SUMMARY OF SELECTED VISION TOPICS

Working Paper 30

Patrick H. Winston

Massachusetts, Institute of Technology
Artificial Intelligence Laboratory

July 1972

ABSTRACT

This is an introduction to some of the MIT AI 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 working papers ot theses.

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

Summary of Selected Vision Topics

VISION FLASH 30

by

Patrick H. Winston

Massachusetts Institute of Technology

Artificial Intelligence Laboratory

Robotics Section

JULY 1972

Abstract

This is an introduction to some of the MIT AI 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.

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

Vision flashes are informal papers intended for internal use.

This memo is located in TJ6-able form on file VIS;VF30 >.

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 exibited 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 polarizes 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.

Finally a scenerio exhibits the flavor of the system in a situation where a simple structure is copied from spare parts.

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 B 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 interpretions.

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

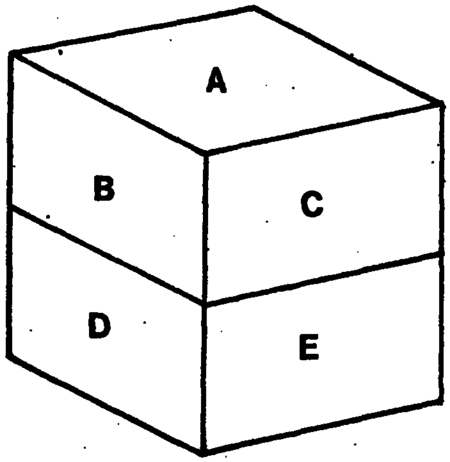


Figure 1
The task of the body finding program is to understand how the regions of the scene form bodies.

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 binding. Whenever a coincidence produces an accidental link, as for example the links

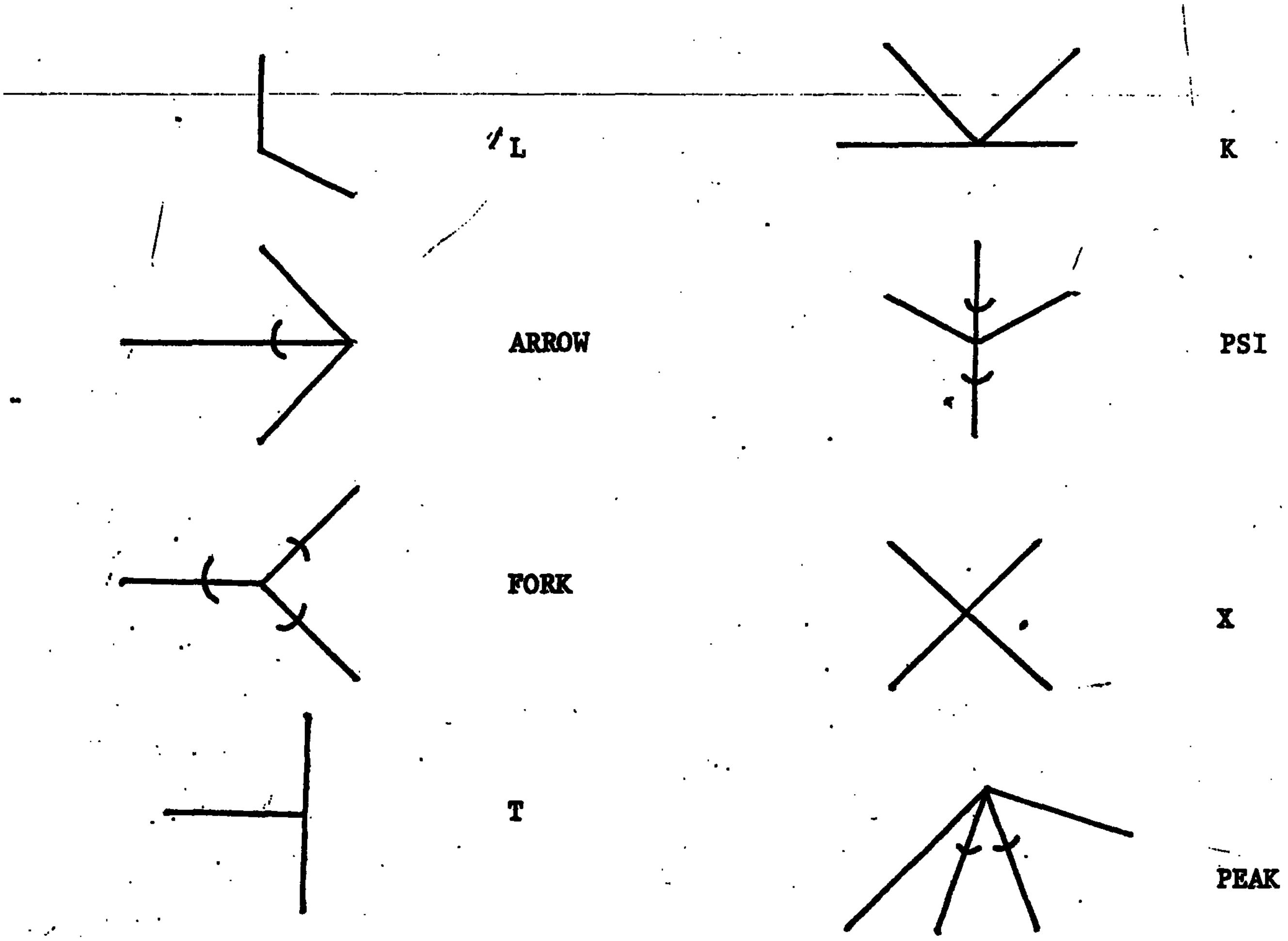


Figure 2
The Guzman links for various vertex types.

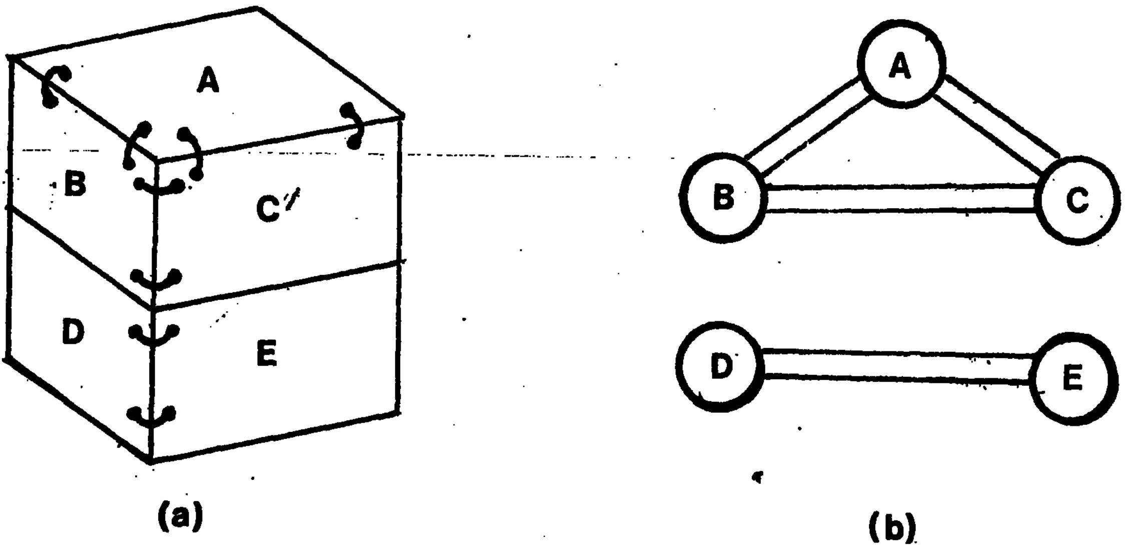


Figure 3
The links formed by the vertices of a simple scene.

