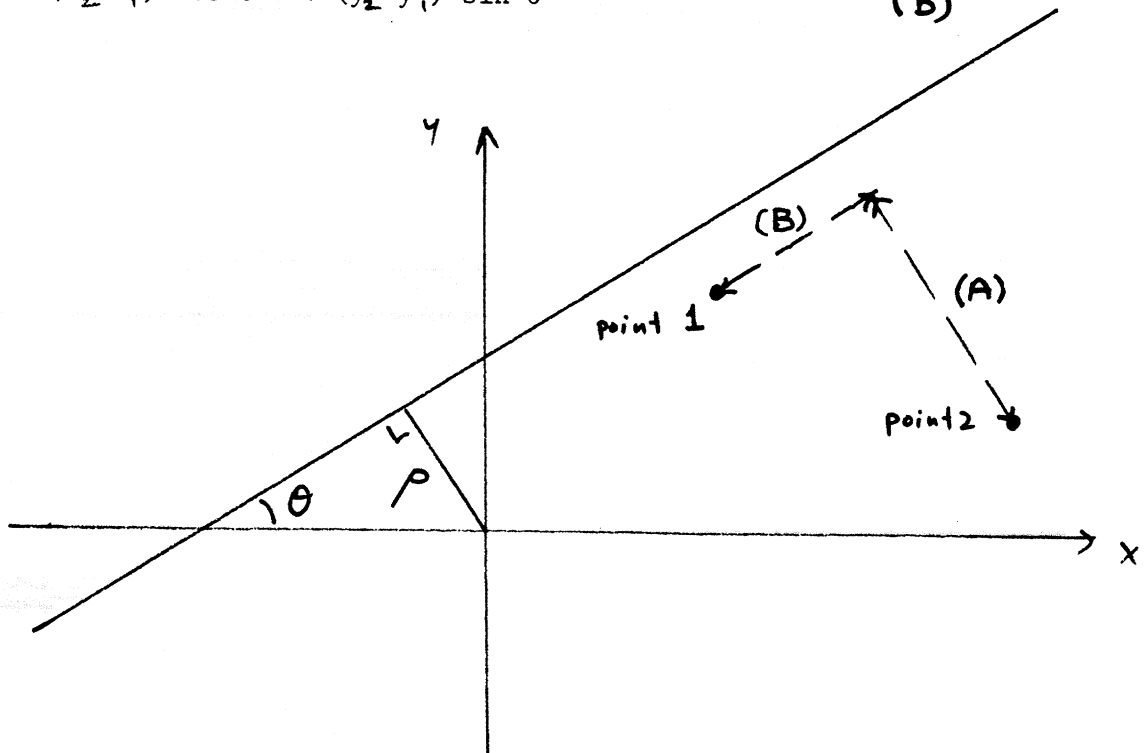
In particular the distance of a point from the line is :

$$x \sin \theta - y \cos \theta + \rho$$

Not surprisingly the equation shows the line to consist of points with zero distance from the line.

The separation (along the line) of two points:

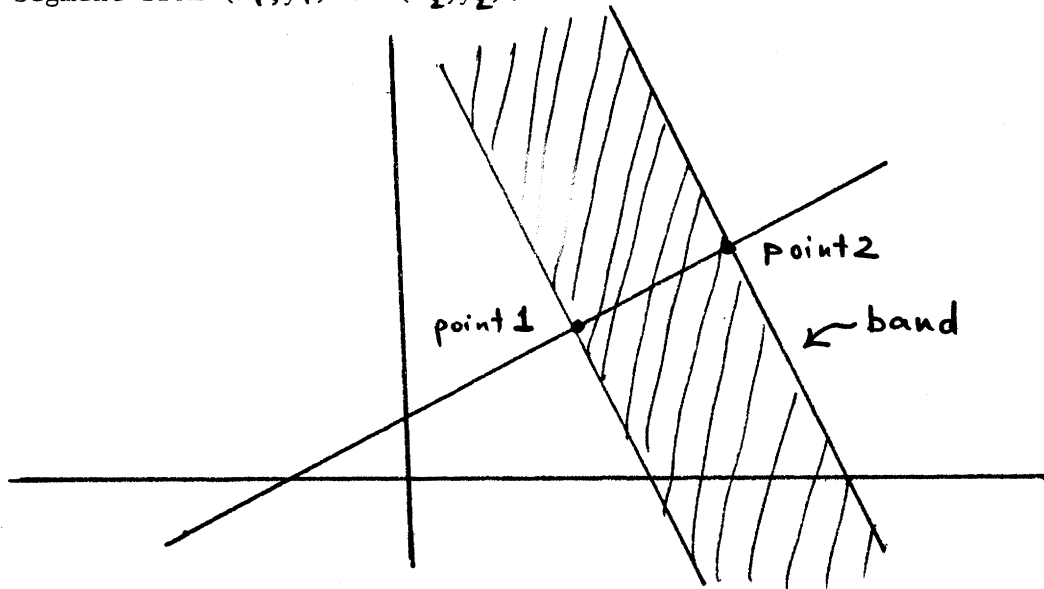
$$(x_2-x_1) \cos \theta + (y_2-y_1) \sin \theta \quad (\text{B})$$



In particular suppose that the end-points of a line segment are (x_1, y_1) and (x_2, y_2) . Then a point lies in the band generated by projecting this line segment perpendicular to the line if:

$$\left[(x-x_1) \cos \theta + (y-y_1) \sin \theta \right] \left[(x-x_2) \cos \theta + (y-y_2) \sin \theta \right] < 0$$

This can be used to determine if a point on the line lies within the line segment from (x_1, y_1) to (x_2, y_2) .



The sine of the angle between two lines:

$$\Delta = \sin(\theta_2 - \theta_1) = \sin \theta_1 \cos \theta_2 - \cos \theta_1 \sin \theta_2$$

We use this in the equation for the intersection of two lines:

$$x = (\cos \theta_2 p_1 - \cos \theta_1 p_2) / \Delta$$

$$y = (\sin \theta_2 p_1 - \sin \theta_1 p_2) / \Delta$$

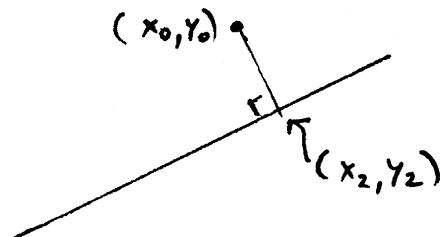
Naturally if $\Delta = 0$, the lines are parallel and we lose.

To find out if two line-segments intersect, we use these equations to find the intersection of the corresponding lines, then apply the above "band" test twice to see if this point is inside both line-segments.

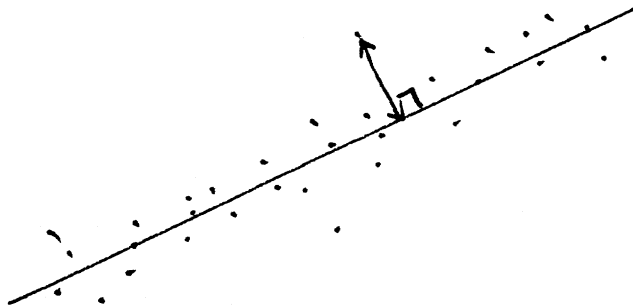
Next, to project a point perpendicularly onto the line we perform two opposite rotations about the origin:

$$\begin{cases} x_1 = x_0 \cos \theta + y_0 \sin \theta \\ y_1 = \rho \end{cases}$$

$$\begin{cases} x_2 = x_1 \cos \theta - y_1 \sin \theta \\ y_2 = x_1 \sin \theta + y_1 \cos \theta \end{cases}$$



Next we get to least-squares fitting a line to a set of points:



We minimise the sum of squares of perpendicular distances of the points from the line (moment of inertia):

$$e^2 = \sum_i (x_i \sin \theta - y_i \cos \theta + \rho)^2$$

$$\frac{d}{d\rho} e^2 = 2 \left[\sin \theta \sum_i x_i - \cos \theta \sum_i y_i + \rho \sum_i 1 \right]$$

For this to be 0 we must have:

$$\rho = -(x_0 \sin \theta - y_0 \cos \theta) \quad \text{where}$$

$$x_0 = \sum_i x_i / n \quad y_0 = \sum_i y_i / n$$

That is, the line passes through the centre of gravity of the points. Changing coordinates to a system relative to this favoured point we get:

$$x'_i = x_i - x_0 \quad y'_i = y_i - y_0$$

$$e^2 = \sum_i [(x_i - x_0) \sin \theta - (y_i - y_0) \cos \theta]^2$$

$$= \sum_i (x'_i \sin \theta - y'_i \cos \theta)^2$$

$$= (\sin \theta)^2 \sum_i x'^2_i - 2(\sin \theta)(\cos \theta) \sum_i x'_i y'_i + (\cos \theta)^2 \sum_i y'^2_i$$

$$= 1/2 \left[\left(\sum_i x'^2_i + \sum_i y'^2_i \right) - \left(\sum_i x'^2_i - \sum_i y'^2_i \right) \cos 2\theta - 2 \sum_i x'_i y'_i \sin 2\theta \right]$$

Note that we can find some of these terms as follows:

$$\sum_i x'_i = \sum_i x_i^2 - n x_0^2 \quad \sum_i y'_i = \sum_i y_i^2 - n y_0^2$$

$$\sum_i x'_i y'_i = \sum_i x_i y_i - n x_0 y_0$$

For compactness let:

$$a = 2 \sum_i x'_i y'_i \quad b = \sum_i x_i^2 - \sum_i y_i^2 \quad c = \sqrt{a^2 + b^2}$$

We get:

$$\frac{d}{d\theta} e^2 = b \sin 2\theta - a \cos 2\theta$$

For this to be zero we must have:

$$\tan 2\theta = a/b, \text{ i.e. } \sin 2\theta = a/c \quad \cos 2\theta = b/c$$

$$\text{Now } \cos \theta = \frac{\sqrt{1 + \cos 2\theta}}{2} = \sqrt{\frac{b+c}{2c}}$$

$$\text{And } \sin \theta = \frac{\sin 2\theta}{2 \cos \theta} = \frac{(a/2c) / \sqrt{\frac{b+c}{2c}}}{2 \cos \theta}$$

If $a = 0$, so is $\sin \theta$. Again we can choose to multiply the whole thing by -1 to decide the direction.

An equivalent way of getting this last result is the following:

$$\tan \theta = \frac{-1 + \sqrt{(\tan 2\theta)^2 + 1}}{(\tan 2\theta)} = \frac{-b + \sqrt{a^2 + b^2}}{a} = \frac{c-b}{a} = \frac{a}{c+b}$$

$$\cos \theta = 1 / \sqrt{1 + (\tan \theta)^2} = \frac{a}{\sqrt{a^2 + (c-b)^2}} = \frac{a}{\sqrt{2c(c-b)}} = \sqrt{\frac{b+c}{2c}}$$

Different least-squares fits can be described (for example one which minimises the sum of squares of distance parallel to the y-axis), but this one has the property that the fit does not depend on the coordinate system (invariant with rotation for example).

Note that while trigonometric functions appear all over these results, none ever get evaluated. Trigonometric functions are a most useful intellectual crutch; there is, however, seldom a need to actually use them in numeric calculations. One can usually replace them using the well-known relationships amongst them and reduce the required operations to +, -, *, / and $\sqrt{\quad}$ only. In the above formulas for least-square fitting for example, the only requirement is for two square root evaluations.

We ought to also check that we in fact have a minimum:

$$\frac{d^2}{d\rho^2} e^2 = n > 0$$

$$\frac{d^2}{d\theta^2} e^2 = 2b \cos 2\theta + 2a \sin 2\theta = 2 \frac{a^2 + b^2}{c} = 2\sqrt{a^2 + b^2} > 0$$

So indeed we have a minimum. We might also want to know the average error, and the average error if, instead, we had chosen the worst line (one at right angles to the best line).

$$e_1^2 = (\sin \theta)^2 \sum_i x_i'^2 - 2 \cos \theta \sin \theta \sum_i x_i' y_i' + (\cos \theta)^2 \sum_i y_i'^2$$

$$e_2^2 = (\cos \theta)^2 \sum_i x_i'^2 + 2 \sin \theta \cos \theta \sum_i x_i' y_i' + (\sin \theta)^2 \sum_i y_i'^2$$

Let $d = \sum_i x_i'^2 + \sum_i y_i'^2$ then we can also write the above as:

$$e_1^2 = 1/2 (d - c)$$

$$e_2^2 = 1/2 (d + c)$$

The ratio can be used as a "form-factor".

Note that all of this line-fitting can be modified to handle weighted points by simple multiplying the coordinates (x_i, y_i) by the weights w_i , and using $\sum_i w_i$ instead of n .

Now suppose we are given several lines and are required to find a point with minimum sum of squares of perpendicular distance.

$$\begin{aligned} e^2 &= \sum_i (x \sin \theta_i - y \cos \theta_i + \rho_i)^2 \\ &= x^2 \sum_i (\sin \theta_i)^2 - 2xy \sum_i \sin \theta_i \cos \theta_i + y^2 \sum_i (\cos \theta_i)^2 \\ &\quad + 2x \sum_i \sin \theta_i \rho_i - 2y \sum_i \cos \theta_i \rho_i + \sum_i \rho_i^2 \end{aligned}$$

$$\frac{d}{dx} e^2 = 2(x \sum_i (\sin \theta_i)^2 - y \sum_i \sin \theta_i \cos \theta_i + \sum_i \rho_i \sin \theta_i)$$

$$\frac{d}{dy} e^2 = 2(-x \sum_i \sin \theta_i \cos \theta_i + y \sum_i (\cos \theta_i)^2 - \sum_i \rho_i \cos \theta_i)$$

$$\text{Let } \Delta = \sum_i (\sin \theta_i)^2 \cdot \sum_i (\cos \theta_i)^2 - \left(\sum_i \sin \theta_i \cos \theta_i \right)^2$$

This will only be zero if all the lines are parallel. Solving the above set of equations in x and y we get:

$$x = \left[\sum_i (\cos \theta_i)^2 \sum_i \rho_i \sin \theta_i + \sum_i \sin \theta_i \cos \theta_i \sum_i \rho_i \cos \theta_i \right] / \Delta$$

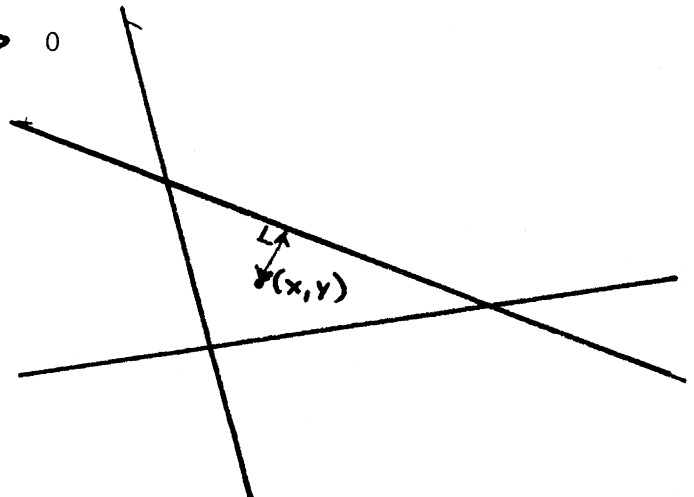
$$y = \left[\sum_i (\sin \theta_i \cos \theta_i) \sum_i \rho_i \sin \theta_i + \sum_i (\sin \theta_i)^2 \sum_i \rho_i \cos \theta_i \right] / \Delta$$

We also ought to check whether this gives us a minimum:

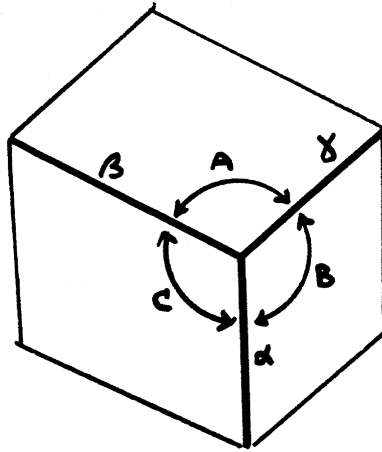
$$\frac{d^2}{dx^2} e^2 = 2 \sum_i (\sin \theta_i)^2 > 0$$

$$\frac{d^2}{dy^2} e^2 = 2 \sum_i (\cos \theta_i)^2 > 0$$

We can weight the lines by simply multiplying $\sin \theta_i$, $\cos \theta_i$, ρ_i by the weights w_i .



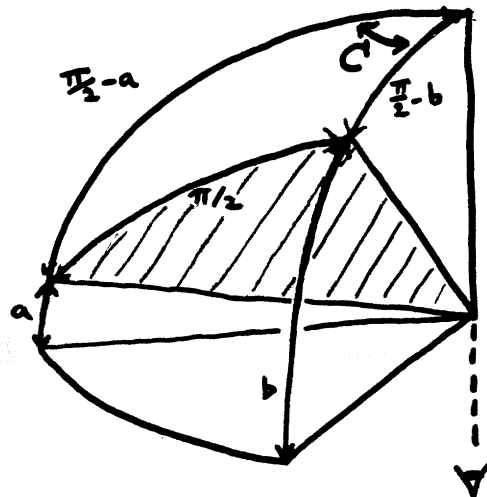
PROJECTION OF A RECTANGULAR CORNER:



Given that the tri-hedral vertex is formed by three planes meeting at right angles, find the inclinations of the three lines α, β, γ relative to the image plane. Let these angles be a, b, c . Note that the angle between the plane containing β, γ relative to the view vector is also a and so on for the other sides of the object.

This information is useful in defining the elevation and rotation of the eye relative to the coordinate system implicitly defined by the rectangular object. The angles can also be used to correct the foreshortening introduced by the inclination of the lines relative to the image plane.

Now consider the following spherical triangle:



Using a spherical trigonometry formula we get:

$$\begin{aligned} 0 &= \cos \frac{\pi}{2} = \cos\left(\frac{\pi}{2} - a\right) \cos\left(\frac{\pi}{2} - b\right) + \sin\left(\frac{\pi}{2} - a\right) \sin\left(\frac{\pi}{2} - b\right) \cos C \\ &= \sin a \sin b + \cos a \cos b \cos C \end{aligned}$$

$$\cos C = -\tan a \tan b \quad \text{and by symmetry:}$$

$$\cos B = -\tan c \tan a$$

$$\cos A = -\tan b \tan c$$

$$\tan^2 a = -\frac{\cos B \cos C}{\cos A}$$

$$\cos a = \frac{1}{\sqrt{1 + \tan^2 a}} = \sqrt{\frac{\cos A}{\cos A - \cos B \cos C}}$$

Where $a < 0$ if and only if $A > \pi$.