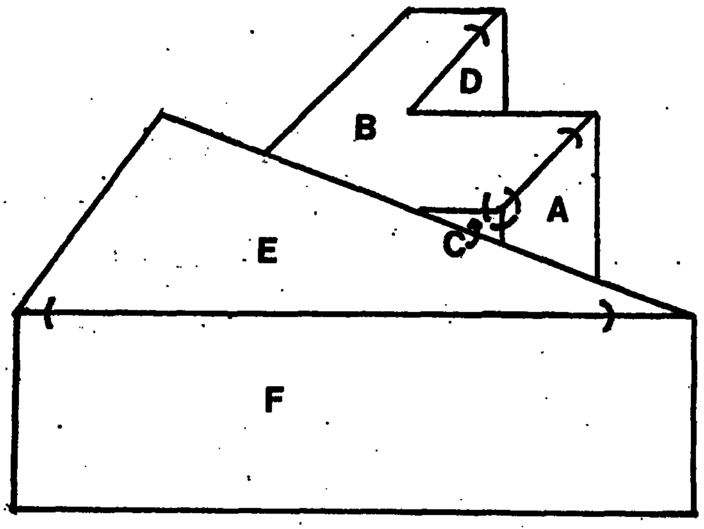


Figure 4
Various linking algorithms cause this to be seen as two, three, or four bodies.

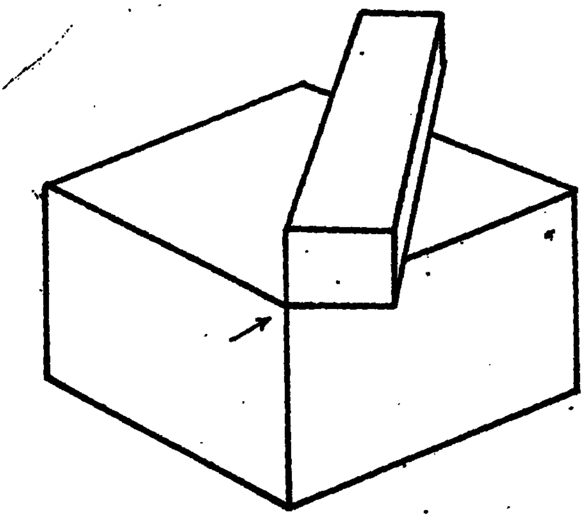


Figure 5
A coincidence causes placement of an incorrect link.

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

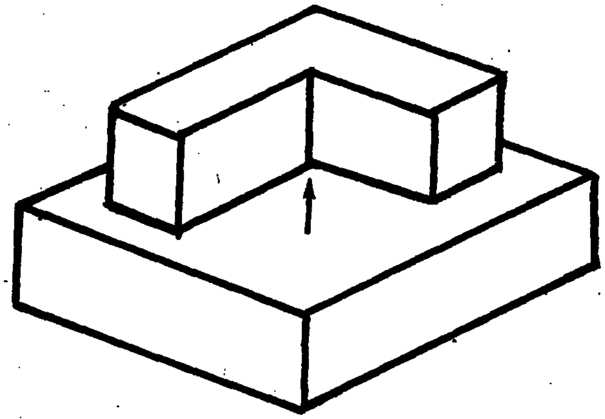


Figure 6
The fork vertex causes the two bodies to be linked together unless the offending links are inhibited by the adjacent arrows.

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?

Experiment 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 this should 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 perjurous 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's 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 the common situation where short shadow boundaries inkvolve 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.

Waltz and Semantic Interpretation

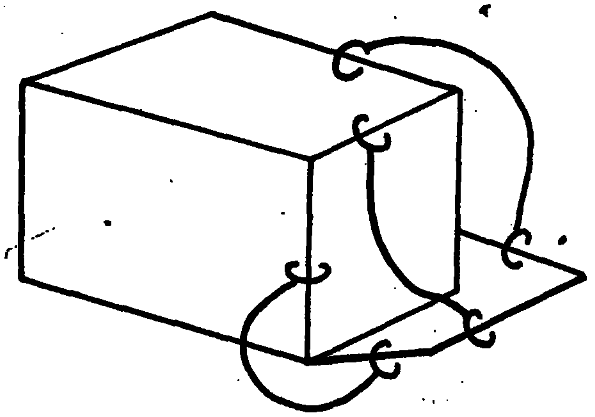


Figure 7
Simple heuristics allow shadow lines to be associated with the edges causing them.

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 arangements of labels around a vertex are physically possible. We will never see a L type vertex with both wings labeled plus no matter how many legal line drawings we examine. (It is presumed

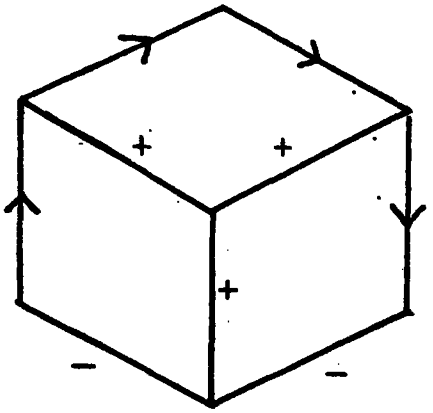


Figure 8
Huffman lables for a cube. Plus implies a convex edge, minus implies concave, and an arrow implies only one of the edge-forming surfaces is visible.

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 on 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 about the structure.

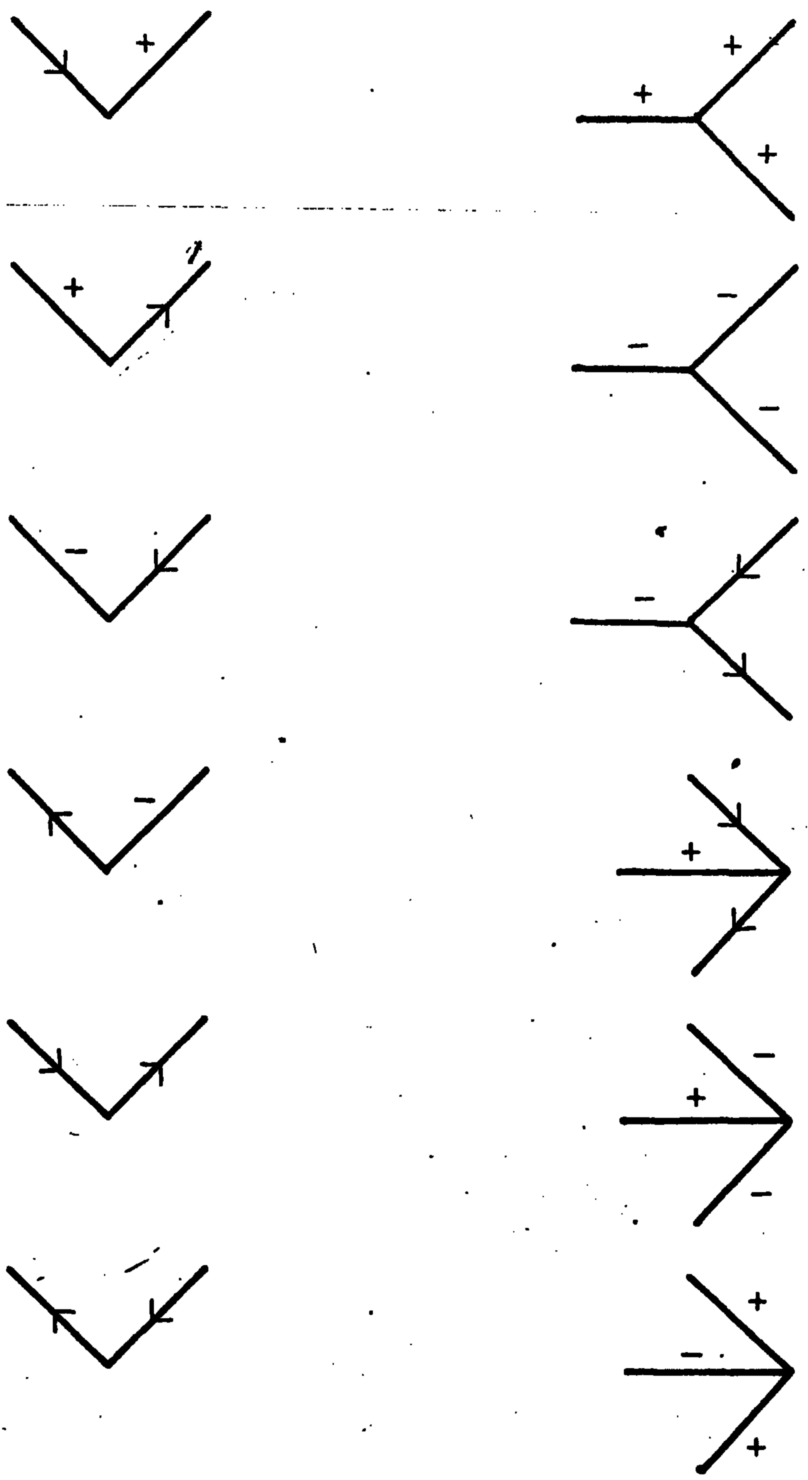


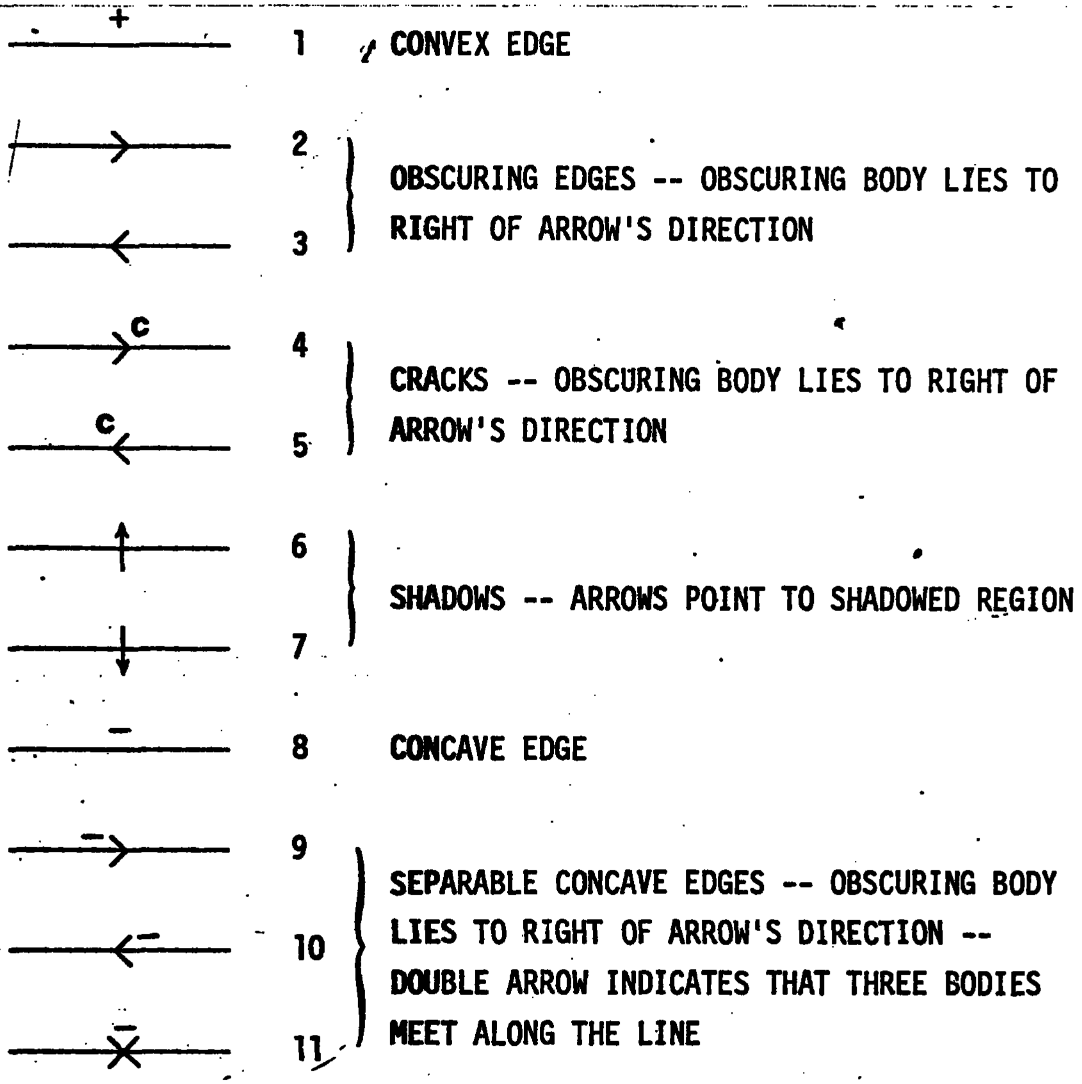
Figure 9
Physically possible configurations of lines around vertices.

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 possibillities suggest that the set of legal labels would include 11 X 3 X 3 = 99 entries, but a few simple facts immediately eliminates 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 11 gives the result. Happily the possible vertex labelings



CONVEX EDGE

OBSCURING EDGES -- OBSCURING BODY LIES TO RIGHT OF ARROW'S DIRECTION

CRACKS -- OBSCURING BODY LIES TO RIGHT OF ARROW'S DIRECTION

SHADOWS -- ARROWS POINT TO SHADOWED REGION

CONCAVE EDGE

SEPARABLE CONCAVE EDGES -- OBSCURING BODY LIES TO RIGHT OF ARROW'S DIRECTION -- DOUBLE ARROW INDICATES THAT THREE BODIES MEET ALONG THE LINE

Figure 10
Line interpretations recognized by Waltz's program.