So we have the angle of the line α relative to the image plane. As mentioned this also gives us the inclination of the plane containing β and γ relative to the view vector. The others are found by symmetry.

To get the "unforshortened" length of the lines, that is the length in the image if they had been oriented at right angles to the view vector, we just divide by the cosine of the inclination angle:

$$\alpha_u = \alpha_f / \cos a$$

We also note that the cosines needed in the formulae can be obtained simply by using dot-products:

$$\cos A = (\beta \cdot \gamma) / \sqrt{(\beta \cdot \beta)(\gamma \cdot \gamma)}$$

So we don't need to use trig-functions at all, only +, -, *, / and $\sqrt{\quad}$.

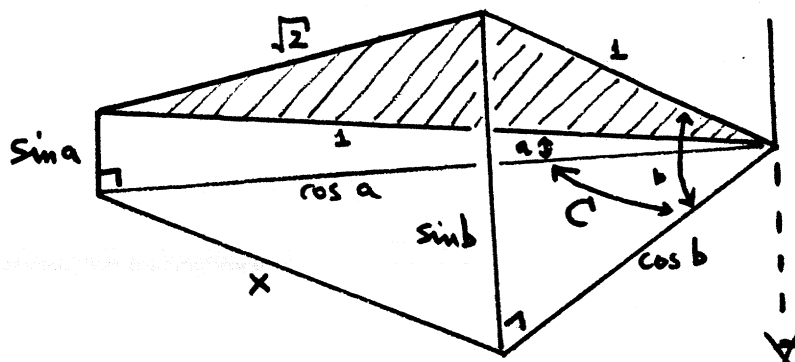
A few other random formulae in this relation:

Since $A + B + C = 2\pi$, $\cos A = \cos B \cos C - \sin B \sin C$

$$\cos a = \sqrt{-\frac{\cos A}{\sin B \sin C}} \quad \text{because of that.}$$

$$\sin a = \sqrt{\frac{\tan^2 a}{1 + \tan^2 a}} = \sqrt{\frac{\cos B \cos C}{\cos B \cos C - \cos A}} = \sqrt{\frac{\cos B \cos C}{\sin B \sin C}}$$

Another derivation not involving spherical triangles is as follows:



Using the formulae for plane triangles:

$$(\sin a - \sin b)^2 + x^2 = 2$$

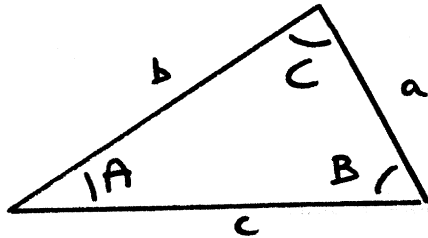
$$x^2 = \cos^2 a + \cos^2 b - 2 \cos a \cos b \cos C$$

$$2 = (\sin^2 a + \cos^2 a) + (\sin^2 b + \cos^2 b) - 2 \sin a \sin b - 2 \cos a \cos b \cos C$$

$$\cos C = -\tan a \tan b$$

as before

RELATIONS BETWEEN SIDES AND ANGLES OF ANY PLANE TRIANGLE:

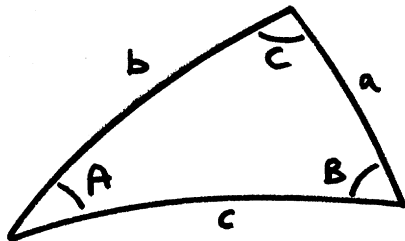


$$\frac{\sin A}{a} = \frac{\sin B}{b} = \frac{\sin C}{c} \quad (= \text{diameter of circumscribed circle})$$

$$a^2 = b^2 + c^2 - 2bc \cos A$$

$$a = b \cos C + c \cos B$$

RELATIONS IN ANY SPHERICAL TRIANGLE:



$$\frac{\sin A}{\sin a} = \frac{\sin B}{\sin b} = \frac{\sin C}{\sin c}$$

$$\cos a = \cos b \cos c + \sin b \sin c \cos A$$

$$\cos A = -\cos B \cos C + \sin B \sin C \cos a$$

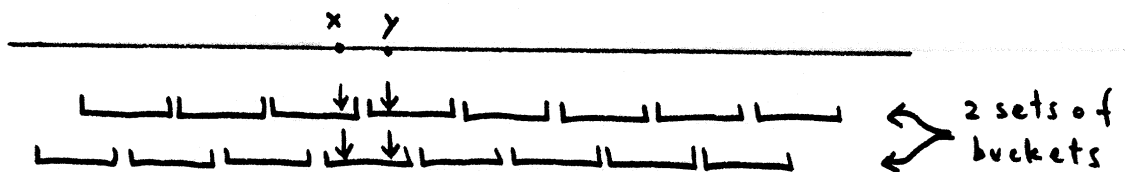
(a, b, c are the length of the arc on a unit sphere, alternatively one can think of them as the angle (in radians) subtended by the arc at the centre of the sphere)

PROXIMITY FINDING:

In the later phases of line-finding programs it is often necessary to repeatedly locate lines that are close together, lines that pass close to a given vertex and so on. To do this efficiently we require a fast access method to locate likely candidates for more sensitive tests. First consider the problem of deciding if two points are close together.

To tell if x and y are close together we can quantise both (by dividing by the quantisation interval size and truncating). If the two numbers $[x/d]$ and $[y/d]$ are the same we win, but it may be that x and y just straddle a boundary defined by our truncation algorithm. We need a second, interlaced set of boundaries and calculate $[x/d+.5]$, $[y/d+.5]$. Now if either pair of numbers matches we know that $|x-y| < d$. Conversely if $|x-y| < d/2$ we are guaranteed that at least one pair will match.

This method only comes into its own if we have large sets of points. We then simply find the two integer-codes for each one and add it to the appropriate buckets. To find which points are near a given point we determine its two integer-codes and collect the union of the two corresponding buckets.



This method can now be extended to n dimensions. We need at least $n+1$ sets of buckets if we use n -tetrahedrons. It may be more convenient to use 2^n sets of buckets if the unit cell is an n -cube. (d/n versus $d/2$ min sep)

Line-segments and curves can be handled by entering each point on them into the system. In practice one will only enter a set of points separated by the minimum distance guaranteed by the geometry used. Retrieval works in a symmetric way.

LEAST SQUARES SOLUTION OF AN OVERDETERMINED SET OF EQUATIONS:

Let us write the equations as follows:

$$A x = y + e$$

Where A is a given m by n matrix ($m > n$), x is the unknown n -vector, y is a given m -vector and e is an m -vector of errors which we are trying to minimise.

$$\begin{aligned} e^T e &= (A x - y)^T (A x - y) \\ &= (x^T A^T - y) (A x - y) \\ &= x^T A^T A x - y^T A x - x^T A^T y + y^T y \end{aligned}$$

$$\frac{d}{dx} e^T e = x^T A^T A + (A^T A x)^T - y^T A - (A^T y)^T + 0$$

$$\text{So: } \quad x^T A^T A = y^T A \quad \text{for this to be 0}$$

$$A^T A x = A^T y$$

$$x = (A^T A)^{-1} A^T y$$

$$\frac{d^2}{dx^2} e^T e = 2 A^T A$$

The diagonal elements of this will clearly be positive so we do have a minimum.

LEAST SQUARES CURVE FITTING:

Suppose we have a function $g(x)$ defined at the n points x_1, x_2 etc., and that we are trying to fit a function $f(x)$ which depends on the m parameters a_1, a_2 etc. so as to minimise the sum of squares of errors at the points x_1, x_2 etc. Let e_i be the error at point x_i .

$$e_i = f(x_i) - g(x_i)$$

Let \underline{e} be the n -vector of errors, \underline{f} the n -vector of fitted values, \underline{g} the n -vector of defined values. Then we are trying to minimise:

$$\underline{e}^T \underline{e} = (\underline{f} - \underline{g})^T (\underline{f} - \underline{g})$$

by varying the parameter m -vector \underline{a} . The derivative of $\underline{e}^T \underline{e}$ w.r.t. to this vector must be zero (and the second derivative positive).

$$(\underline{f} - \underline{g}) \frac{d\underline{f}}{d\underline{a}} = 0$$

where $\frac{d\underline{f}}{d\underline{a}}$ is an n by m matrix

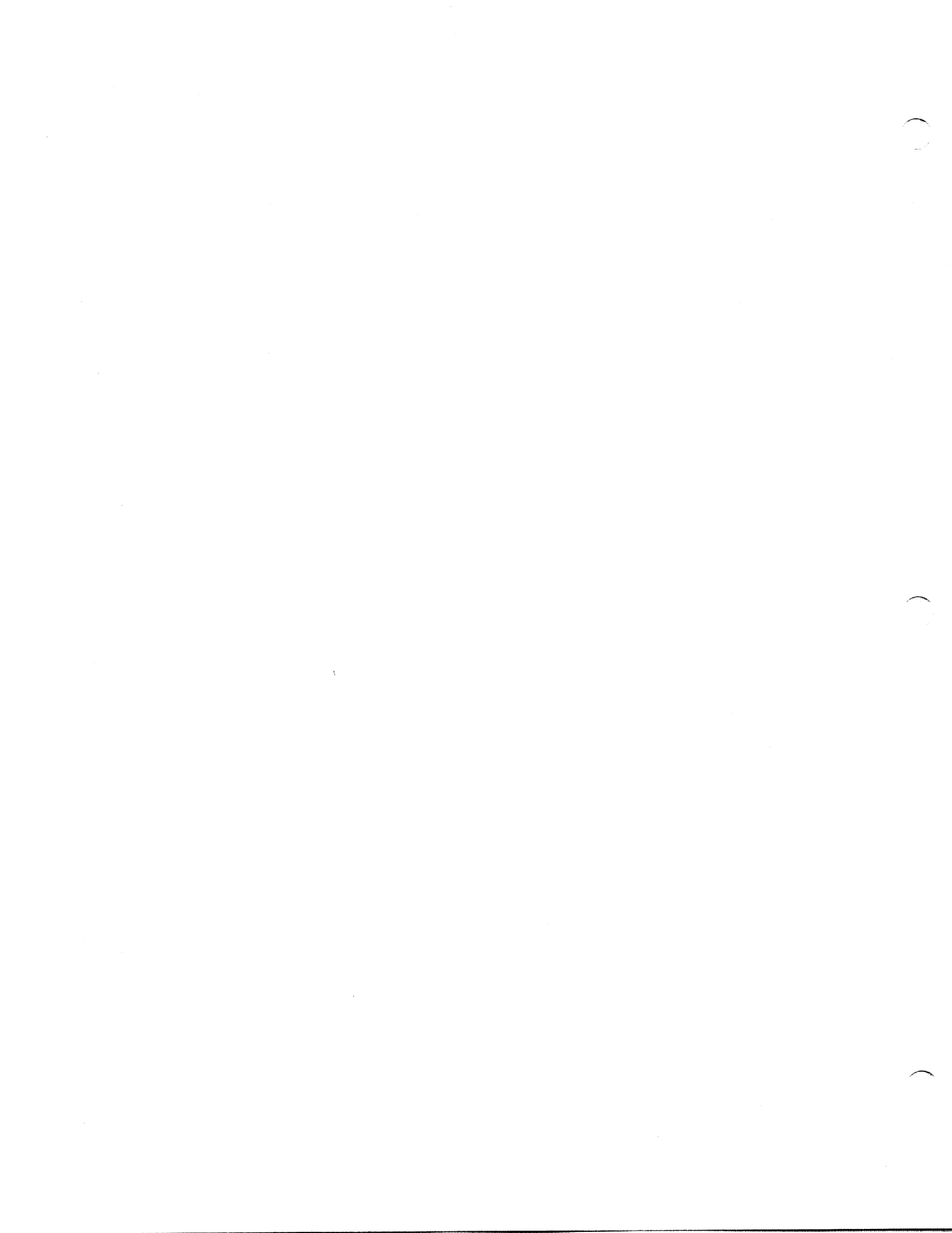
$$\text{So: } \underline{g} \frac{d\underline{f}}{d\underline{a}} = \underline{f} \frac{d\underline{f}}{d\underline{a}}$$

or written out in full:

$$\begin{array}{c|c|c|c|c} \frac{df(x_1)}{da_1} & \frac{df(x_2)}{da_1} & \vdots & \frac{df(x_n)}{da_1} & g(x_1) \\ \frac{df(x_1)}{da_2} & \frac{df(x_2)}{da_2} & \vdots & \frac{df(x_n)}{da_2} & g(x_2) \\ \dots & \dots & \dots & \dots & \dots \\ \frac{df(x_1)}{da_m} & \frac{df(x_2)}{da_m} & \dots & \frac{df(x_n)}{da_m} & g(x_n) \end{array} = \begin{array}{c|c|c|c|c} \frac{df(x_1)}{da_1} & \frac{df(x_2)}{da_1} & \vdots & \frac{df(x_n)}{da_1} & f(x_1) \\ \frac{df(x_1)}{da_2} & \frac{df(x_2)}{da_2} & \vdots & \frac{df(x_n)}{da_2} & f(x_2) \\ \dots & \dots & \dots & \dots & \dots \\ \frac{df(x_1)}{da_m} & \frac{df(x_2)}{da_m} & \dots & \frac{df(x_n)}{da_m} & f(x_n) \end{array}$$

To be able to solve these equations we choose a particularly simple form for $f(x)$ namely $\sum a_j f_j(x_i) = f(x_i)$, that is a linear one.

In this case the terms in the matrix $d\underline{f}/d\underline{a}$, namely $df(x_i)/da_j$ become:



$$\frac{df_i}{da_j} = f_{j,i}(x_i)$$

So the matrices become quite simple and let us denote them by F^T

$$F_{ji} = \frac{df_i}{da_j} = f_{j,i}(x_i)$$

We also note that because of the simple dependence of $f(x)$ on the parameters a_j we get:

$$\underline{f} = F \underline{a}$$

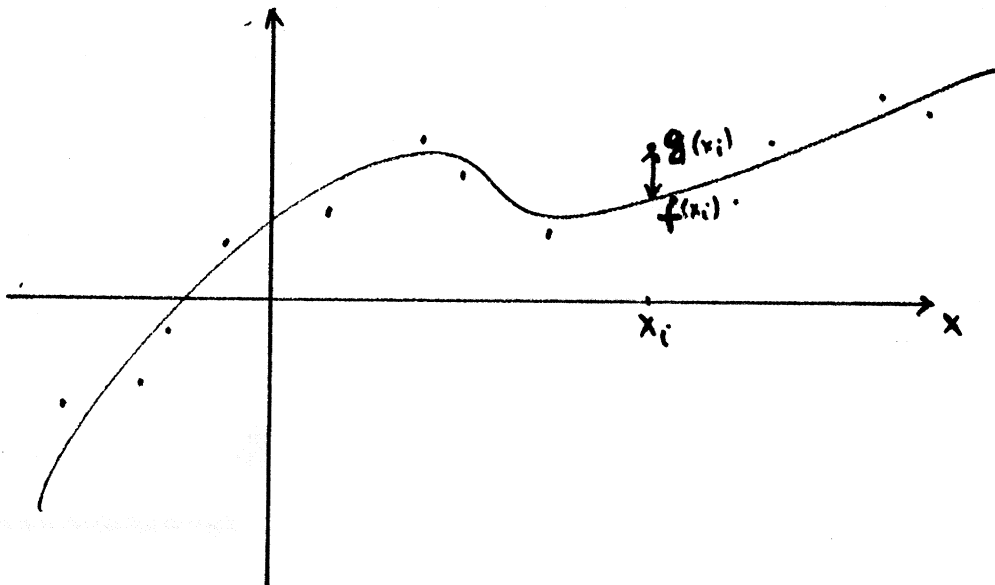
And we can rewrite the main equation as:

$$F^T \underline{g} = F^T F \underline{a}$$

Since $F^T F$ is square we can attempt to invert it and get:

$$\underline{a} = (F^T F)^{-1} F^T \underline{g}$$

So inverting the normal matrix allows us to solve for the parameters \underline{a} .



EXAMPLE OF FITTING A STRAIGHT LINE:

$$f(x) = a_1 + a_2 x \quad \text{and let } y_i = g(x_i)$$

$$\text{Here } \underline{a} = (a_1, a_2) \quad \underline{g} = (y_1, y_2, \dots, y_n) \quad f_1(x)=1 \quad f_2(x)=x$$

$$F^T = \begin{vmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \end{vmatrix} \quad F^T F = \begin{vmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{vmatrix}$$

$$\text{Let } \Delta = n \sum x_i^2 - (\sum x_i)^2, \text{ then}$$

$$(F^T F)^{-1} = \frac{1}{\Delta} \begin{vmatrix} \sum x_i^2 & -\sum x_i \\ -\sum x_i & n \end{vmatrix} \quad F^T \underline{g} = \begin{vmatrix} \sum y_i \\ \sum x_i y_i \end{vmatrix}$$

$$\underline{a} = (F^T F)^{-1} F^T \underline{g} = \frac{1}{\Delta} \begin{vmatrix} \sum x_i^2 & -\sum x_i \\ -\sum x_i & n \end{vmatrix} \begin{vmatrix} \sum y_i \\ \sum x_i y_i \end{vmatrix}$$

$$a_1 = (\sum x_i^2 \sum y_i - \sum x_i \sum x_i y_i) / \Delta$$

$$a_2 = (-\sum x_i \sum y_i + n \sum x_i y_i) / \Delta$$

Note that this is an unsymmetrical method different from the one demonstrated elsewhere in this memo and not suited for fitting lines in a line-drawing, for example.