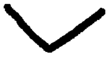
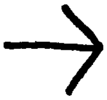
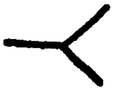
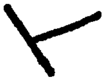
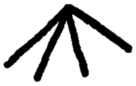
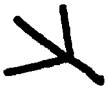
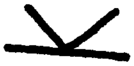
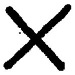
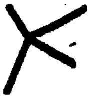
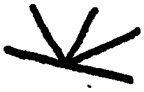
	APPROXIMATE NUMBER OF COMBINATORIALLY POSSIBLE LABELINGS	APPROXIMATE NUMBER OF PHYSICALLY POSSIBLE LABELINGS
	2,500	80
	125,000	70
	125,000	500
	125,000	500
	6×10^6	10
	6×10^6	300
	6×10^6	100
	6×10^6	100
	6×10^6	100
	3×10^8	30

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

constitute only a tiny fraction of the ways labels can be arrayed arround 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 pluzles gives an idea of what is happening: The shape 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 offeres 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'oeurve. Waltz's forthcoming doctoral disertation 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 getting into those exciting results, I can comment on how his

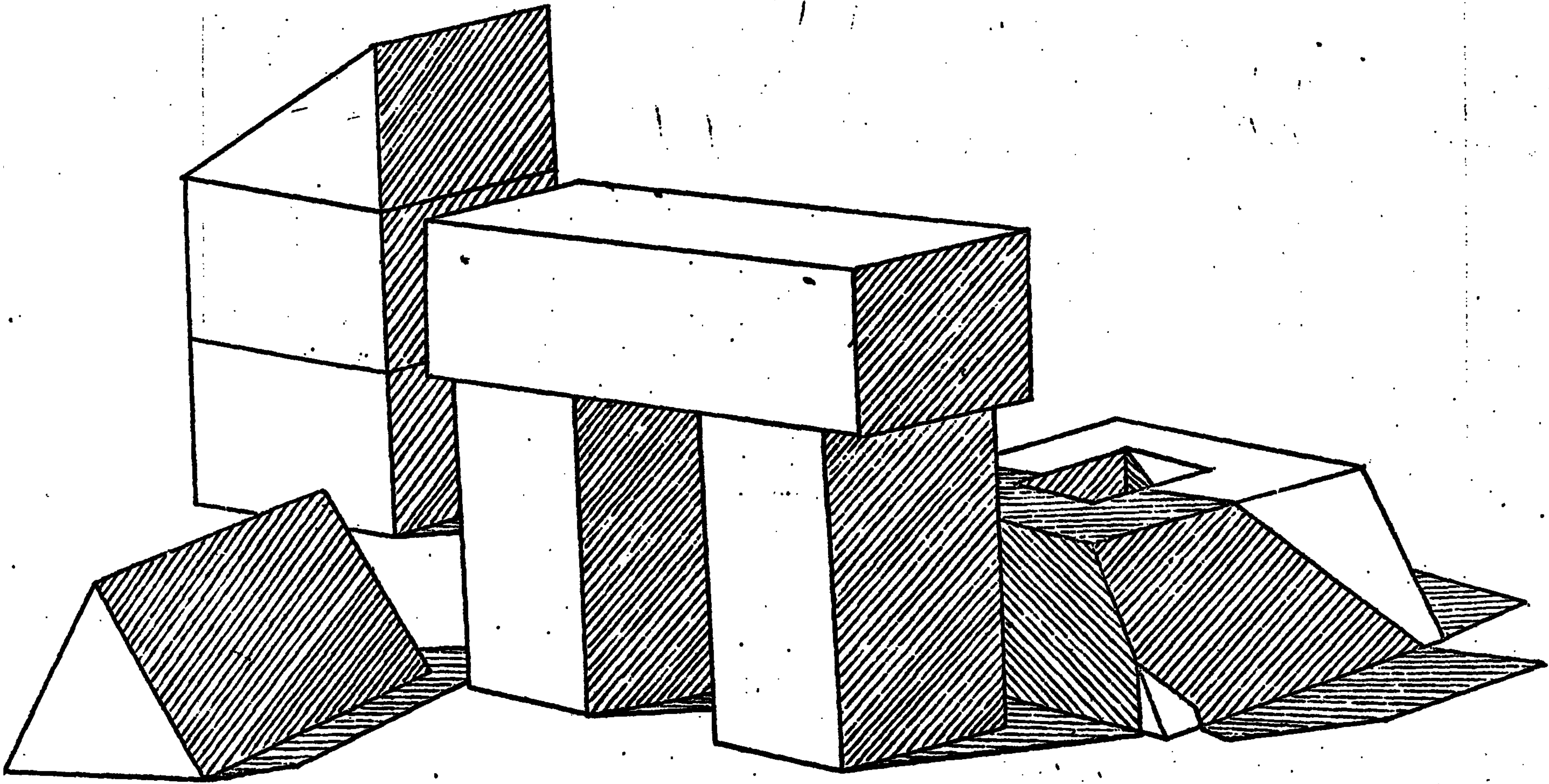


Figure 12
Waltz's program easily handles complicated scenes.

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 dimentional 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, and Huffman and Clowes. Each step built on the ideas and experiments with the previous one, either as 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

Heterarcy

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. We put it together only to have quickly a vehicle 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.

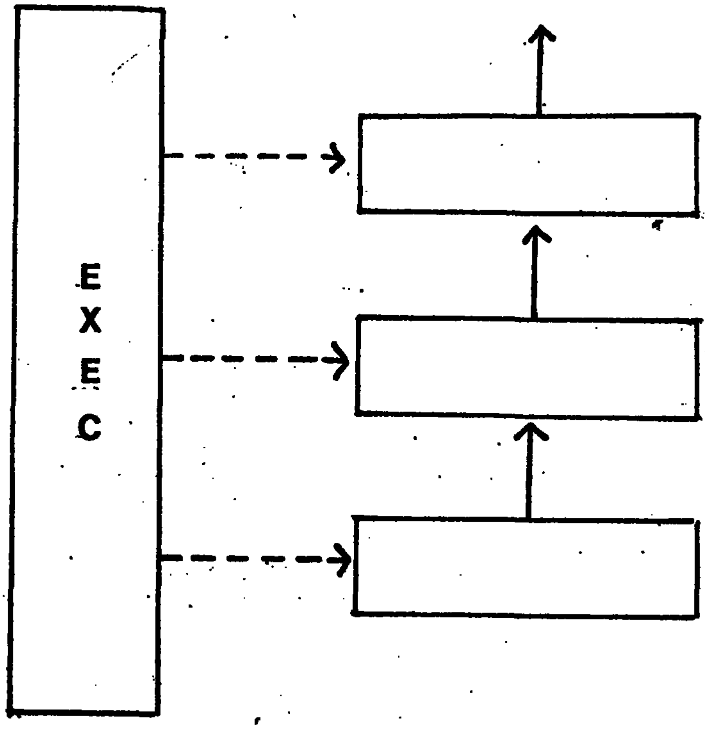


Figure 13
The simple pass oriented system metaphor.

1. A complex system should be goal oriented. 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 the system will be in 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 go.

Graphically such a system looks more like a network of procedures rather than an orderly, immutable sequence. Each procedure is connected to others via potential control transfer links. In practice which of these links are 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. Low 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.

Finin 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 wandering along a beach rift with ant sized obstacles. The ant's curvacious path might seem to be an insanely complex ritual to someone looking 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 whos 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 the dimensions of the brick in question may be determined. Figure 14 shows some of the skeletons found in various situations by this module.

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. Oops! 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?

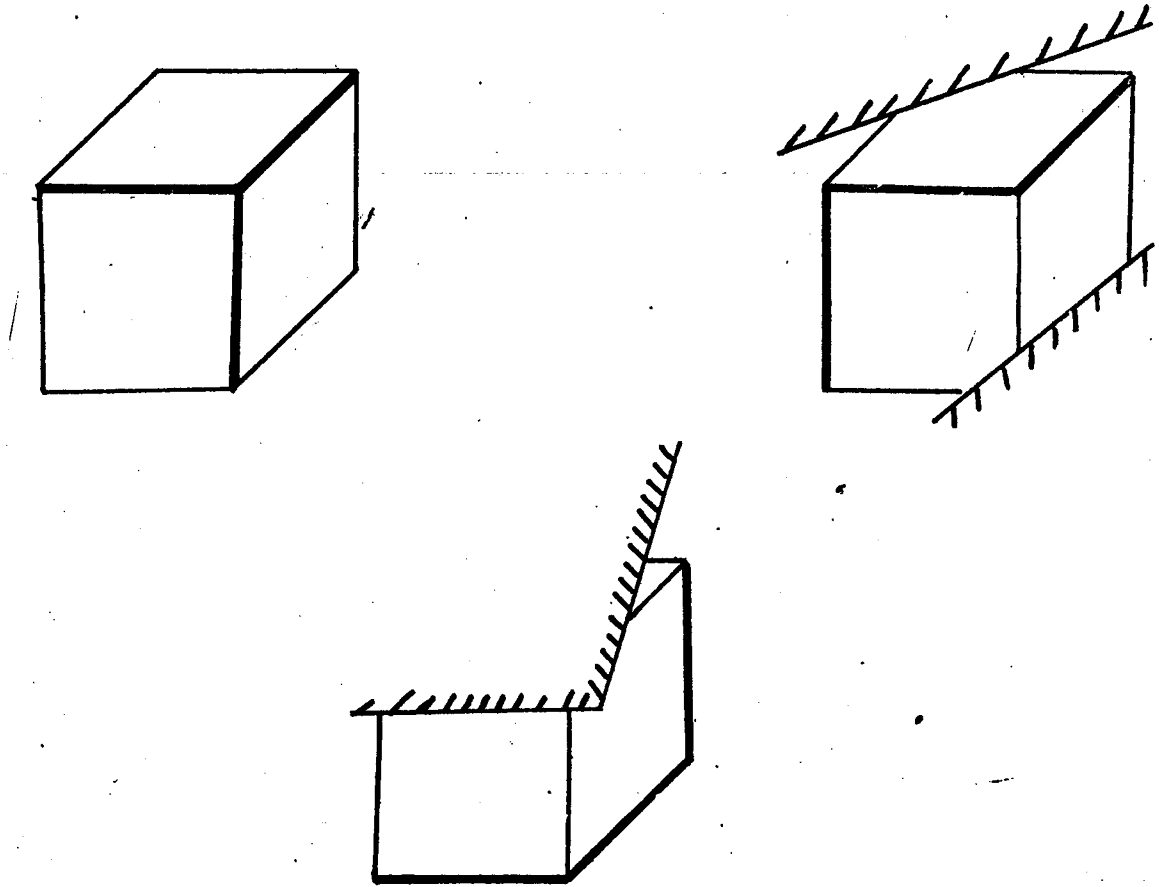


Figure 14
Some skeletons found for bricks.

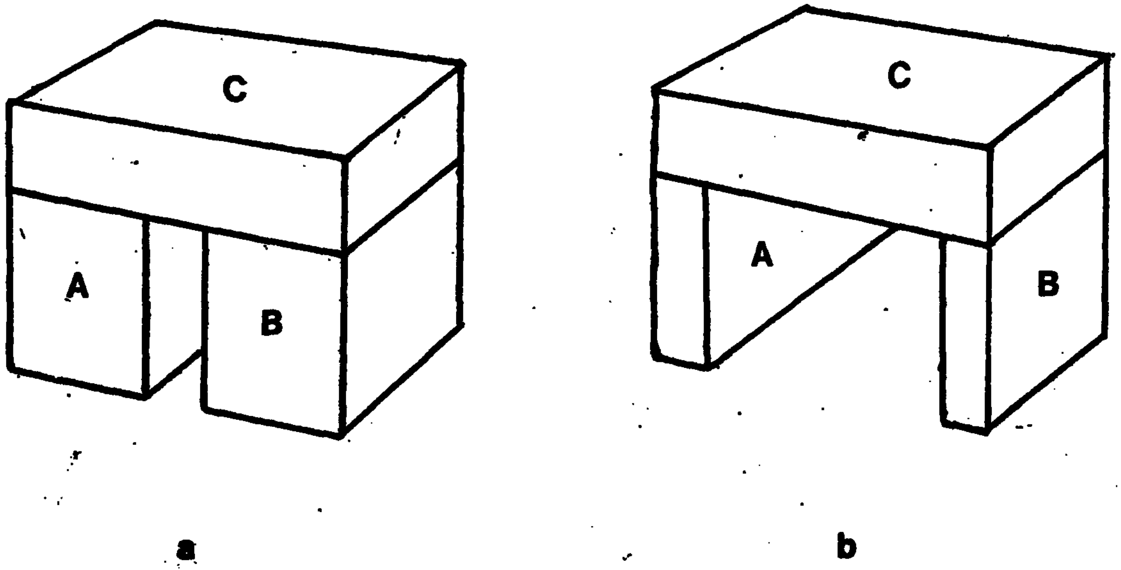


Figure 15
In one case, A's depth is extrapolated from B's. In the other no hypothesis can be confirmed.

Yes, Both are bricks.

Are they both oriented the same way?

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 realtionship 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 hill climbs to a set of actual coordinates for the vertices on the retina (Winston and Lerman 1972). From these retinal coordinates, revised three-dimensional coordinates can be determined, 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