APPLICATION TO FITTING A POLYNOMIAL:

$$f(x) = \sum_{j=0}^{m-1} a_j x^j \quad \text{and let } y_i = g(x_i)$$

$$\underline{a} = (a_0, a_1, \dots, a_{m-1}) \quad \underline{g} = (y_0, y_1, \dots, y_n)$$

$$f_j(x) = x^j$$

$$F^T = \begin{vmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ \vdots & \vdots & \dots & \vdots \\ x_1^{m-1} & \dots & \dots & x_n^{m-1} \end{vmatrix}$$

$$F^T F = \begin{vmatrix} n & \sum x_i & \sum x_i^2 & \dots & \sum x_i^{m-1} \\ \sum x_i & \sum x_i^2 & & & \vdots \\ \sum x_i^2 & & & & \vdots \\ \vdots & & & & \vdots \\ \sum x_i^{m-1} & & & \dots & \sum x_i^{2m-2} \end{vmatrix}$$

The "normal" matrix

$$F^T \underline{g} = \begin{vmatrix} \sum y_i \\ \sum x_i y_i \\ \vdots \\ \sum x_i^{m-1} y_i \end{vmatrix}$$

At this stage we obtain the parameters $\underline{a} = (F^T F)^{-1} F^T \underline{g}$

FITTING EXPONENTIALS:

$$f(x_i) = \sum_{j=0}^{m-1} a_j (s_j)^{x_i} \quad \underline{g} = (y_0, y_1, \dots, y_{n-1})$$

$$f_j(x) = (s_j)^x$$

$$F^T = \begin{pmatrix} x_0 & x_1 & \dots & x_{n-1} \\ s_0 & s_0 & \dots & s_0 \\ x_0 & x_1 & \dots & x_{n-1} \\ s_1 & s_1 & \dots & s_1 \\ \vdots & \vdots & \ddots & \vdots \\ x_0 & x_1 & \dots & x_{n-1} \\ s_{m-1} & s_{m-1} & \dots & s_{m-1} \end{pmatrix}$$

$$(F^T F)_{ij} = \sum_{k=0}^{n-1} (s_i s_j)^{x_k} \quad \text{Now suppose we have regular intervals } x_k = k \tau$$

$$\text{Let } s_i^! = s_i^\tau \quad \text{and } s_j^! = s_j^\tau$$

$$(F^T F)_{ij} = \sum_{k=0}^{n-1} (s_i s_j)^{k\tau} = \sum_{k=0}^{n-1} (s_i^! s_j^!)^k = \frac{1 - (s_i^! s_j^!)^n}{1 - s_i^! s_j^!}$$

$$\text{Next take the special case: } s_a^! = e^{(2\pi i/n)a}, \quad w = e^{-(2\pi j/n)}$$

$$(F^T F)_{ij} = 0 \text{ for } i+j \neq 0 \text{ or } n \quad (F^T F)_{ij} = n \text{ for } i+j = 0 \text{ or } n$$

$$(F^T F) = n \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & \dots & 1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & 0 & \dots & 0 & 0 \end{pmatrix} \quad (F^T F)^{-1} = \frac{1}{n} \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & 0 & \dots & 1 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & 0 & \dots & 0 & 0 \end{pmatrix}$$

$$\underline{a} = (F^T F)^{-1} \times \begin{pmatrix} \sum w^{-0k} y_k \\ \sum w^{-1k} y_k \\ \vdots \\ \sum w^{-(n-1)k} y_k \end{pmatrix} = \frac{1}{n} \begin{pmatrix} \sum w^{0k} y_k \\ \sum w^{1k} y_k \\ \vdots \\ \sum w^{(n-1)k} y_k \end{pmatrix}$$

Where we used $w^{(n-a)k} = w^{-ak}$. Note: \underline{a} is the discrete F.T. of \underline{g} .

APPLICATION TO FITTING NON-HARMONICALLY RELATED SINES AND COSINES:

Assume regular sample intervals: $x_i = i\tau$. Let $s_j = 2\pi f_j \tau$, where f_j are the frequencies of the various components. The s_j should be non-zero, positive and unique.

$$f(x_i) = a_0 + \sum_{j=1}^m a_j \cos(s_j i) + \sum_{j=1}^m b_j \sin(s_j i)$$

$$F^T = \begin{vmatrix} 1 & 1 & 1 & & 1 \\ 1 & \cos(s_1) & \cos(2s_1) & \dots & \cos((n-1)s_1) \\ 1 & \cos(s_2) & \cos(2s_2) & \dots & \cos((n-1)s_2) \\ & \dots & & & \dots \\ 1 & \cos(s_m) & \cos(2s_m) & \dots & \cos((n-1)s_m) \\ 0 & \sin(s_1) & \sin(2s_1) & \dots & \sin((n-1)s_1) \\ 0 & \sin(s_2) & \sin(2s_2) & \dots & \sin((n-1)s_2) \\ & \dots & & & \dots \\ 0 & \sin(s_m) & \sin(2s_m) & \dots & \sin((n-1)s_m) \end{vmatrix}$$

Now note that $2 \cos A \cos B = \cos(A+B) + \cos(A-B)$
 $2 \sin A \cos B = \sin(A+B) + \sin(A-B)$
 $2 \sin A \sin B = \cos(A-B) - \cos(A+B)$ and let $s_0=0$

$$(F^T F)_{i,j} = \sum_{k=0}^{n-1} \cos(s_i k) \cos(s_j k) = (1/2) \sum \cos((s_i + s_j)k) + \sum \cos((s_j - s_i)k)$$

for $0 \leq i \leq m$ and $0 \leq j \leq m$

$$(F^T F)_{i,j+m} = \sum_{k=0}^{n-1} \cos(s_i k) \sin(s_j k) = (1/2) \sum \sin((s_i + s_j)k) + \sum \sin((s_j - s_i)k)$$

for $0 \leq i \leq m$ and $1 \leq j \leq m$

$$(F^T F)_{i+m,j+m} = \sum_{k=0}^{n-1} \sin(s_i k) \sin(s_j k) = (1/2) \sum \cos((s_j - s_i)k) - \sum \cos((s_i + s_j)k)$$

for $1 \leq i \leq m$ and $1 \leq j \leq m$

The other terms can be found using the symmetry of $(F^T F)$.

Now
$$\sum_{k=0}^{n-1} e^{jwk} = (1 - e^{jwn}) / (1 - e^{jw})$$

$$= (e^{jwn/2} / e^{jw/2}) (e^{jwn/2} - e^{-jwn/2}) / (e^{jw/2} - e^{-jw/2})$$

$$= e^{jw(n-1)/2} \sin(nw/2) / \sin(w/2)$$

Now since $\cos(A) = \text{Re} (e^{jA})$ and $\sin(A) = \text{Im} (e^{jA})$:

$$\sum_{k=0}^{n-1} \cos(wk) = \cos(w(n-1)/2) \sin(nw/2) / \sin(w/2)$$

$$= (1/2) (\sin(w/2) + \sin((2n-1)w/2)) / \sin(w/2)$$

$$= (1/2) (1 + \sin((2n-1)w/2) / \sin(w/2))$$

unless $w=0$ in which case the sum is n .

$$\sum_{k=0}^{n-1} \sin(wk) = \sin(w(n-1)/2) \sin(nw/2) / \sin(w/2)$$

$$= (1/2) (\cos(w/2) - \cos((2n-1)w/2)) / \sin(w/2)$$

unless $w=0$ in which case the sum is 0.

$$(F^T F)_{i,j} = \frac{1}{4} \left(2 + \frac{\sin((2n-1)(s_i+s_j)/2)}{\sin((s_i+s_j)/2)} + \frac{\sin((2n-1)(s_i-s_j)/2)}{\sin((s_i-s_j)/2)} \right)$$

for $0 \leq i \leq m$ and $0 \leq j \leq m$ and $i \neq j$. If $i=j$, the third term is $2n-1$.

for $i=j=0$, the second and third term become $2n-1$.

$$(F^T F)_{i,j+m} = \frac{1}{4} \left(\frac{\cos((s_i+s_j)/2) - \cos((2n-1)(s_i+s_j)/2)}{\sin((s_i+s_j)/2)} + \frac{\cos((s_j-s_i)/2) - \cos((2n-1)(s_j-s_i)/2)}{\sin((s_j-s_i)/2)} \right)$$

for $0 \leq i \leq m$ and $1 \leq j \leq m$ and $i \neq j$. If $i=j$, the second term is 0.

$$(F^T F)_{i+m,j+m} = \frac{1}{4} \left(\frac{\sin((2n-1)(s_j-s_i)/2)}{\sin((s_j-s_i)/2)} - \frac{\sin((2n-1)(s_j+s_i)/2)}{\sin((s_i+s_j)/2)} \right)$$

for $1 \leq i \leq m$ and $1 \leq j \leq m$ and $i \neq j$. If $i=j$ the first term is $2n-1$.

The first element in this array is n , the other diagonals are near $n/2$, while most of the other terms are small, of the order of 1. The only large elements will be the result of two narrowly separated frequencies. This makes for good numerical stability when inverting $(F^T F)$ by the simplest methods.

When the frequencies are harmonically related, we have $s_i = (2\pi i/n)$. Then all off-diagonal terms will be 0, and those on the diagonal will be exactly $n/2$ except the first which will be n . The inverse of $(F^T F)$ then is also diagonal with the first element $1/n$ and the rest $2/n$. We are back to discrete fourier transforms in this case.

Note that if we use:

$$\sin(A+B) = \sin A \cos B + \cos A \sin B$$

$$\sin(A-B) = \sin A \cos B - \cos A \sin B$$

$$\cos(A+B) = \cos A \cos B - \sin A \sin B$$

$$\cos(A-B) = \cos A \cos B + \sin A \sin B$$

we can obtain all the entries in the array using only a few operations on $\sin(s_i/2)$, $\cos(s_i/2)$ and $\sin((2n-1)s_i/2)$, $\cos((2n-1)s_i/2)$.

SOLVING SETS OF SIMULTANEOUS LINEAR DIFFERENCE EQUATIONS:

$$(x_1)_{n+1} = a_{11} (x_1)_n + a_{12} (x_2)_n \quad \dots \quad + a_{1m} (x_m)_n$$

$$(x_2)_{n+1} = a_{21} (x_1)_n + a_{22} (x_2)_n \quad \dots \quad + a_{2m} (x_m)_n$$

...

$$(x_m)_{n+1} = a_{m1} (x_1)_n + a_{m2} (x_2)_n \quad \dots \quad + a_{mm} (x_m)_n$$

$$\underline{x}_{n+1} = A \underline{x}_n \quad \text{where } A \text{ is the given coefficient set}$$

Assume a solution of the form r^n for each x_i :

$$\underline{x}_n = \underline{a} r^n \quad \text{where } \underline{a} \text{ is a } m\text{-vector of parameters}$$

$$r(\underline{a}r^n) = A(\underline{a}r^n) \quad (A - Ir)\underline{a} = 0 \quad \text{since } r^n \neq 0$$

A non-zero solution for \underline{a} requires that $\det(A - Ir) = 0$. The possible \underline{a} 's are eigenvectors, the possible r 's are eigenvalues of the matrix A . The determinant is a polynomial of degree m in r and will usually have m solutions, possibly complex. We get the usual problems if two roots coincide and have to introduce additional solutions of the form nr^n , n^2r^n and so on. Having found r , we can solve for \underline{a} using some normalising conditions (since any multiple will also be a solution). Then using the linearity of the set of equations we can add up the solutions into a more general one:

$$\underline{x}_n = \sum_{j=1}^m \underline{a}_j (r_j)^n \quad \text{where } r_j \text{ are the various solutions}$$

Often we are only interested in stability and just check the roots:

$$|r_j| < 1$$

EXAMPLE FOR A TWO VARIABLE SYSTEM:

$$(x_1)_{n+1} = a_{11}(x_1)_n + a_{12}(x_2)_n$$

$$(x_2)_{n+1} = a_{21}(x_1)_n + a_{22}(x_2)_n$$

$$\det \begin{vmatrix} (a_{11} - r) & a_{12} \\ a_{21} & (a_{22} - r) \end{vmatrix} = (a_{11} - r)(a_{22} - r) - a_{12}a_{21} = 0$$

$$(a_{11}a_{22} - a_{12}a_{21}) - (a_{11} + a_{22})r + r^2 = 0$$

$$r = (1/2) \left((a_{11} + a_{22}) \pm \sqrt{(a_{11} + a_{22})^2 - 4(a_{11}a_{22} - a_{12}a_{21})} \right)$$

$$r = (1/2) \left((a_{11} + a_{22}) \pm \sqrt{(a_{11} - a_{22})^2 + 4a_{12}a_{21}} \right)$$

Stability: When is $|(a \pm \sqrt{a^2 - 4b})| < 2$?

Case 1: $4b > a^2$ Complex roots $|a| < 2$ for stability

Case 2: $4b < a^2$ Real roots $a > 0$. $a + \sqrt{a^2 - 4b} < 2$

$$a < b + 1$$

Case 3: $4b < a^2$ Real roots $a < 0$. $-a + \sqrt{a^2 - 4b} < 2$

$$-a < b + 1$$

Case 2 & 3 $4b < a^2$ Real roots $|a| < b + 1$ for stability

Substituting:

$$\text{Case 1: } 4(a_{11}a_{22} - a_{12}a_{21}) > (a_{11} + a_{22})^2$$

$$-4a_{12}a_{21} > (a_{11} - a_{22})^2$$

$$|a_{11} + a_{22}| < 2$$

Case 2 & 3: Opposite of above relations

$$|a_{11} + a_{22}| < 1 + (a_{11}a_{22} - a_{12}a_{21})$$

MULTI-DIMENSIONAL NEWTON-RAPHSON ZERO-FINDING:

Suppose we have n functions F_i each of n parameters a_j . We are trying to find values for the a_j 's such that the F_i 's are all zero.

Assume we have the value \underline{a}_n at step n for the parameter vector.

This gives us the value for the function vector $\underline{F}_n = F(\underline{a}_n)$.

Now consider a small change $d\underline{a}$ in \underline{a} . To a first approximation we get:

$$F(\underline{a}_n + d\underline{a}) = F(\underline{a}_n) + F'(\underline{a}_n) d\underline{a}$$

Where $F'(\underline{a}_n)$ is the matrix of derivatives:

$$\begin{pmatrix} \frac{dF_1}{da_1} & \frac{dF_1}{da_2} & \dots & \frac{dF_1}{da_n} \\ \frac{dF_2}{da_1} & \frac{dF_2}{da_2} & \dots & \frac{dF_2}{da_n} \\ \dots & \dots & \dots & \dots \\ \frac{dF_n}{da_1} & \frac{dF_n}{da_2} & \dots & \frac{dF_n}{da_n} \end{pmatrix}$$

For $F(\underline{a}_n + d\underline{a}) = 0$ we need $-F(\underline{a}_n) = F'(\underline{a}_n) d\underline{a}$

So $d\underline{a} = -(F'_n)^{-1} F_n$

$$\underline{a}_{n+1} = \underline{a}_n - (F'_n)^{-1} F_n$$

So we iterate to a solution, requiring one matrix inversion per step. There are better methods, but few simpler. For bad hill-climbing type problems one can use the method of conjugate directions and various variations such as the so-called mixed method which is also fairly rapid.

SIMPLE INTERPOLATION OF A FUNCTION FROM A STORED GRID:

Rectangular grid: Suppose the origin is x_0, y_0 and the spacing d .

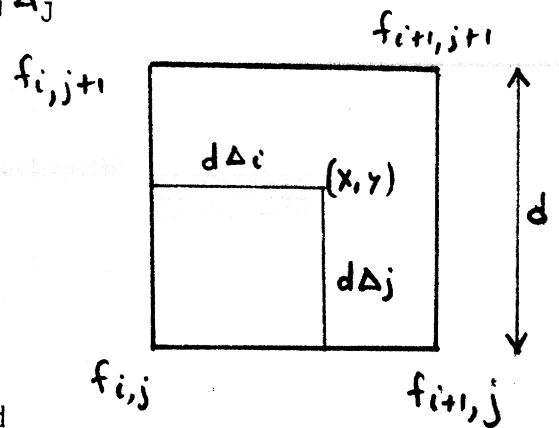
To find an interpolated value at a point x, y calculate as follows:

$$x' = (x - x_0)/d \quad y' = (y - y_0)/d$$

$$i = [x'] \quad j = [y']$$

$$\Delta i = x' - i \quad \Delta j = y' - j$$

$$f(x,y) \approx f_{i,j} (1-\Delta i)(1-\Delta j) + f_{i+1,j} \Delta i(1-\Delta j) \\ + f_{i,j+1} (1-\Delta i)\Delta j + f_{i+1,j+1} \Delta i \Delta j$$



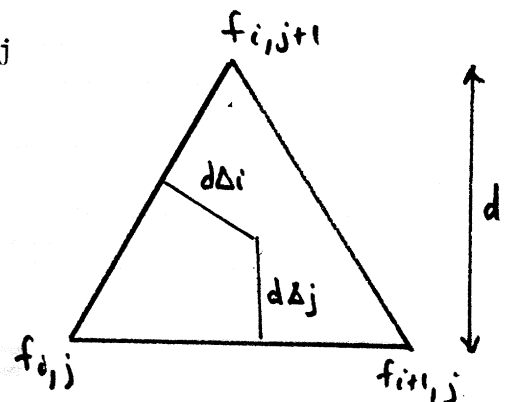
Triangular grid: Again origin at x_0, y_0

$$x' = \left(\frac{\sqrt{3}}{2}(x-x_0) - \frac{1}{2}(y-y_0)\right)/d \quad y' = (y-y_0)/d$$

$$i = [x'] \quad j = [y']$$

$$\Delta i = x' - i \quad \Delta j = y' - j$$

$$f(x,y) \approx f_{i,j} (1-\Delta i - \Delta j) + f_{i+1,j} \Delta i + f_{i,j+1} \Delta j$$



WHAT ELLIPSE IS IT:

Given an ellipse in the form:

$$A x^2 + B xy + C y^2 + D x + E y + F = 0$$

Determine its center, angular orientation and major and minor axes.

$$x_0 = (BE - 2CD)/(B^2 - 4AC) \quad y_0 = (2EA - DB)/(B^2 - 4AC)$$

x_0, y_0 are the center because we can expand as follows:

$$A(x-x_0)^2 + B(x-x_0)(y-y_0) + C(y-y_0)^2 + F' = 0$$

$$\begin{aligned} A x^2 + B xy + C y^2 &+ (-2Ax_0 - By_0)x + (-2Cy_0 - Bx_0)y \\ &+ (F' + Ax_0^2 + Bx_0y_0 + Cy_0^2) = 0 \end{aligned}$$

So we have:

$$2Ax_0 + By_0 = -D$$

$$Bx_0 + 2Cy_0 = -E$$

Solving this set of equations we get the above expression for x_0, y_0 .

We also now have a useful new quantity:

$$F' = F - (Ax_0^2 + Bx_0y_0 + Cy_0^2)$$

The orientation of the ellipse is found as follows:

$$\tan 2\theta = B/(A-C)$$

And the major and minor axes can be found as well:

$$a^2 = -2F' / ((A+C) - \sqrt{B^2 + (A-C)^2})$$

$$b^2 = -2F' / ((A+C) + \sqrt{B^2 + (A-C)^2})$$

These last results follow from expansion after change of coordinates:

$$\left(\frac{x \cos \theta + y \sin \theta}{a}\right)^2 + \left(\frac{-x \sin \theta + y \cos \theta}{b}\right)^2 = 1$$

$$\left(\frac{\sin \theta}{b^2} + \frac{\cos \theta}{a^2}\right) x^2 + 2 \sin \theta \cos \theta \left(\frac{1}{a^2} - \frac{1}{b^2}\right) xy + \left(\frac{\sin \theta}{a^2} + \frac{\cos \theta}{b^2}\right) y^2 = 1$$

Identifying the appropriate terms with A, B, C and F' we get:

$$B/(-F') = \sin 2\theta \left(\frac{1}{a^2} - \frac{1}{b^2}\right)$$

$$(A-C)/(-F') = \cos 2\theta \left(\frac{1}{a^2} - \frac{1}{b^2}\right)$$

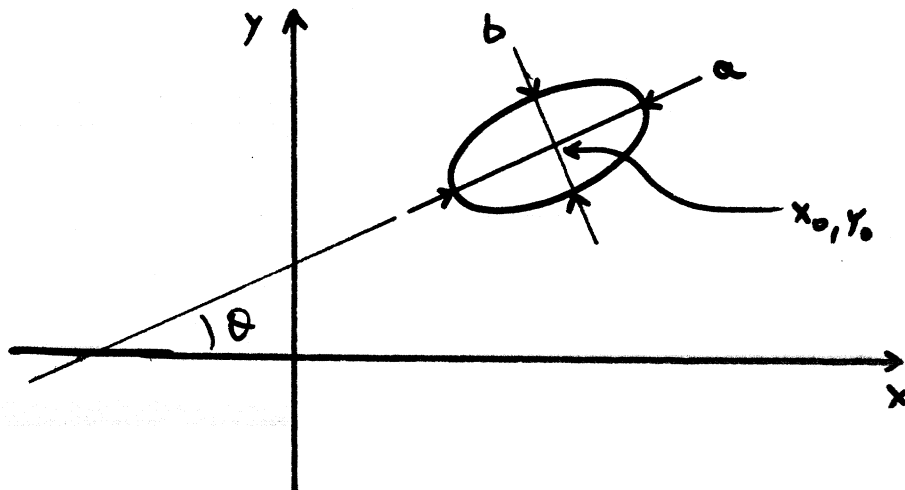
Since $2 \sin \theta \cos \theta = \sin 2\theta$ and $(\cos \theta)^2 - (\sin \theta)^2 = \cos 2\theta$

$$(A+C)/(-F') = \left(\frac{1}{a^2} + \frac{1}{b^2}\right)$$

It's also clear that:

$$\left(\frac{1}{a^2} - \frac{1}{b^2}\right) = \sqrt{B^2 + (A-C)^2} / (+F') \quad (\text{Assuming } a > b)$$

The rest follows from these simple equations.



APPROXIMATION TO $n!$

Stirling's formula: $n! \approx \sqrt{2\pi n} (n/e)^n$

Better approximation: $n! \approx \sqrt{2\pi n} (n/e)^n e^{(1/12n)}$

The fractional error of the latter is much smaller. For $n=10$ for example it is $.27E-5$ versus $.8E-2$ and for $n=50$ it is $.22E-7$ versus $.1E-3$. This is useful in calculating large binomial coefficients.

OBTAINING A NORMALLY DISTRIBUTED RANDOM VARIABLE FROM ONE UNIFORMLY DISTRIBUTED:

Suppose x_j is the output of our random (pseudo ...) number generator.

$$1. \quad \left(\sum_{i=1}^{n+1} x_i - 6 \right) / 6$$

$$2. \quad \sqrt{-2 \log x_i} \sin(2\pi x_{i+1})$$

3. Let $f(x)$ be the distribution we are aiming for. Now integrate it:

$$F(x) = \int_{-\infty}^x f(t) dt$$

A random variable distributed as desired will be $(F^{-1})(x_j)$

MULTIPLICATIVE RANDOM GENERATORS:

$$x_{i+1} = A^k x_i \pmod{p} \quad p \text{ a large prime}$$

A a primitive root of p , k not a factor of $p-1$.

$$\text{Example: } p = 2^{35} - 31 \quad A = 5 \quad k = 5$$

$$p = 2^{31} - 1 \quad A = 7 \quad k = 5$$

Additive congruential generators are better, though.

FAST IN-POSITION MATRIX INVERSE:

```

Do i = 0 ( 1 ) n-1
com ← a(i,i)
a(i,i) ← 1.

    Do j = 0 ( 1 ) n-1
    a(i,j) ← a(i,j) / com
    End

    Do k = 0 ( 1 ) n-1 and i ≠ k
    com ← a(k,i)
    a(k,i) ← 0.

        Do j = 0 ( 1 ) n-1
        a(k,j) ← a(k,j) - a(i,j) * com
        End

    End

End

End

```

Note that rows and columns are never shuffled and that there will be matrices which while not singular will cause this procedure to fail.

The matrix is n by n and stored in the array $a(i,j)$ where i and j range from 0 to $n-1$.

GENERATING A BIT-REVERSING TABLE:

```

b(0) ← 0
m ← 1

Do i = 0 ( 1 ) ln-1
    Do j = 0 ( 1 ) m-1
    b(j) ← b(j)*2
    b(j+m) ← b(j)+1
    End
    m ← m*2
End

```

SOME FOURIER TRANSFORM METHODS FOR IMAGES:

Fourier transforms for images are two-dimensional and two-sided. In this they differ from time-series type transforms which are one-dimensional and often pertain to one-sided functions (impulse responses must be 0 for negative values of time).

The general formula for n-dimensions is: (Note: f and g are complex)

$$g(\underline{u}) = \frac{1}{(2\pi)^{n/2}} \int_{\underline{v}} f(\underline{x}) e^{-i \underline{u} \cdot \underline{x}} d\underline{x}$$

$$f(\underline{x}) = \frac{1}{(2\pi)^{n/2}} \int_{\underline{v}} g(\underline{u}) e^{+i \underline{u} \cdot \underline{x}} d\underline{u}$$

Where \underline{x} is the n-dimensional source-space vector, \underline{u} is the n-dimensional transform-space vector and g is the transform of f. For two dimensions:

$$g(u,v) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-i(ux+vy)} dx dy$$

$$f(x,y) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(u,v) e^{+i(ux+vy)} du dv$$

Many functions of interest are rotationally symmetric and can be dealt with by use of the one-dimensional integrals obtained after introducing the polar coordinates (r, θ) for (x, y) and (ρ, ϕ) for (u, v) .

$$g(\rho) = \int_0^{\infty} f(r) r J_0(r\rho) dr$$

$$f(r) = \int_0^{\infty} g(\rho) \rho J_0(r\rho) d\rho$$

Where J_0 is the zeroth order Bessel function.
Note that f and g are now real-valued.

This follows from:

$$\begin{aligned} & \frac{1}{2\pi} \int_0^{\infty} \int_0^{2\pi} f(r) e^{-i(r\rho \cos \phi \cos \theta + r\rho \sin \phi \sin \theta)} d\theta dr \\ &= \frac{1}{2\pi} \int_0^{\infty} f(r) \int_0^{2\pi} e^{-i r\rho \cos(\phi - \theta)} d\theta dr \\ &= \frac{1}{2\pi} \int_0^{\infty} f(r) 2\pi r J_0(r\rho) dr \end{aligned}$$

We can apply these results to a few useful examples:

The pillbox: $f(r) = 1$ for $r \leq R$, 0 otherwise. This is the point-spread function produced by defocusing.

$$\begin{aligned} g(\rho) &= \int_0^R r J_0(r\rho) dr = \frac{1}{\rho^2} \int_0^{R\rho} x J_0(x) dx = \frac{R\rho}{\rho^2} J_1(R\rho) \\ g(\rho) &= R^2 \frac{J_1(R\rho)}{(R\rho)} \quad \text{Since } \int_0^{\infty} x J_0(x) dx = x J_1(x) \end{aligned}$$

So the function $J_1(x)/x$ plays the role here that $\sin x/x$ plays for one-dimensional systems.

The gaussian:

$$f(r) = e^{-\frac{1}{2}\left(\frac{r}{\sigma}\right)^2}$$

$$g(\rho) = \int_0^{\infty} e^{-\frac{1}{2}\left(\frac{r}{\sigma}\right)^2} r J_0(r\rho) dr = \sigma^2 e^{-\frac{1}{2}(\sigma\rho)^2}$$

$$\text{Since } \int_0^{\infty} e^{-\alpha x^2} J_n(bx) x^{n+1} dx = \frac{b}{(2\alpha)^{n+1}} e^{-\frac{b^2}{4\alpha}}$$

The gaussian has some interesting properties. First it is the only rotationally symmetric function that can be factored into a product of a function of x and a function of y . Secondly it is the only function that "transforms into itself".

The gaussian is also a good first approximation to point-spread functions in some devices(at least for small r).

A scatter function: $f(r) = e^{-\sigma r} / r$

Analysis of total reflections in the face-plate of an imaging device leads to an equation of this form (at least for large r).

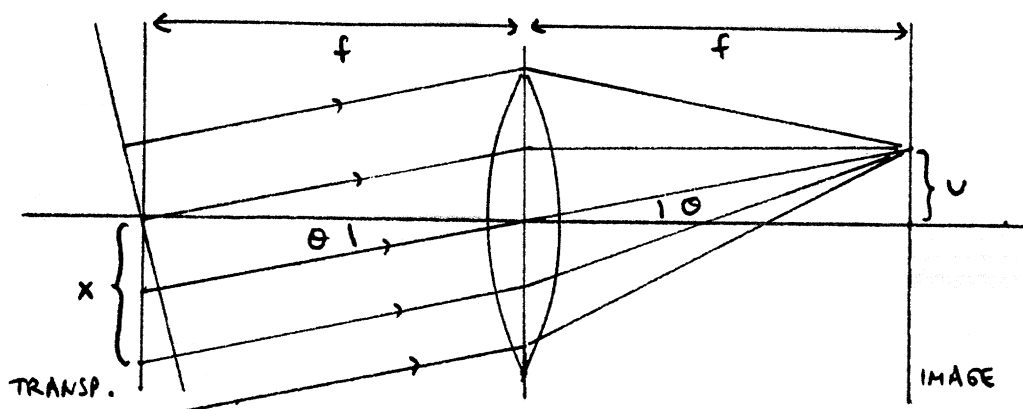
$$\begin{aligned} g(\rho) &= \int_0^{\infty} e^{-\sigma r} J_0(r\rho) r/r dr \\ &= \int_0^{\infty} e^{-\sigma r} J_0(r\rho) dr = \frac{1}{\sqrt{\sigma^2 + \rho^2}} \end{aligned}$$

Note on scaling: For the gaussian we have the following relationship:

$$r_H \rho_H = 1.386 \quad (r_H = 1.177 \sigma, \rho_H = 1.177 / \sigma)$$

where r_H is the half-intensity radius in the source space,
and ρ_H is the half-intensity radius in the transform space.

HOW COHERENT MONOCHROMATIC LIGHT AND A LENS DO FOURIER TRANSFORMS:



Let f be the focal length of the lens and λ the wavelength of light. Plane monochromatic coherent light enters from the left and passes through the transparency, being then focused by the lens on the image plane. We assume that x and u are relatively small compared to f , so that θ will be small. We then have for the distance that the ray has to travel from the point x on the source plane to the point u on the image plane:

$$f/\cos \theta + f \cos \theta + x \sin \theta = f(2 + \theta^4/4 + \theta^6/120 \dots) + x(\theta - \theta^3/6 \dots)$$

For small θ this is approximately: $2f + x \theta$

The phase-shift in radians is then: $(2f + x \theta) * 2\pi/\lambda$

We can ignore the constant part of this and considering that light will arrive at the point u from all over the transparency we get:

$$g(u) = \int_{-\infty}^{+\infty} f(x) e^{2\pi i \frac{x u}{\lambda f}} dx$$

Where $f(x)$ is the amount of light passed through at the point x .

Now extend this to two dimensions and we finally have:

$$g(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{2\pi i \left(\frac{xu + yv}{\lambda f} \right)} dx dy$$

Note that $g(u,v)$ is complex. We can get an idea of scaling from this equation.

SOME HEURISTICS FOR TELLING WHAT HAPPENS WHEN YOU TRANSFORM A FUNCTION

Source domain	Transform domain
Periodic	Discrete (non-zero only for some f)
Symmetric (about 0)	Real
Non-zero for finite distance	Non-zero out to infinity
Compact	Spread-out
Sharp transitions	Lots of high frequency components
Sample of $f(t)$	Periodic copies of $F(w)$
Sum of $f(t)$ and $g(t)$	Sum of $F(w)$ and $G(w)$
Convolution of $f(t)$ and $g(t)$	Product of $F(w)$ and $G(w)$
Time shift of $f(t)$ by	Multiply $F(w)$ by e^{jw}
Integral of $f(t)$	Divide by jw
Differential of $f(t)$	Multiply by jw

These rules apply going either way in the transformation and may be used simultaneously. Discrete fourier transforms, for example, are both periodic and discrete in both domains.

MEASURING MODULATION TRANSFER FUNCTION USING SQUARE WAVES:

It is very hard to produce images in which the intensity varies sinusoidally. Yet such images are required in the traditional determination of frequency response or modulation transfer function. An alternative is the use of simple-to-produce square wave intensity modulated images. Then however we have to recover the transfer function from the measured results.

Let t be one of the spacial dimensions and $\omega = (2\pi)/T$, where T is the repetition interval. The input can be analysed into:

$$f(t) = 1/2 + \sum_1^{\infty} b(n) \cos n\omega t \quad \text{where } b(n) = \frac{(2/\pi n)(-1)^{\frac{n-1}{2}}}{2} \text{ for } n \text{ odd, } 0 \text{ otherwise}$$

Let the transfer function be $a(\omega)$. Then the output will be:

$$g(t) = (1/2)a_0 + \sum_1^{\infty} b(n) a(n\omega) \cos n\omega t$$

We can easily normalise to let $a_0 = 1$. Let $c(\omega) = \sum_1^{\infty} b(n)a(n\omega)$.

The problem is to recover $a(\omega)$ from $c(\omega)$. In the case of square-waves:

$$c(\omega) = (2/\pi) (a(\omega) - \frac{1}{3}a(3\omega) + \frac{1}{5}a(5\omega) - \frac{1}{7}a(7\omega) + \frac{1}{9}a(9\omega) - \frac{1}{11}a(11\omega) \dots)$$

$$c(3\omega) = (2/\pi) (a(3\omega) - \frac{1}{3}a(9\omega) + \frac{1}{5}a(15\omega) \dots)$$

$$c(5\omega) = (2/\pi) (a(5\omega) - \frac{1}{3}a(15\omega) \dots)$$

$$c(7\omega) = (2/\pi) (a(7\omega) \dots)$$

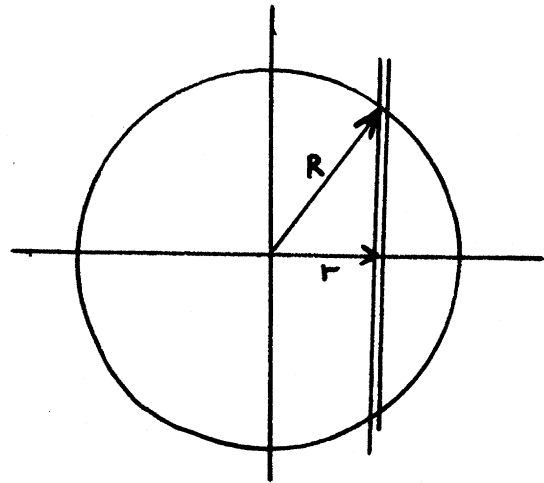
$$c(9\omega) = (2/\pi) (a(9\omega) \dots)$$

Now add appropriate high-order terms to $c(\omega)$ to cancel out high-order terms of $a(\omega)$ and get:

$$a(\omega) \cong (\pi/2) (c(\omega) + \frac{1}{3}c(3\omega) - \frac{1}{5}c(5\omega) + \frac{1}{7}c(7\omega) + \frac{1}{11}c(11\omega) - \frac{1}{13}c(13\omega) \dots)$$

CONVOLUTIONS OF PILL-BOXES:

With a line:



Clearly the convolution is $2 \sqrt{R^2 - r^2} = 2 R \sqrt{1 - \left(\frac{r}{R}\right)^2}$ for $|r| < R$

This then gives us the intensity profile of a defocused line.

Convolution of a pill-box with a step:

We simply integrate the above:

$$\int_{-R}^r 2 \sqrt{R^2 - x^2} dx$$

$$= \left[x \sqrt{R^2 - x^2} + R^2 \sin^{-1} \left(\frac{x}{R} \right) \right]_{-R}^r$$

$$F(r) = \pi R^2 \left(\frac{1}{2} + \frac{1}{\pi} \sin^{-1} \left(\frac{r}{R} \right) + \frac{1}{\pi} \left(\frac{r}{R} \right) \sqrt{1 - \left(\frac{r}{R} \right)^2} \right)$$

So we have the intensity profile of a defocused edge.