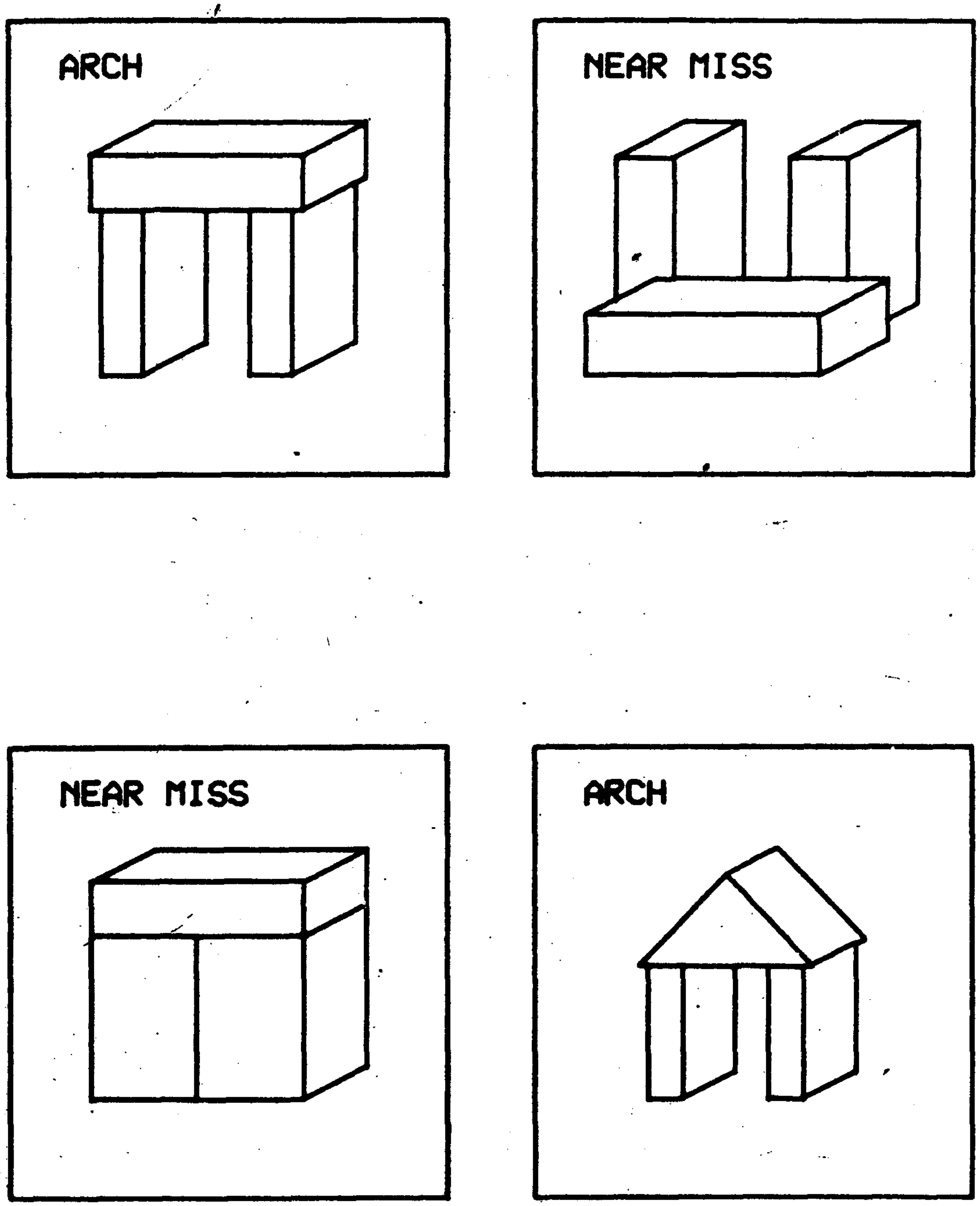


Figure 16
An arch training sequence.

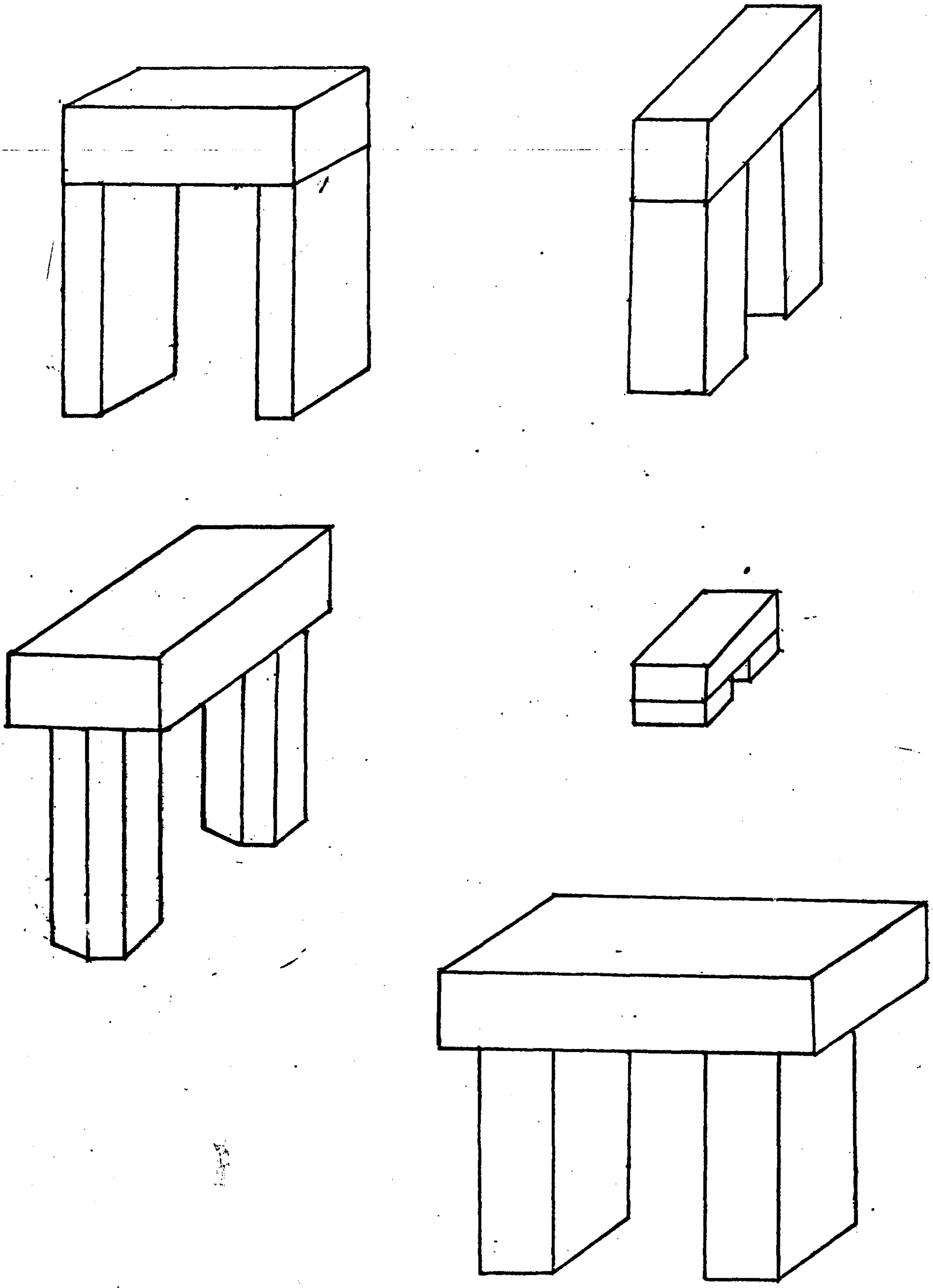


Figure 17
Structures recognized as arches.

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 admissable 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.