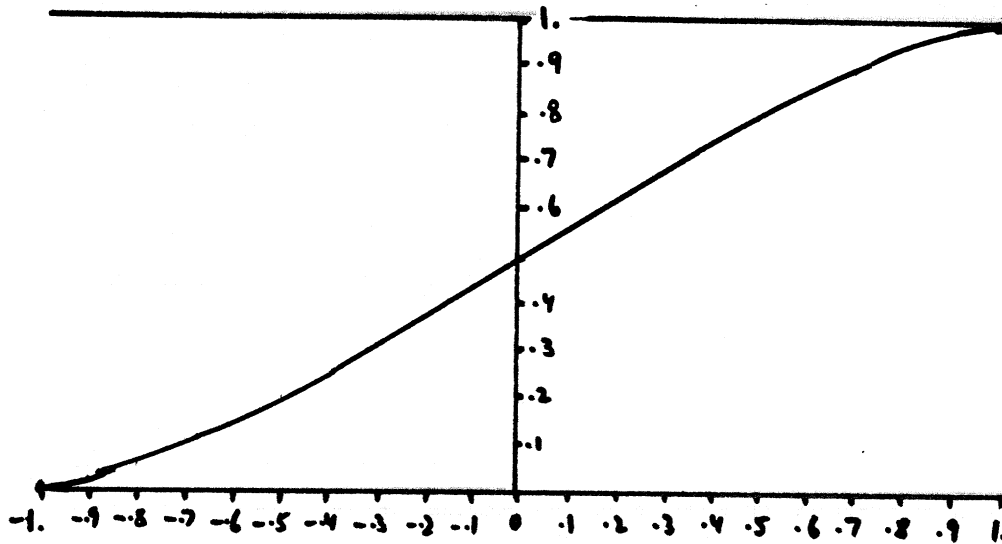
This can be rewritten in a slightly different form using:

$$\sin^{-1} x + \cos^{-1} x = \frac{\pi}{2}$$

We can also use this to find the convolution of two pillboxes as

$$2 F \left(- \frac{|r|}{2} \right)$$

WHAT A DEFOCUSSED EDGE LOOKS LIKE:



Vertical: relative intensity, horizontal: (distance from edge/defocus radius)

Central slope: $2/(\pi R)$, 10% to 90% distance = $1.38 R$

Derivative is $(2/\pi R) \sqrt{1 - (r/R)^2}$

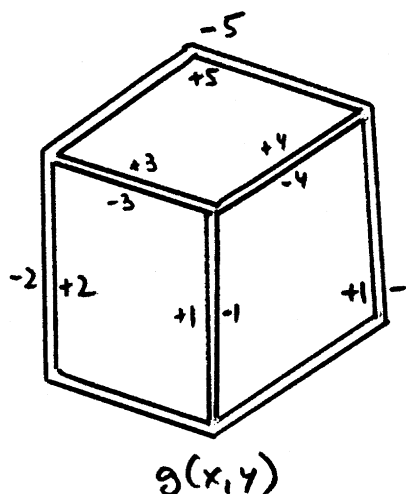
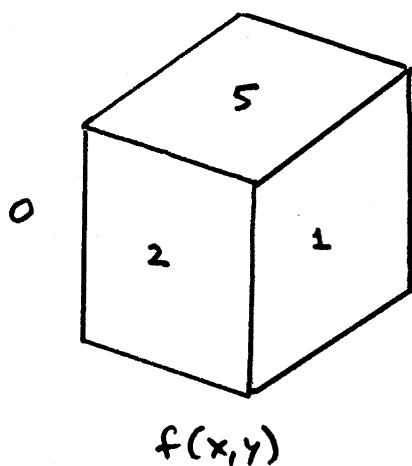
A LINEAR THEORY OF FEATURE POINT MARKING

A first step in many line-finding programs is a process for determining which points in the image are likely to be on an edge. This is usually done by locating areas of rapid intensity variations. Various ad hoc linear and non-linear techniques of varying support in the image are brought into play. It would be useful to have an anchor point on this spectrum of possible procedures. Since a lot is known about linear methods we might ask what linear method applied to a somewhat idealised image would do the job.

Given a function $f(x,y)$ which is constant within polygonal areas in the image, we are looking for a convolution function $h(x,y)$ which when applied to $f(x,y)$ will be zero everywhere except on the edges.

$$g(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x-x',y-y') h(x',y') dx' dy'$$

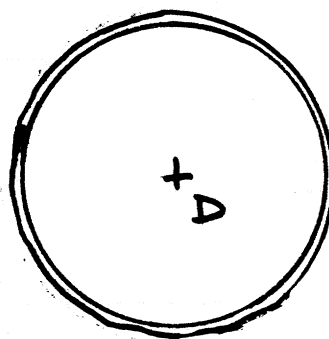
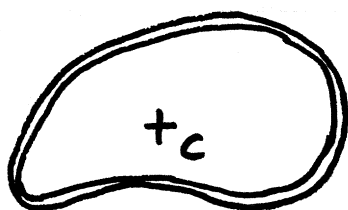
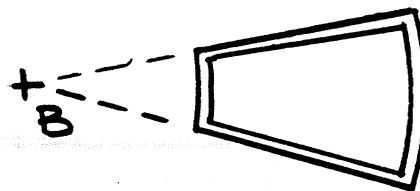
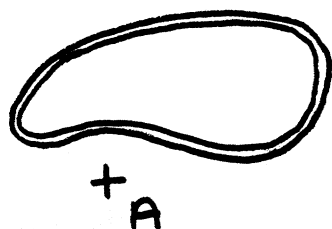
To attempt to answer this question we might start by asking what values we expect $g(x,y)$ to take on the edges. Linearity considerations dictate that it somehow be proportional to the intensity step. In addition it must reflect the orientation of the step, to insure that superposition will work. A combination of a negative and a positive pulse will do the trick, provided the area under each is equal to the intensity step. Since the image is actually two-dimensional we will have two pulse walls running along each edge.



Note, by the way, that the regions of uniform intensity don't have to be polygonal. Now it is pretty hard to guess what form $h(x,y)$ will take. A way to get a handle on this is to ask the inverse question: what $h'(x,y)$ when convolved with $g(x,y)$ will produce $f(x,y)$?

$$f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x-x', y-y') h'(x', y') dx' dy'$$

Well, it helps to look at some simple cases first. In particular if we only have one contour (one closed curve made of the double pulsed wall) we expect to get 0 if the convolution is about a point outside this contour, and the intensity step if the point is inside the contour.



A and C illustrate the above statements, while B and D are special cases useful for deriving equations. From D in particular we find that $2\pi r (d/dr) h'(r) = 1$. This is assuming that h must be rotationally symmetric which is clear from the other examples. We also noted that convolving with the double pulse wall is just like taking the derivative.

$$h'(r) = (1/2\pi) \int (1/r) dr = - (1/2\pi) \log r$$

Since this function also does the right thing for example B we have the desired result. Now we need to find $h(x,y)$ from this. We do this by finding the fourier transform of $h'(x,y)$ and noting that it must be the algebraic inverse of the transform of $h(x,y)$. Since the functions are rotationally symmetric we get:

$$H'(\rho) = -(1/2\pi) \int_0^{\infty} \log r \cdot r J_0(r\rho) dr$$

Integrating by parts and using $\int x J_0(x) dx = x J_1(x)$ as well as $\int J_1(x) dx = -J_0(x)$ we obtain:

$$H'(\rho) = (1/2\pi\rho^2)$$

To obtain the transform of $h(x,y)$ we just invert this:

$$H(\rho) = 1/H'(\rho) = 2\pi\rho^2$$

When we try to inverse transform this we get into convergence difficulties and soon discover that we have to expand our universe to that of generalised functions if we expect to win, even if we use convergence factors. It then also becomes reasonable to guess at the answer. Consider the sequence of "functions" obtained by repeatedly differentiating a unit step. The first is a pulse at the origin, the second two pulses of opposite sign. This function corresponds to (d/dx) in the following sense: if we convolve it with a function $f(x)$ we obtain the derivative $f'(x)$. Similarly, the next member of this sequence consists of a negative, a double height positive and another negative pulse and corresponds to (d^2/dx^2) and so on.

When we try to transform these "functions" we obtain the following:
 $T(d/dx) = iu$, $T(d^2/dx^2) = -u^2$, $T(d/dy) = iv$, $T(d^2/dy^2) = -v^2$. And:

$$T(d^2/dx^2 + d^2/dy^2) = -u^2 - v^2 = -\rho^2 \quad !$$

So our $h(x,y)$ is some multiple of the laplacian $(d^2/dx^2 + d^2/dy^2)$.

One can amuse oneself by showing that the convolution of $h(x,y)$ and $h'(x,y)$ is in fact zero everywhere except at the origin as it ought to be:

$$h(x,y) \otimes h'(x,y) = (d^2/dx^2 + d^2/dy^2)(-1/2\pi) \log r$$

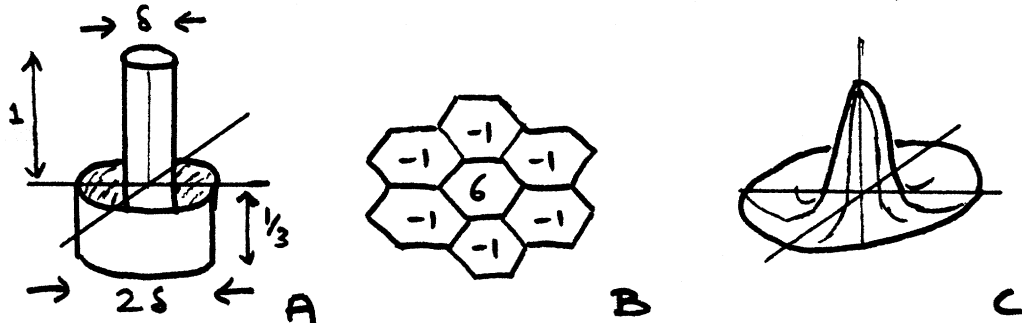
$$d/dx \log r = d/dx (1/2) \log(x^2+y^2) = x/(x^2+y^2)$$

$$d^2/dx^2 \log r = -(x^2-y^2)/(x^2+y^2)$$

$$d^2/dy^2 \log r = (x^2-y^2)/(x^2+y^2) \quad \text{by symmetry}$$

$$(d^2/dx^2 + d^2/dy^2) \log r = 0 \quad \text{except for } x=y=0$$

So $\log r$ is the function which has the surprising property of having a curvature at each point which is exactly opposite to the curvature at right angles. Next we might be interested in a discrete approximation to the laplacian, particularly a rotationally symmetric one (**B**):



Now since we have all this nice linear theory ala Wiener available we might as well mention that if the image is corrupted by gaussian spatially independent noise we can apply his results to produce a least squares approximation to $g(x,y)$. We then find our convolution functions more spread out than the laplacian and in fact they will contain a central peak surrounded by a larger negative depression (**C**). The only problem is that only some part of the noise in the image satisfies the criterion, a great deal of it not being spatially independent and what's worse, there is no reason to suppose that a least squares approximation to our pulse walls would be at all useful. Anyway, here it is, our anchor point for the spectrum of feature point (or inhomogeneous) finders.

FAST FOURIER TRANSFORM

Once we have bit-reversed the complex array x containing the function to be transformed we proceed as follows, assuming $\ln = \log_2 n$.

```

n ← 2↑ln
itn ← n/2
igr ← n/2
iga ← 2
is ← 1

Do i = 1 ( 1 ) ln
  Do ist = 0 ( iga ) n-1
    Do k = ist ( 1 ) ist+is-1 , iwb = 0 ( igr )
      a ← x(k+is)*w(iwb)+x(k)
      b ← x(k+is)*w(iwb+itn)+x(k)
      x(k) ← a
      x(k+is) ← b
    End
  End
  igr ← igr/2
  is ← iga
  iga ← iga*2
End

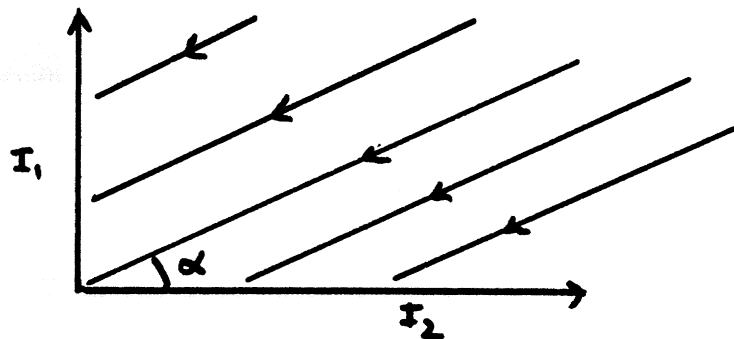
```

Where $w(a) = e^{\frac{2\pi ai}{n}}$

Note that the arrays x and w are complex valued and dimension n .

CONTRAST IN A RECTANGULAR CORNER:

One of the problems in generating line-drawings from complex scenes is that in addition to the contrast-reduction due to scatter in the imaging device there is also a great reduction in contrast due to mutual illumination. To get a handle on this problem, consider the simple case of two semi-infinite planes meeting at right-angles. The light is incident at an angle α w.r.t. one of the planes. The surface is such that ρ of the incident light is reflected. Clearly for any point on one of the half-planes one half is reflected into empty space, the rest onto the other surface. Light incident at any point is a sum of the light from the source and that reflected from the other plane. If both planes are semi-infinite the intensity on each one will be uniform since a point receives an amount of light from the other plane that does not depend on the position of the point.



$$I_1 = (\rho/2) I_2 + a \cos \alpha$$

$$I_2 = (\rho/2) I_1 + a \sin \alpha$$

$$I_1 = (\cos \alpha + (\rho/2) \sin \alpha) a / (1 - (\rho/2)^2)$$

$$I_2 = (\sin \alpha + (\rho/2) \cos \alpha) a / (1 - (\rho/2)^2)$$

$$\text{Contrast} = \left| \frac{I_1 - I_2}{I_1 + I_2} \right| = \left| \frac{((\rho/2-1)\cos\alpha - (\rho/2-1)\sin\alpha)}{((\rho/2+1)\cos\alpha + (\rho/2+1)\sin\alpha)} \right| = \frac{2-\rho}{2+\rho} \left| \frac{\cos\alpha - \sin\alpha}{\cos\alpha + \sin\alpha} \right|$$

$$\text{Contrast} = \left| \frac{I_1 - I_2}{I_1 + I_2} \right| = \frac{2 - \rho}{2 + \rho} \left| \tan(\alpha - \pi/4) \right|$$

In the absence of reflection this will be $\left| \tan(\alpha - \pi/4) \right|$, so the contrast is reduced by a factor

$$(2 - \rho) / (2 + \rho)$$

This factor ranges from 1/3 to 1 as ρ ranges from 1 to 0. So for objects that reflect most of the incident light, such as our white cubes this effect is worst, reducing the contrast by a factor 3.

If we consider finite half-planes things get more hairy and the intensity on a given plane is no longer independent of position, falling off as one goes outward from the corner. In the corner itself however the situation is unchanged in the limit. So as far as the contrast across the edge in the image is concerned we can still use the above formula. Note that with finite half-planes a rigorous analysis would require knowledge of the distribution of reflected light with angle which was not needed in the above.

If we consider other angles we find that the problem increases as the angle gets smaller.

Suppose the angle between the two planes is π/k instead of $\pi/2$. Then instead of $(\rho/2)$ we have $(1 - 1/k)\rho$. The reduction in contrast then is:

$$\frac{1 - (1 - 1/k)\rho}{1 + (1 - 1/k)\rho}$$

And when $\rho = 1$, the worst case, we have a reduction of $1/(2k-1)$.

It is clear that gray blocks are very much better in this respect than white ones. For example if $\rho = .5$ instead of 1.0, the reduction is only 3/5 instead of 1/3 for the contrast.

SCATTER IN OUR IMAGE DISSECTOR (TVC):

A considerable reduction in contrast in our image dissector is caused by scatter of the incident light. This scatter goes undetected when one concerns oneself with the point-spread function because it corresponds to a very low, very wide skirt around the central blob. The size of the central blob is determined by the resolution of the device (or visa versa) and in our case has a half-intensity radius of around .09 mm (in the centre of the field of view). The scatter skirt however extends easily to the edge of the field of view 38 mm away. It is so low that it would go undetected due to dim-cutoff if we looking at point sources. Only when it is integrated over large areas is its effect noticable. It turns out that about 33 % of the incident light is scattered in this way. This causes a dramatic reduction of contrast.

Several causes can be traced for this phenomenon. The lens contributes some small amount of scatter but the major defects occur because of multiple reflections in the face-plate and reflection from the aperture plate at the end of the drift-tube. It is not known whether any electron optic effects come into this as well. The light enters the face plate and is partially absorbed by the photocathode; some light is, however, reflected and may bounce repeatedly inside the face-plate. Some fraction of the light also passes right through the photo-cathode and strikes the shiny nickel (!) aperture plate only to be reflected onto the back of the photo-cathode.

These problems could be ameliorated by optically coating the face-plate to avoid multiple reflections or to use a fiber-optic front-plate. The aperture plate clearly ought to be made of some more reasonable material (to avoid the magnetic problems) and should be fairly non-reflective.

(We might expect, by the way, that the front-plate scatter is worse for larger iris diameter (lower f-stops) because the light will be entering the face-plate from larger angles relative to the optical axis)

As pointed out this phenomena only occurs when we are integrating signals over large areas. To measure the effect, then, we have to illuminate large areas. One method involves the use of a series of white discs on a black background to be viewed by the image dissector. For each size disc one records the intensity at the centre. This method suffers from the fact that it is hard to find paper surfaces that have a high reflectivity ($> 50\%$) and others having a low reflectivity ($< 10\%$). The observed effect is then considerably lower than expected; in addition, the scatter in the lens is included.

A better method is that of removing the lens, using a point source of light (such as a distant lamp reflected in a metal sphere) and using the iris to allow variable diameter circles of light to fall on the photocathode. We observe in this way the integral of this scatter function. Let the point-spread function be rotationally symmetric, $f(r)$.

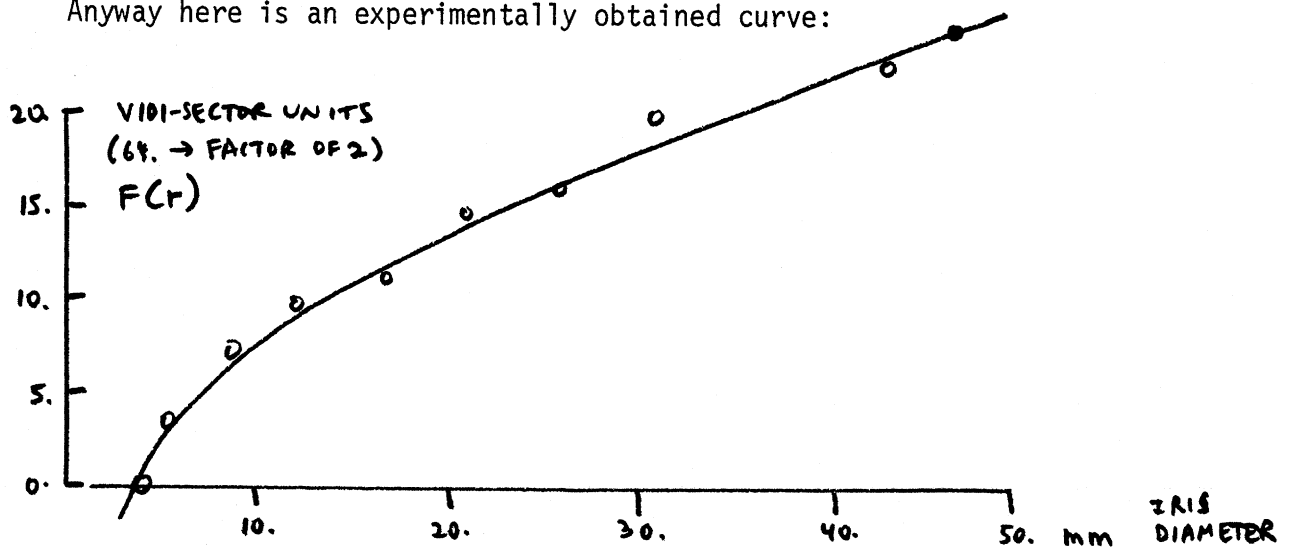
$$F(r) = 2\pi \int_0^r f(t) t dt$$

If we wish we can differentiate the observed function and get:

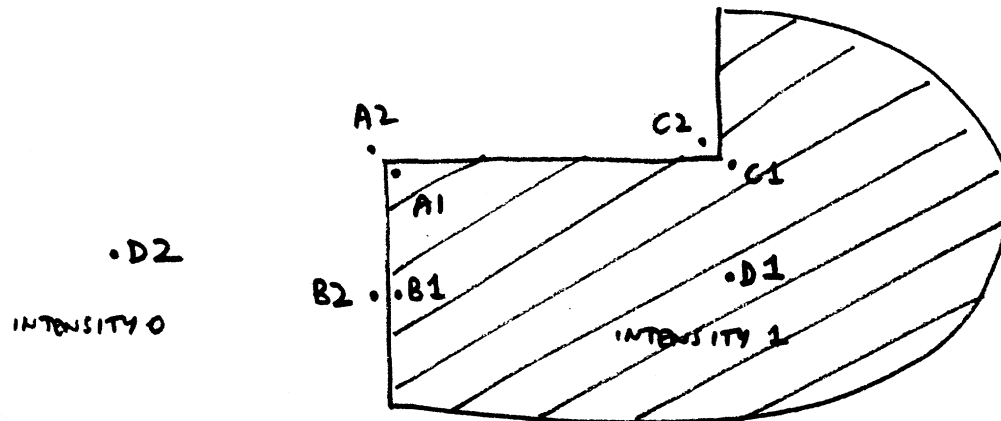
$$2\pi f(r) r$$

There is some reason to suppose that $f(r)$ can be approximated by $e^{-\sigma r}/r$

Anyway here is an experimentally obtained curve:



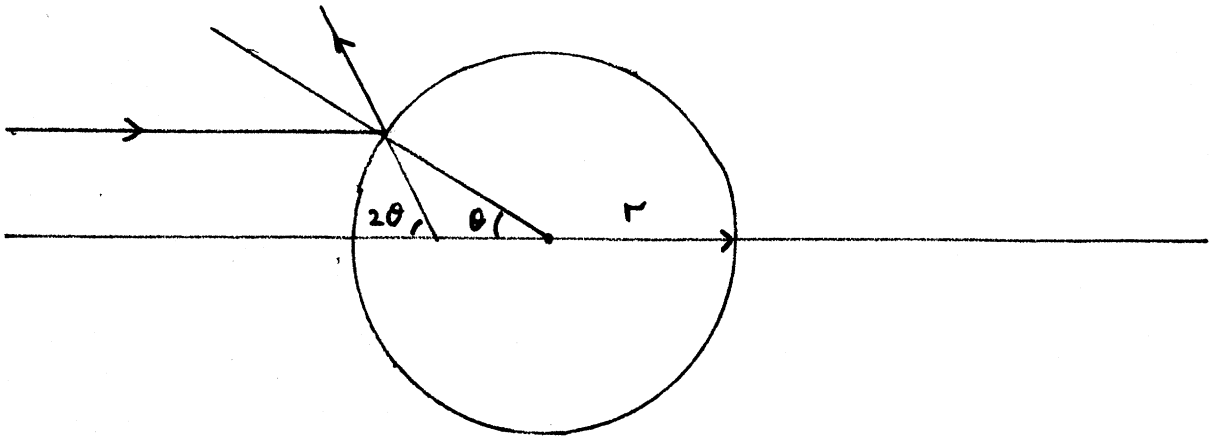
We can use our results to estimate the intensity at various points in an image consisting of large polygonal areas of uniform intensity.



Let's look at the intensity at the points A1, A2, B1, B2, C1, C2 assuming 33 % spill-over:

A1 (90° out of 360° illuminated)	$1. - .33 (3/4) = .75$
A2 (" " " " ")	$0. + .33 (1/4) = .08$
B1 (180° out of 360° illuminated)	$1. - .33 (1/2) = .84$
B2 (" " " " ")	$0. + .33 (1/2) = .18$
C1 (270° out of 360° illuminated)	$1. - .33 (1/4) = .92$
C2 (" " " " ")	$0. + .33 (3/4) = .25$
D1	1.
D2	0.

WHY THE VIRTUAL IMAGE OF A POINT-SOURCE LOOKS EQUALLY BRIGHT FROM ALL DIRECTIONS:



Consider the small surface ring where the light is incident at an angle θ w.r.t to the surface normal.

The incident area is: $2\pi r^2 \sin \theta \cos \theta d\theta = \pi r^2 \sin 2\theta d\theta$

Light falling into this ring is reflected at an angle 2θ w.r.t to the incident ray and with a spread $2 d\theta$. At the distance R , the light reflected from the ring is spread into an area $2 R^2 \sin 2\theta 2 d\theta$. The intensity per unit area at distance R is:

$$I (\pi r^2 \sin 2\theta d\theta) / (2\pi R^2 \sin 2\theta 2 d\theta) = I (r/R)^2 / 4$$

So its independent of what angle one is looking at it from.

This has implications for reflectivity models of surfaces made of spherical particles. It is also useful in producing point-sources with very small source areas (we are assuming both source and observer distant from the sphere). Note that the factor of 4 comes from the fact that the incident is πr^2 , while the light is reflected into and area $4\pi R^2$.

GLOB-TRACKING:

Suppose we have an intensity glob such as a ping-pong ball against a dark background. The object is to track it using the random access camera. Define a two-dimensional pattern of points. The spread and position of this pattern will be servoed using the intensities read.

At each step input the intensities, find their maximum and minimum, IMAX and IMIN. If IMAX is too small, go into search mode, otherwise calculate the following sums

$$\sum X_i I_i \quad \sum Y_i I_i \quad \sum I_i$$

Then adjust the position:

$$\bar{X}_{n+1} = \bar{X}_n + \theta_1 \left(\frac{\sum X_i I_i}{\sum I_i} - \bar{X}_n \right)$$

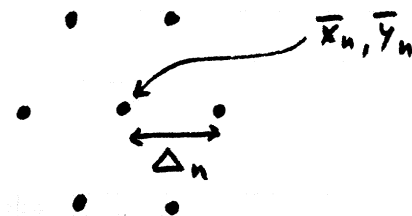
$$\bar{Y}_{n+1} = \bar{Y}_n + \theta_1 \left(\frac{\sum Y_i I_i}{\sum I_i} - \bar{Y}_n \right)$$

Then adjust the size of the pattern:

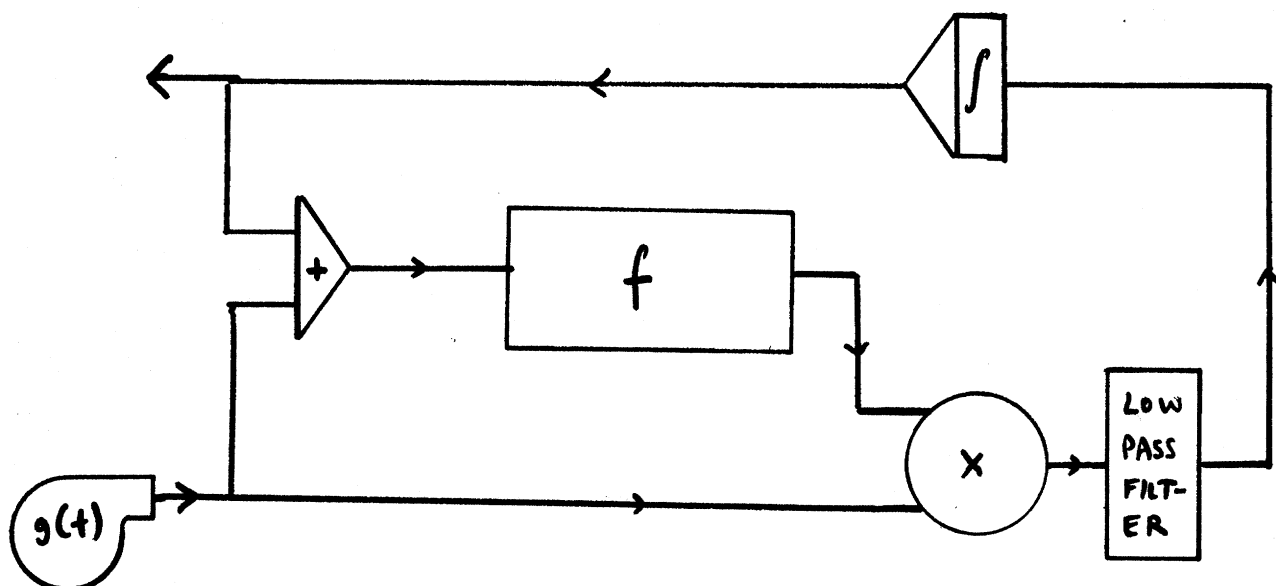
$$\Delta_{n+1} = \Delta_n \left(1 + \theta_2 \left(\frac{IMAX' - IMIN'}{IMAX - IMIN} - 1 \right) \right)$$

Where $(IMAX' - IMIN')$ is the desired state of intensity range.

Usually $\theta_1 > \theta_2$ eg $\theta_1 = 1/2$ $\theta_2 = 1/8$



A RELATED ANALOG TYPE GLOB-TRACKER



Here $g(t)$ is some test function like $\cos(\omega t)$, for example, and f is the external function such as intensity. An interesting case is obtained if we combine two of these circuits, one for x and one for y coordinates in an image dissector camera. We then have a star-tracker. The two $g(t)$'s will need to be "orthogonal" then, $\cos(\omega t)$ and $\sin(\omega t)$, for example.

A similar circuit or equivalent program can be used for light-pen tracking. Interesting variations concern the question of whether the low pass filter can be eliminated or replaced by some other device and whether $g(t)$ can be removed or "self-generated". In other words one aims at a system that is self-contained and samples the image in a way dependent on what is in the image rather than some fixed predetermined pattern.

THE SURVEYOR'S MARK AND FRIENDS:

To track an object using the image dissector camera it is desirable to have to read the intensity at as few steps as possible at each time interval. The pattern to look at must also be designed for three conflicting requirements: ease of acquisition, ease of tracking in fast motion and accuracy of locating when stationary. The first two cause the object to be fairly large, the last requires that some point on it be well defined. The program should have no difficulty in processing the intensities read and should be fairly independent of distance and orientation of the pattern. A radially symmetric pattern with black and white areas seems suitable. In particular, one consisting of a number of intersecting lines with alternate segments filled in black and white seems a winner. The one used by surveyors uses two lines, our robotics calibration programs use three-line patterns.



The image processing is simple. One reads the intensity at a number of points on the circumference of a circle, finds the maximum and minimum and sets up hysteresis thresholds. The lines are detected at the points where the intensity crosses both thresholds in sequence. The six points define three lines. The centre is then estimated to be near the point of minimum sum of squares of perpendicular distances to the three lines. Image motion between successive scans can be almost the radius of the pattern, while its centre can be located extremely accurately by shrinking the sampling circle.

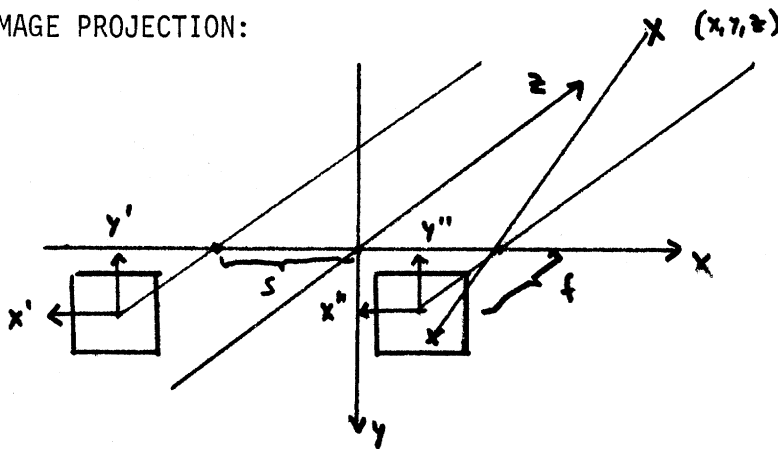
OBJECT ROTATION MATRIX:

Consider an object rotated first about the x-axis (pitch, p), then about the y-axis (yaw, y) and finally about the z-axis (roll, r).

We are interested in the corresponding transformation matrix:

$$\begin{pmatrix} \cos r \cos y & (\cos r \sin y \sin p - \sin r \cos p) & (\cos r \sin y \cos p + \sin r \sin p) \\ \sin r \cos y & (\sin r \sin y \sin p + \cos r \cos p) & (\sin r \sin y \cos p - \cos r \sin p) \\ -\sin y & \cos y \sin p & \cos y \cos p \end{pmatrix}$$

STEREO IMAGE PROJECTION:



$$\text{Left eye: } x' = (x+s)f/z \quad y' = (y)f/z$$

$$\text{Right eye: } x'' = (x-s)f/z \quad y'' = (y)f/z$$

Projection of point (x, y, z)

f is the distance the resulting images are to be viewed from.

$2s$ is the eye separation.

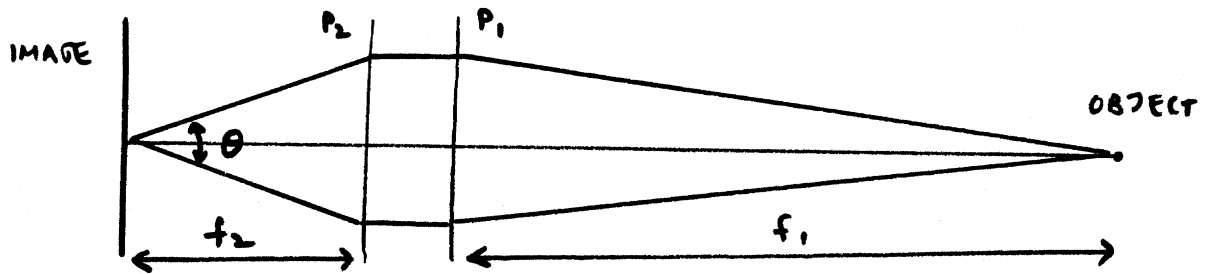
EXPOSURE GUIDE FOR OUR DEC 340 DISPLAY

$$f = \frac{k}{r_1+r_2} \sqrt{\frac{A N P}{F}} (1+I)^{3/2}$$

I	(1+I) ^{3/2}
0	1
1	2.8
2	5
3	8
4	11
5	15
6	18
7	23

- f - f-number indicated on lens.
- k - empirically found to be about 1/125 (gives rise to density of about 2 in negative; i.e. almost overexposed).
- r₁ - Half-intensity radius of spot on DEC 340, varies somewhat with I. use .5 mm unless you have good reason to suspect other value.
- r₂ - Half-intensity radius of blur in camera projected back onto display surface - varies with lens and film used. use .5 mm unless you have good reason to suspect other value.
- r₃ - Spacing of points in image you are displaying. Use ∞ if all the points can be resolved in the image. Use .25 mm * 2^s for vectors, increments and characters of scale s.
- s - scale send to DEC 340, 0-3.
- A -- ASA rating of film.
 For polaroid B/W: 3000
 For 35mm TRI-X : 300
 For 16mm TRI-X : 200
- N - Argument to .NDIS ; i.e. number of times points are displayed.
- P - Packing factor.
 1 for resolved points.
 $\max(1, \frac{r_1+r_2}{r_3})$ for one-dimensional sets of points (vectors, increments, characters)
 $\max(1, (\frac{r_1+r_2}{r_3})^2)$ for two-dimensional sets of points (rasters).
- F - Filter factor. 1 for no filter, 2 for Wratten 15 (afterglow only), 8 for Wratten 47 (flash only).
- I - Intensity parameter send to scope. If varying intensities are to be recorded, use I=5 - highlights will be slightly overexposed but the dark-areas will not be completely under-exposed. 0-7.

SOME LENS FORMULAE:



Let P_1 be the front principal plane, be P_2 the rear principal plane. Let f be the focal length and the media on the two sides of the lens be the same. Let f_1 be the object-lens distance and f_2 the lens-image distance.

The de-magnification of the image is then:

$$M = \frac{f_1}{f_2}$$

We know that: $\frac{1}{f_1} + \frac{1}{f_2} = \frac{1}{f}$ i.e. $(f_1 - f)(f_2 - f) = f^2$

$$f_1 = (1 + M)f$$

$$f_2 = (1 + 1/M)f$$

Let d be the object to image distance (ignoring thick lens effect):

$$d = f_1 + f_2 = f(M + 2 + 1/M) = f \frac{(1+M)^2}{M}$$

Let $x = \left(\frac{d}{2f} - 1 \right)$ then $M^2 - 2xM + 1 = 0$

$$x = \frac{M^2 + 1}{2M} = \frac{1}{2}(M + 1/M)$$

$$M = (x-1) \pm \sqrt{x^2 - 1}$$

$$\frac{dd}{dM} = f \left(1 - \frac{1}{M^2} \right) = f \left(\frac{M^2 - 1}{M^2} \right) \quad \frac{df_1}{dM} = f \quad \frac{df_2}{dM} = -f/M^2$$

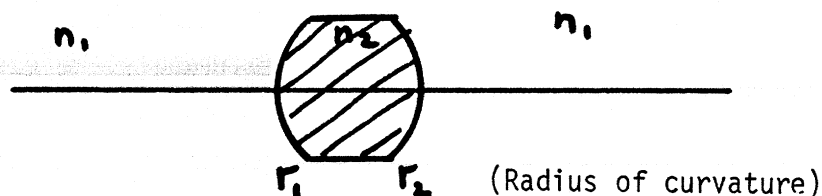
$$\frac{df_1}{dd} = \frac{M^2}{M^2 - 1} \quad \frac{df_2}{dd} = \frac{-1}{M^2 - 1} \quad \frac{df_1}{df_2} = -M^2$$

These formulae are useful for calculating focusing accuracy for example.