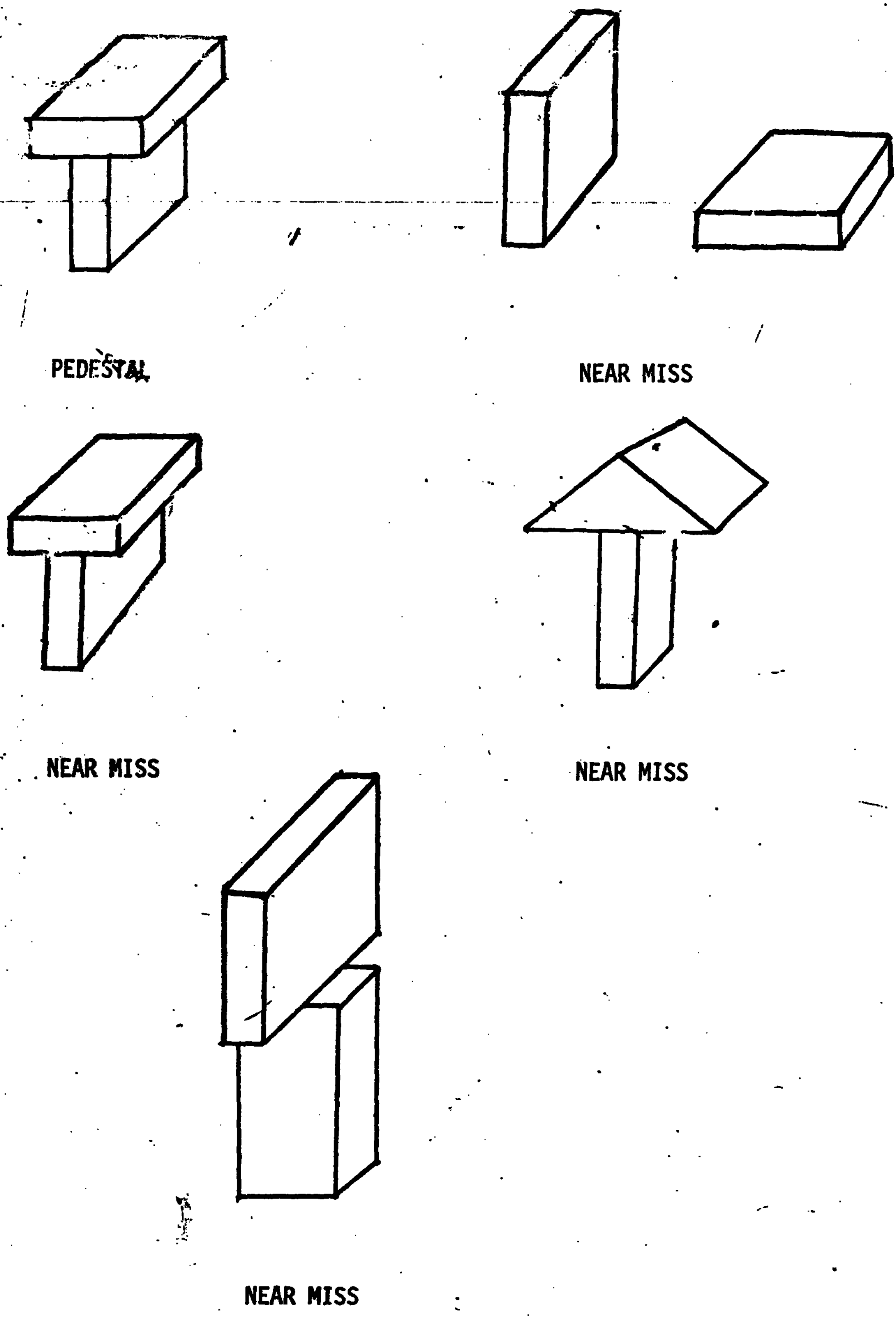
The purpose of this learning process is to create in the machine whatever is needed to identify instances of learned concepts. This leads directly to the notion of a model. To be precise, I use the term as follows:

> 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 heirarchical 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.

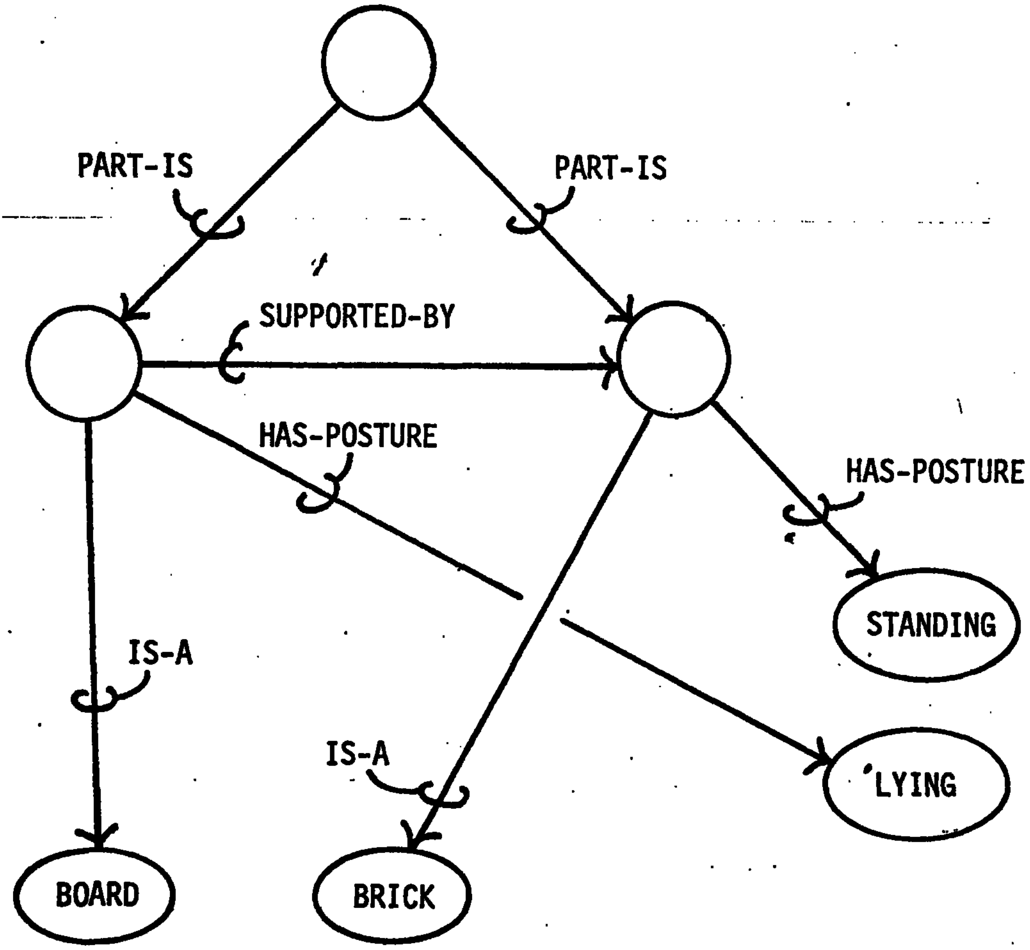


Figure 19
A pedestal description.