Numerical aperture is defined as $n \sin(\theta/2)$, the f-stop as $\frac{1}{2 \sin(\theta/2)}$.

Where θ is the angle subtended by the lens at the centre of the image.

The intensity at the image is proportional to $1/(\text{f-stop})^2$.



Then we have the lens-makers equation:

$$\frac{1}{f_1} + \frac{1}{f_2} = \frac{1}{f} = (n_2 - n_1) \left(\frac{1}{r_1} - \frac{1}{r_2} \right)$$

Optimal pin-hole radius (for diffraction to equal hole spread):

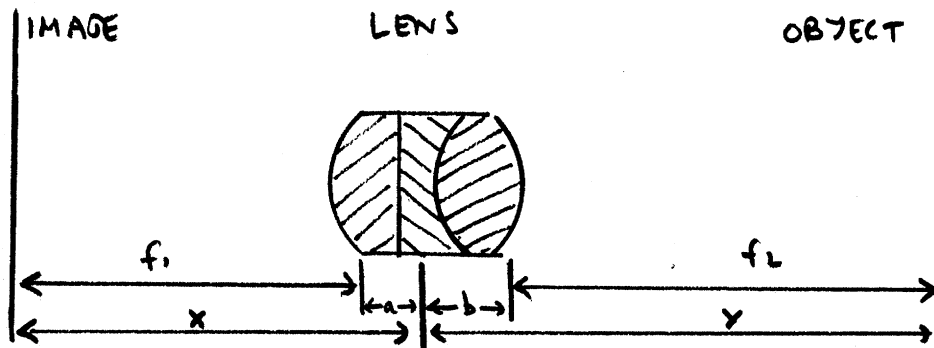
$$r = \sqrt{d \lambda} \quad \text{where } d \text{ is the hole-image distance, } \lambda \text{ the wavelength}$$

Airy radius: $2 (3.8317/(2\pi)) \lambda (\text{f-stop}) = 1.22 \lambda (\text{f-stop})$

(Since 3.8317 is the first zero of $J_1(x)/x$)

FOCAL LENGTH CALIBRATION:

For accurate camera models one needs good measurements of the focal length and the position of the principal planes.



Now $x = f_1 + a$, $y = f_2 + b$ and $(1/f_1) + (1/f_2) = (1/f)$

We measure several combinations of x_i and y_i and attempt to find a , b and most important, f . We can assume that a and b are relatively small relative to f and that f is known approximately. We clearly require three such sets of measurements and could use least-squares methods if we had more. Unfortunately the equations are non-linear. We can make them into polynomials in a , b and f however:

$$1/(x_i - a) + 1/(y_i - b) = 1/f$$

$$((x_i + y_i) - (a + b)) f - (x_i - a)(y_i - b) = 0$$

$$-(ab + bf + fa) + (x_i(f + b) + y_i(f + a)) - x_i y_i = 0$$

We could solve this set of second order polynomials in 3 variables in a number of ways. Perhaps the easiest is multi-dimensional Newton-Raphson iteration. We consider this last expression as a function F of the parameters a , b and f and are aiming for $F(a, b, f) = 0$. For this we require the derivatives:

$$dF/da = y_i - b - f, \quad dF/db = x_i - a - f, \quad dF/df = (x_i + y_i) - (a + b)$$

We can also use guessing or a least squares method. If we can select the x_i and y_i we can also simplify the problem.

Special Case: If we can choose x_i and y_i we might try the following:

$$y_1 = \infty \quad x_1 = f+a \quad a = x_1 - f$$

$$x_2 = \infty \quad y_2 = f+b \quad b = y_2 - f$$

We need one more measurement:

$$1/(x_3 - x_1 + f) + 1/(y_3 - y_2 + f) = 1/f$$

$$\text{Let } x_3 - x_1 = x', \quad y_3 - y_2 = y'$$

$$(y' + f + x' + f) f - (x' + f)(y' + f) = 0$$

$$(x' + y')f + 2f^2 - x'y' - (x' + y')f - f^2 = 0$$

$$-x'y' + f^2 = 0 \quad f = \sqrt{x'y'}$$

$$f = \sqrt{(x_3 - x_1)(y_3 - y_2)}$$

For accuracy we want both differences large. this implies that we want x_3 about the same magnitude as y_3 .

DETERMINING THE TRANSFORM FROM ARM TO EYE SPACE:

Being a rotation and translation we expect:

$$\begin{vmatrix} x_v \\ y_v \\ z_v \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \begin{vmatrix} x_a \\ y_a \\ z_a \end{vmatrix} + \begin{vmatrix} a_{14} \\ a_{24} \\ a_{34} \end{vmatrix}$$

And the matrix ought to be orthogonal (i.e. $A^T A = I$). The coordinates with a-subscripts are arm coordinates, those with a v-subscript are eye coordinates. By allowing the matrix to be non-orthogonal we can absorb some of the distortions and non-linearities. In any case forcing it to be orthogonal introduces a non-linear constraint that messes up the mathematics! We then have to use iterative methods well-known in the art of reducing aerial photographs.

Next we have to consider the projection into the image plane:

$$u = (x_v/z_v)\alpha + u_0$$

$$v = (y_v/z_v)\beta + v_0$$

α and β are normally the same more or less and depend on the focal length and the translation from image coordinates to deflection units. u_0 and v_0 are zero if we choose the image origin on the optical axis which may at times be convenient. It is not hard to show that α, β, u_0 and v_0 can be absorbed into our first transform and we can consider the simpler case:

$$u = x_v/z_v \quad \text{and} \quad v = y_v/z_v$$

This does make the matrix non-orthogonal however. Clearly, multiplying all the a_{ij} 's by any factor causes no change in the image coordinates and we can therefore choose a fixed value for one of them, say $a_{34} = 1$. We then have the problem of determining the values of the other 11 terms.

We need at least 11 equations then and preferably more so as to allow a least-squares solution. One method of determining the transformation matrix depends on moving the arm into n known positions and recording the corresponding x_{ai} , y_{ai} , z_{ai} and image coordinates u_i and v_i . It is convenient to track a special object held in the hand as it moves around rather than to blindly move the hand and try and locate it in the image.

For each such measurement we get 2 equations:

$$x_v - z_v u_i = 0 \quad \text{and} \quad y_v - z_v v_i = 0$$

$$\begin{aligned} & \rightarrow a_{11}x_{ai} + a_{12}y_{ai} + a_{13}z_{ai} + a_{14} & -a_{31}u_i x_{ai} - a_{32}u_i y_{ai} - a_{33}u_i z_{ai} - u_i a_{34} = 0 \\ & a_{21}x_{ai} + a_{22}y_{ai} + a_{23}z_{ai} + a_{24} & -a_{31}v_i x_{ai} - a_{32}v_i y_{ai} - a_{33}v_i z_{ai} - v_i a_{34} = 0 \end{aligned}$$

For n such measurements we get $2n$ such equations which can be separated into two groups and written in matrix form as follows:

$$\begin{bmatrix} x_{a1} & y_{a1} & z_{a1} & 1 & 0 & 0 & 0 & 0 & -u_1 x_{a1} & -u_1 y_{a1} & -u_1 z_{a1} & -u_1 \\ x_{a2} & y_{a2} & z_{a2} & 1 & 0 & 0 & 0 & 0 & -u_2 x_{a2} & -u_2 y_{a2} & -u_2 z_{a2} & -u_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{an} & y_{an} & z_{an} & 1 & 0 & 0 & 0 & 0 & -u_n x_{an} & -u_n y_{an} & -u_n z_{an} & -u_n \\ 0 & 0 & 0 & 0 & x_{a1} & y_{a1} & z_{a1} & 1 & -v_1 x_{a1} & -v_1 y_{a1} & -v_1 z_{a1} & -v_1 \\ 0 & 0 & 0 & 0 & x_{a2} & y_{a2} & z_{a2} & 1 & -v_2 x_{a2} & -v_2 y_{a2} & -v_2 z_{a2} & -v_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & x_{an} & y_{an} & z_{an} & 1 & -v_n x_{an} & -v_n y_{an} & -v_n z_{an} & -v_n \end{bmatrix} * \begin{bmatrix} a_{11} \\ a_{12} \\ \vdots \\ a_{34} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Setting a_{34} to 1 and taking the resulting constant terms $(u_1 \dots u_n, v_1 \dots v_n)$ to the right hand side we obtain $2n$ equations in 11 unknowns. We can make do with $5 \frac{1}{2}$ experimental measurements or attempt a least-squares solution for $n \geq 6$ points. Not more than 3 points should be in any one plane in the first instance to avoid degeneracy. It is convenient to use the points at the tips of an octahedron.

- Notes:
1. A slightly different formulation leads to 18 equations in 18 unknowns; the same results are obtained. (This corresponds to the homogeneous representation.)
 2. If we had assumed orthogonality we would have introduced 3 more constraints and needed only 8 equations, that is 4 experimental points which could conveniently be the corners of a tetrahedron.

RELATION BETWEEN THE SIMPLIFIED AND THE REAL IMAGE COORDINATES:

$$\text{Given: } \begin{vmatrix} x_v \\ y_v \\ z_v \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} \begin{vmatrix} x_a \\ y_a \\ z_a \end{vmatrix} + \begin{vmatrix} a_{14} \\ a_{24} \\ a_{34} \end{vmatrix}$$

$$\text{and } u = x_v/z_v \quad \text{and} \quad v = y_v/z_v$$

Where x_a, y_a, z_a are coordinates relative to the arm coordinate system.
 x_v, y_v, z_v are coordinates relative to the eye and u and v are image coordinates in the simplified system.

Now we introduce the real image coordinates:

$$u' = u \alpha + u_0 \quad \text{and} \quad v' = v \beta + v_0$$

$$\alpha x_v - (u' - u_0)z_v = 0 \quad \text{and} \quad \beta y_v - (v' - v_0)z_v = 0$$

$$(\alpha a_{11} + u_0 a_{31})x_a + (\alpha a_{12} + u_0 a_{32})y_a + (\alpha a_{13} + u_0 a_{33})z_a + (\alpha a_{14} + u_0 a_{34}) - u'(a_{31}x_a + a_{32}y_a + a_{33}z_a + a_{34}) = 0$$

$$(\beta a_{21} + v_0 a_{31})x_a + (\beta a_{22} + v_0 a_{32})y_a + (\beta a_{23} + v_0 a_{33})z_a + (\beta a_{24} + v_0 a_{34}) - v'(a_{31}x_a + a_{32}y_a + a_{33}z_a + a_{34}) = 0$$

So finally:

$$\begin{vmatrix} x'_v \\ y'_v \\ z'_v \end{vmatrix} = \begin{vmatrix} (\alpha a_{11} + u_0 a_{31}) & (\alpha a_{12} + u_0 a_{32}) & (\alpha a_{13} + u_0 a_{33}) \\ (\beta a_{21} + v_0 a_{31}) & (\beta a_{22} + v_0 a_{32}) & (\beta a_{23} + v_0 a_{33}) \\ (a_{31}) & (a_{32}) & (a_{33}) \end{vmatrix} \begin{vmatrix} x_a \\ y_a \\ z_a \end{vmatrix} + \begin{vmatrix} (\alpha a_{14} + u_0 a_{34}) \\ (\beta a_{24} + v_0 a_{34}) \\ (a_{34}) \end{vmatrix}$$

A VERTICAL PREDICATE FOR IMAGE LINES:

In a perspective transformation of the world a set of parallel lines will project into a bundle of lines passing through one point, the vanishing point x_f, y_f .

If we simply assume that vertical means parallel to the arm's z-axis:

Then as $z_a \rightarrow \infty$, $x_v \rightarrow a_{13}z_a$, $y_v \rightarrow a_{23}z_a$, $z_v \rightarrow a_{33}z_a$

And so $x_f = a_{13}/a_{33}$ and $y_f = a_{23}/a_{33}$

In practice vertical means perpendicular to the table. Suppose the table equation is given by:

$$p_1x + p_2y + p_3z + p_4 = 0$$

Then let $x_a = \alpha p_1$, $y_a = \alpha p_2$, $z_a = \alpha p_3$ and let $\alpha \rightarrow \infty$.

$$x_v \rightarrow \alpha (a_{11}p_1 + a_{12}p_2 + a_{13}p_3)$$

$$y_v \rightarrow \alpha (a_{21}p_1 + a_{22}p_2 + a_{23}p_3)$$

$$z_v \rightarrow \alpha (a_{31}p_1 + a_{32}p_2 + a_{33}p_3)$$

$$x_f = \frac{a_{11}p_1 + a_{12}p_2 + a_{13}p_3}{a_{31}p_1 + a_{32}p_2 + a_{33}p_3}$$

$$y_f = \frac{a_{21}p_1 + a_{22}p_2 + a_{23}p_3}{a_{31}p_1 + a_{32}p_2 + a_{33}p_3}$$

To test if a line is vertical or near vertical we calculate the angle it makes with the line connecting it to the vanishing point:

$$x_{12} = x_1 - x_2, \quad x_{1f} = x_1 - x_f, \quad y_{12} = y_1 - y_2, \quad y_{1f} = y_1 - y_f$$

$$(\sin \theta)^2 = (x_{1f}y_{12} - x_{12}y_{1f})^2 / (x_{1f}^2 + y_{1f}^2)(x_{12}^2 + y_{12}^2)$$

GOING FROM IMAGE COORDINATES TO ARM SPACE COORDINATES:

Clearly we need some extra information to make up for the lack of one dimension. But first let's look at what we have:

$$u = x_v/z_v \quad \text{and} \quad v = y_v/z_v$$

$$x_v - u z_v = 0 \quad \text{and} \quad y_v - v z_v = 0$$

$$(a_{11}-ua_{31})x_a+(a_{12}-ua_{32})y_a+(a_{13}-ua_{33})z_a=(a_{14}-ua_{34})$$

$$(a_{21}-va_{31})x_a+(a_{22}-va_{32})y_a+(a_{23}-va_{33})z_a=(a_{24}-va_{34})$$

We need a third equation in x_a , y_a and z_a to be able to solve. We could, for example, be given any one of these three coordinates. More likely is the case where we have some relation to the table. Let the equation of the table be given by:

$$p_1x + p_2y + p_3z + p_4 = 0$$

If the point is on the table we can simply use this equation.

If the point is in the same plane parallel to the table as some other known point x_1, y_1, z_1 then we use the equation:

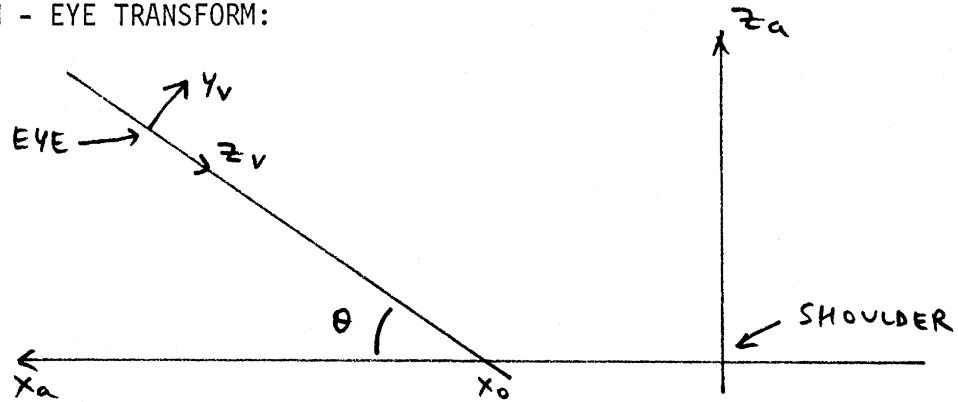
$$p_1x_a + p_2y_a + p_3z_a = p_1x_1 + p_2y_1 + p_3z_1$$

If the point is directly above (along a line normal to the table) some other known point x_2, y_2, z_2 then we have:

$$\begin{aligned} (x_a - x_2) p_3 - (z_a - z_2) p_1 &= 0 \\ (y_a - y_2) p_3 - (z_a - z_2) p_2 &= 0 \end{aligned}$$

We only need one of these and will use the first since it comes out more accurately with the eye-arm geometries we use.

TYPICAL ARM - EYE TRANSFORM:



Suppose we have the above simple geometry. Then:

$$\begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \\ -\cos \theta & 0 & -\sin \theta \end{bmatrix} \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} + \begin{bmatrix} 0 \\ \sin \theta x_0 \\ \cos \theta x_0 + \ell \end{bmatrix}$$

Now we change to real image coordinates:

$$\begin{bmatrix} x'_v \\ y'_v \\ z'_v \end{bmatrix} = \begin{bmatrix} (\alpha a_{11} + u_0 a_{31}) & (\alpha a_{12} + u_0 a_{32}) & (\alpha a_{13} + u_0 a_{33}) \\ (\beta a_{12} + v_0 a_{31}) & (\beta a_{22} + v_0 a_{32}) & (\beta a_{23} + v_0 a_{33}) \\ (a_{31}) & (a_{32}) & (a_{33}) \end{bmatrix} \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} + \begin{bmatrix} (\alpha a_{14} + u_0 a_{34}) \\ (\beta a_{24} + v_0 a_{34}) \\ (a_{34}) \end{bmatrix}$$

So we get:

$$\begin{bmatrix} x'_v \\ y'_v \\ z'_v \end{bmatrix} = \begin{bmatrix} -u_0 \cos \theta & \alpha & -u_0 \sin \theta \\ -\beta \sin \theta - v_0 \cos \theta & 0 & \beta \cos \theta - v_0 \sin \theta \\ -\cos \theta & 0 & -\sin \theta \end{bmatrix} \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} + \begin{bmatrix} u_0 (\cos \theta x_0 + \ell) \\ \beta \sin \theta x_0 + v_0 (\cos \theta x_0 + \ell) \\ \cos \theta x_0 + \ell \end{bmatrix}$$

Now suppose $\theta = 30^\circ$, $\cos \theta = .86\dots$, $\sin \theta = .5$, $u_0 = v_0 = 512$. (Center of coordinates for the image dissector on a scale of 0 - 1024.)

With a lens of 10" focal length we find that $\alpha = 3150$ units/radian.

With a lens of 6.5 " focal length $\alpha = 2000$ units/radian.

(Assuming about 12.5 units per mm on the photocathode)

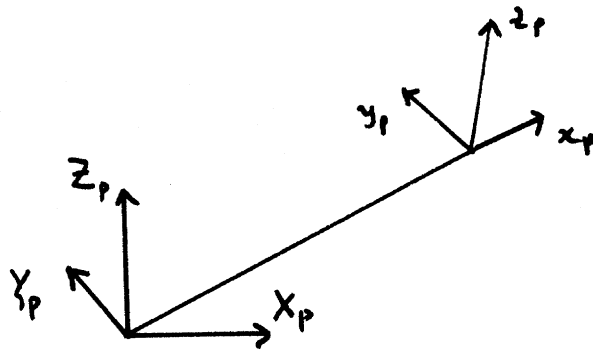
Next suppose $x_0 = 30.0"$, $z = 50.0"$ and use of the 6.5" lens.

$$\begin{array}{ccc|c} -440. & 2000. & -256. & 39800. \\ -1440. & 0. & 1464. & 69800. \\ -.86 & 0. & -.5 & 76. \end{array}$$

Next we normalise by setting $a_{34} = 1$:

$$\begin{array}{ccc|c} -5.8 & 26.3 & -3.38 & 525. \\ -19.0 & 0. & 19.3 & 918. \\ -.0113 & 0. & -.0066 & 1. \end{array}$$

ABSOLUTE ORIENTATION:



$$\begin{bmatrix} X_p \\ Y_p \\ Z_p \end{bmatrix} = R \begin{bmatrix} \alpha x_p \\ \beta y_p \\ \gamma z_p \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

Where R is an orthogonal rotation matrix. α, β, γ are scale factors, often equal to one another. x_0, y_0, z_0 is a displacement vector. There is one of these equations for each point in the object.

Let $X_{ij} = X_i - X_j$ and $x_{ij} = x_i - x_j$, then:

$$\begin{bmatrix} X_{ij} \\ Y_{ij} \\ Z_{ij} \end{bmatrix} = R \begin{bmatrix} \alpha x_{ij} \\ \beta y_{ij} \\ \gamma z_{ij} \end{bmatrix}$$

Multiplying this equation by its transpose we get:

$$\begin{bmatrix} X_{ij} & Y_{ij} & Z_{ij} \end{bmatrix} \begin{bmatrix} X_{ij} \\ Y_{ij} \\ Z_{ij} \end{bmatrix} = \begin{bmatrix} \alpha x_{ij} & \beta y_{ij} & \gamma z_{ij} \end{bmatrix} R^T R \begin{bmatrix} \alpha x_{ij} \\ \beta y_{ij} \\ \gamma z_{ij} \end{bmatrix}$$

Now noting that $R^T R = I$ we get:

$$(x_{ij}^2 + y_{ij}^2 + z_{ij}^2) = (\alpha^2 x_{ij}^2 + \beta^2 y_{ij}^2 + \gamma^2 z_{ij}^2)$$

Now suppose we are given four points in each coordinate system:

$$\begin{vmatrix} (x_{12}^2 + y_{12}^2 + z_{12}^2) \\ (x_{23}^2 + y_{23}^2 + z_{23}^2) \\ (x_{34}^2 + y_{34}^2 + z_{34}^2) \end{vmatrix} = \begin{vmatrix} x_{12}^2 & y_{12}^2 & z_{12}^2 \\ x_{23}^2 & y_{23}^2 & z_{23}^2 \\ x_{34}^2 & y_{34}^2 & z_{34}^2 \end{vmatrix} \begin{vmatrix} \alpha^2 \\ \beta^2 \\ \gamma^2 \end{vmatrix}$$

It is now easy to solve for α, β, γ . Let $x'_{ij} = \alpha x_{ij}$ and so on:

$$\begin{vmatrix} x_{12} \\ x_{23} \\ x_{34} \end{vmatrix} = \begin{vmatrix} x'_{12} & y'_{12} & z'_{12} \\ x'_{23} & y'_{23} & z'_{23} \\ x'_{34} & y'_{34} & z'_{34} \end{vmatrix} \begin{vmatrix} r_{11} \\ r_{12} \\ r_{13} \end{vmatrix}$$

Let X be the vector $[x_{12} \ x_{23} \ x_{34}]^T$ and similarly for Y and Z .

Let x' be the vector $[x'_{12} \ x'_{23} \ x'_{34}]^T$ and similarly for y' and z' .

Combining three equations like the above:

$$(X \ Y \ Z) = (x' \ y' \ z') R^T$$

$$R^T = (x' \ y' \ z')^{-1} (X \ Y \ Z)$$

$$= \begin{vmatrix} x'_{12} & y'_{12} & z'_{12} \\ x'_{23} & y'_{23} & z'_{23} \\ x'_{34} & y'_{34} & z'_{34} \end{vmatrix}^{-1} \begin{vmatrix} x_{12} & y_{12} & z_{12} \\ x_{23} & y_{23} & z_{23} \\ x_{34} & y_{34} & z_{34} \end{vmatrix}$$

Due to measurement inaccuracies R determined this way may not be orthogonal, one can if one wishes adjust it iteratively using Newton-Raphson:

$$R_{n+1} = R_n - .5 (R_n^T R_n - I)$$

or

$$R_{n+1} = .5 ((R_n^T)^{-1} + R_n)$$

Finally we have to find the displacement vector:

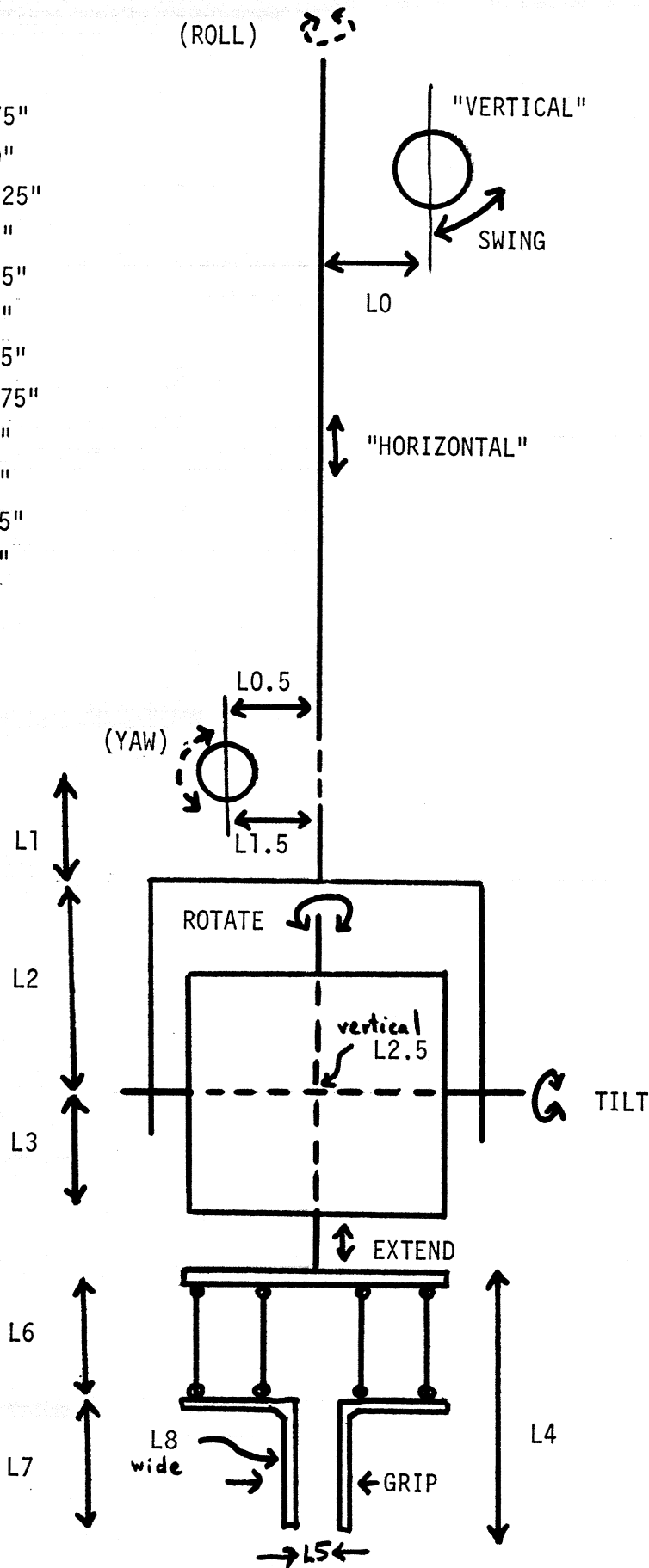
$$\begin{vmatrix} x_0 \\ y_0 \\ z_0 \end{vmatrix} = \begin{vmatrix} x_p \\ y_p \\ z_p \end{vmatrix} - R \begin{vmatrix} \alpha x_p \\ \beta y_p \\ \gamma z_p \end{vmatrix}$$

We can get four estimates from this which we can average if necessary.

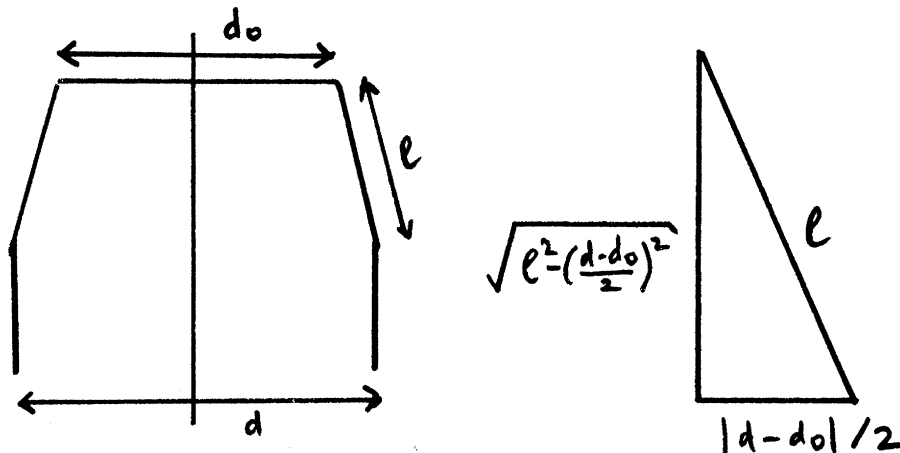
If $\alpha = \beta = \gamma$ we can get away with only three points.

GEOMETRY OF THE AMF-VERSTRAN ARM WITH THE ALLES HAND:

L0	2.75"
L0.5	1.0"
L1	3.625"
L1.5	1.0"
L2	10.5"
L2.5	.75"
L3	4.75"
L4	6.375"
L5	.25"
L6	2.0"
L7	2.75"
L8	.56"



COMPENSATION FOR GRIP MOTION:



The geometry of the grippers is equivalent to the above. We then find a motion along the axis of the grippers w.r.t. the most extended:

$$e \left(1 - \sqrt{1 - \left(\frac{d - d_0}{2e} \right)^2} \right)$$

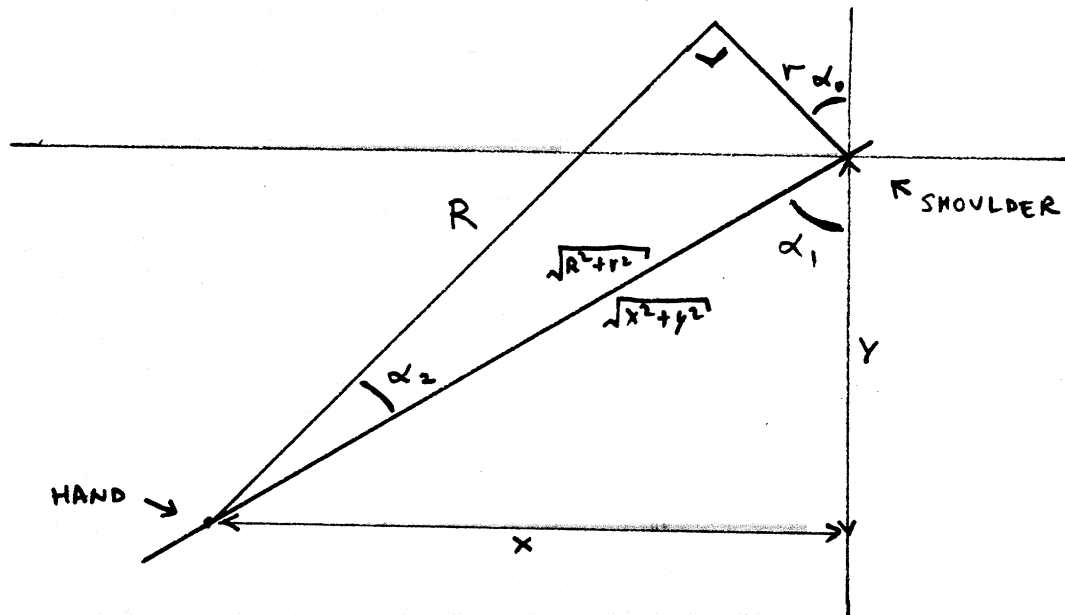
COMPENSATION FOR TILT MOTION:

When the tilt-axis is inclined θ w.r.t vertical one can adjust the horizontal extend by $\sin \theta * \text{hand-extend}$ and the vertical motion by $\cos \theta * \text{hand-extend}$.

On the whole the arm geometry is very simple and allows direct determination of joint angles and extensions given a desired hand position and orientation, keeping in mind that one only has 5 degrees of freedom.

CONVERSION FROM RECTANGULAR COORDINATES TO PSEUDO-POLAR:

The AMF arm has an offset in its otherwise extremely simple geometry:



Given x , y we need to find R and α_0 .

$$R = \sqrt{x^2 + y^2 - r^2} \quad x^2 + y^2 > r^2$$

$$\tan \alpha_1 = x/y \quad \tan \alpha_2 = r/R$$

$$\tan \alpha_0 = 1 / \tan (\alpha_1 - \alpha_2) = \frac{yR + xr}{xR - yr} = \frac{xy + rR}{x^2 - r^2} = \frac{xy + rR}{R^2 - y^2}$$

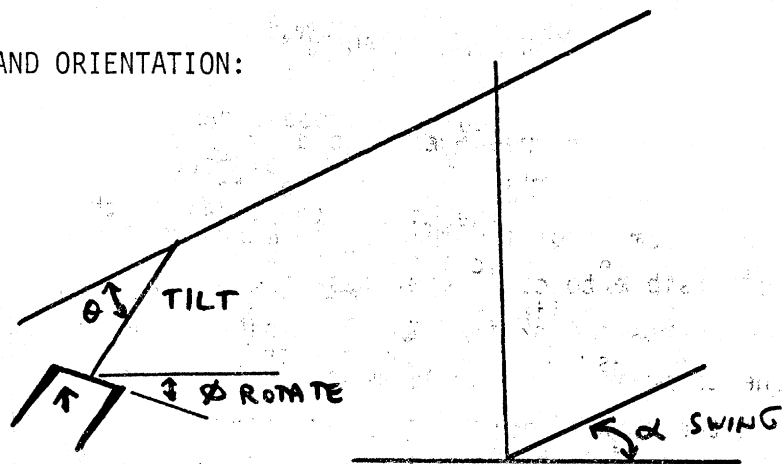
Here we cannot avoid the use of arctan because we actually need the angle. Note that for the AMF arm $r = 2.75$.

We used the fact that $x^2 - r^2 = R^2 - y^2$

And that $\tan(a-b) = (\tan a - \tan b) / (1 + \tan a \tan b)$

R is the "horizontal" extend and α_0 is the "swing".

MAINTAINING A CONSTANT HAND ORIENTATION:



Let the normal unit vector to the plane containing the two fingers be (a, b, c)

$$(a \ b \ c) = \begin{pmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{pmatrix} \begin{vmatrix} 0 \\ 0 \\ 1 \end{vmatrix}$$

$$= \begin{vmatrix} \cos \alpha \sin \theta \cos \phi + \sin \alpha \sin \phi \\ -\sin \alpha \sin \theta \cos \phi + \cos \alpha \sin \phi \\ \cos \theta \cos \phi \end{vmatrix}$$

So given one constraint we can solve for α, θ, ϕ keeping in mind that $a^2 + b^2 + c^2 = 1$

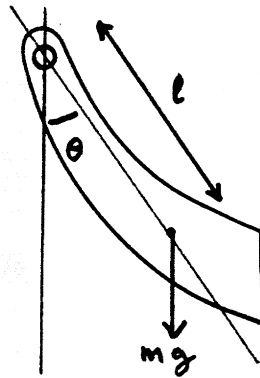
Most commonly we would be given the swing, α , then:

$$\sin \phi = (a \sin \alpha + b \cos \alpha)$$

$$\tan \theta = (a \cos \alpha - b \sin \alpha) / c$$

MEASURING THE INERTIA OF A LINK IN AN ARM:

For fast arm motions one requires a good dynamic model of the arm, including the geometry of joints and motor torques. Also required is the approximate moment of inertia of the links in the arm. It is usually not feasible to calculate these because of the complex shape and number of parts a link is made of. A simple empirical method requires one to measure the total mass and distance of the centre of gravity from the connection to the preceding link as well as the period of oscillation when the link is suspended from this connection.



Let m = mass of link, l = distance of c.g. from preceding connection
 g = acceleration due to gravity (9.8 meter/second²)
 T = period of oscillation, I = moment of inertia

$$I \ddot{\theta} = -mgl \sin \theta \quad \ddot{\theta} = -(mgl/I) \theta$$

$$\theta = A \cos(\sqrt{mgl/I} t) = A \cos(2\pi t/T)$$

$$T = 2\pi \sqrt{I/(mgl)}$$

$$I = mgl (T/2\pi)^2$$

Example: Pendulum made of string and heavy weight: $T = 2\pi \sqrt{l/g}$

$$I = mgl (l/g) = ml^2$$