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 admissable 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 gore 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. Eut 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

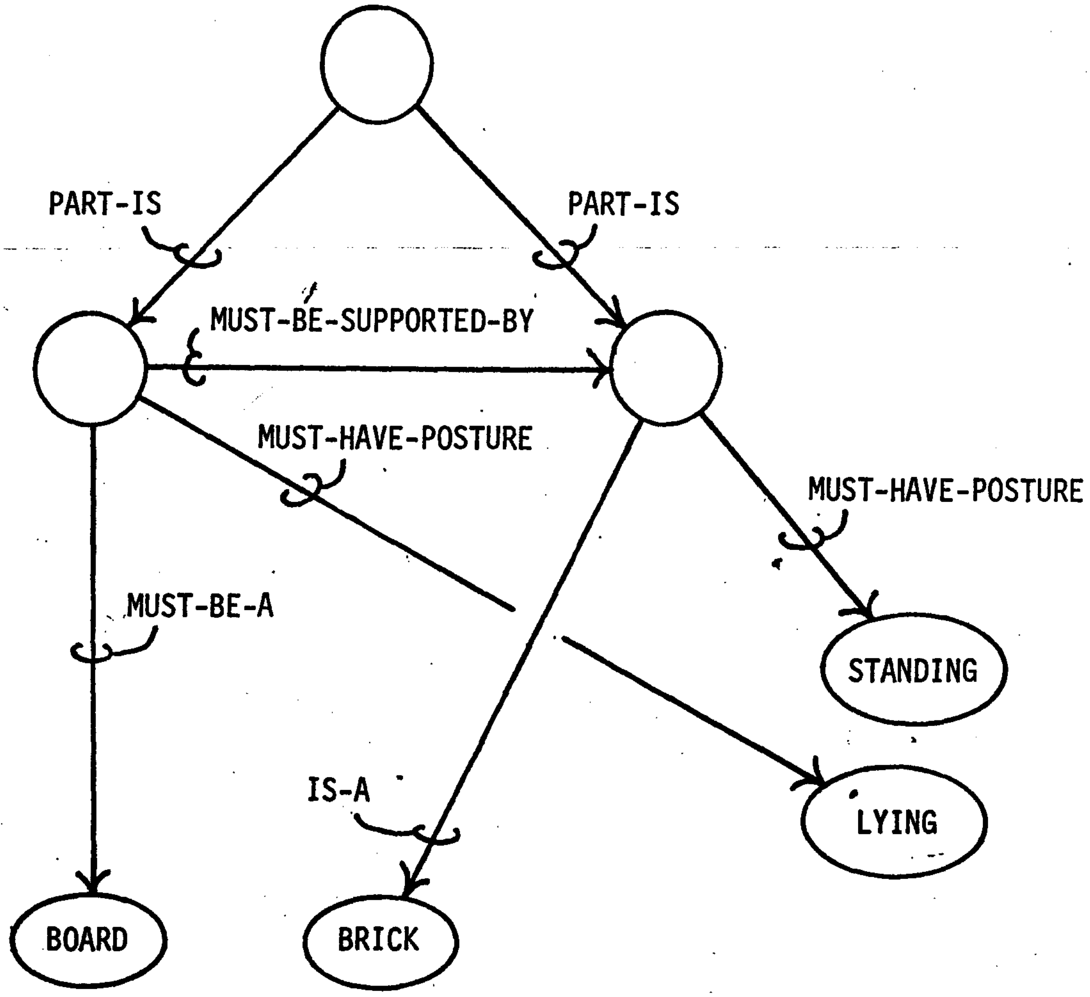


Figure 20
A pedestal model.

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.

Importantly, it is now clear how information of a 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 comparison, 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

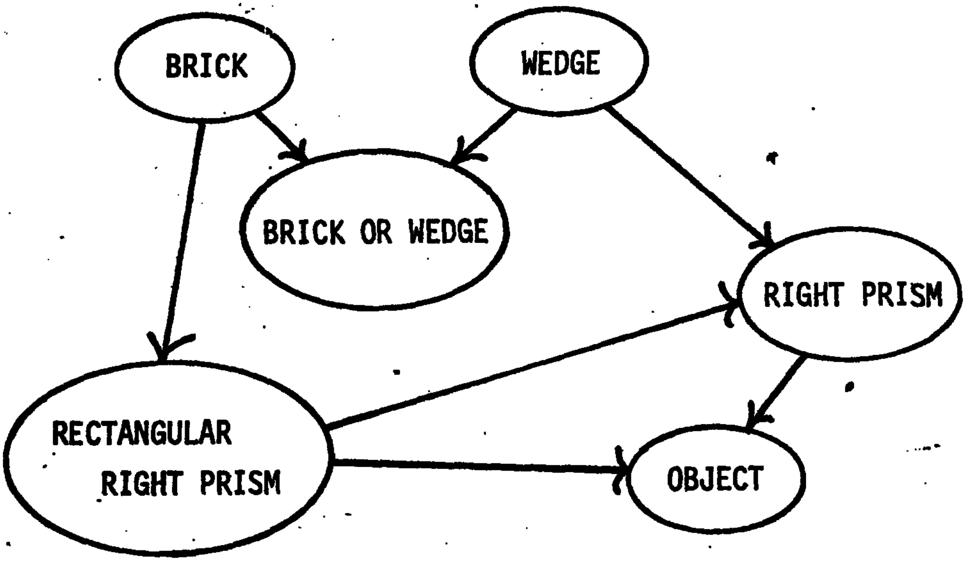


Figure 21
Relations between brick, wedge, and object. All pointers are a-kind-of pointers.

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 to 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

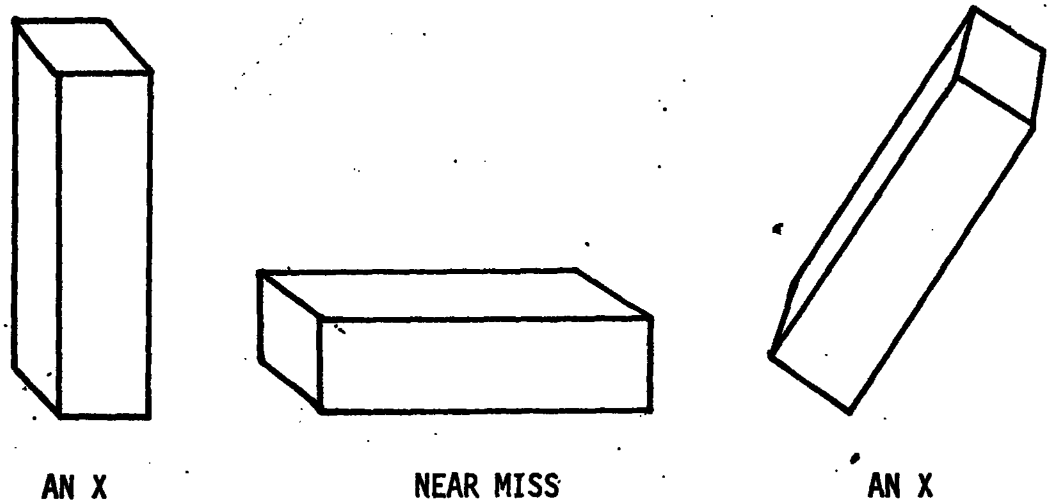


Figure 22
A training sequence that leads to backup.

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 both is tangential to the main thrust and is 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 - hupothesize - criticize idea to find otherwize inaccessible properties for hidden objects.

4) A network of programs that jiggles an object if the arm errs to 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

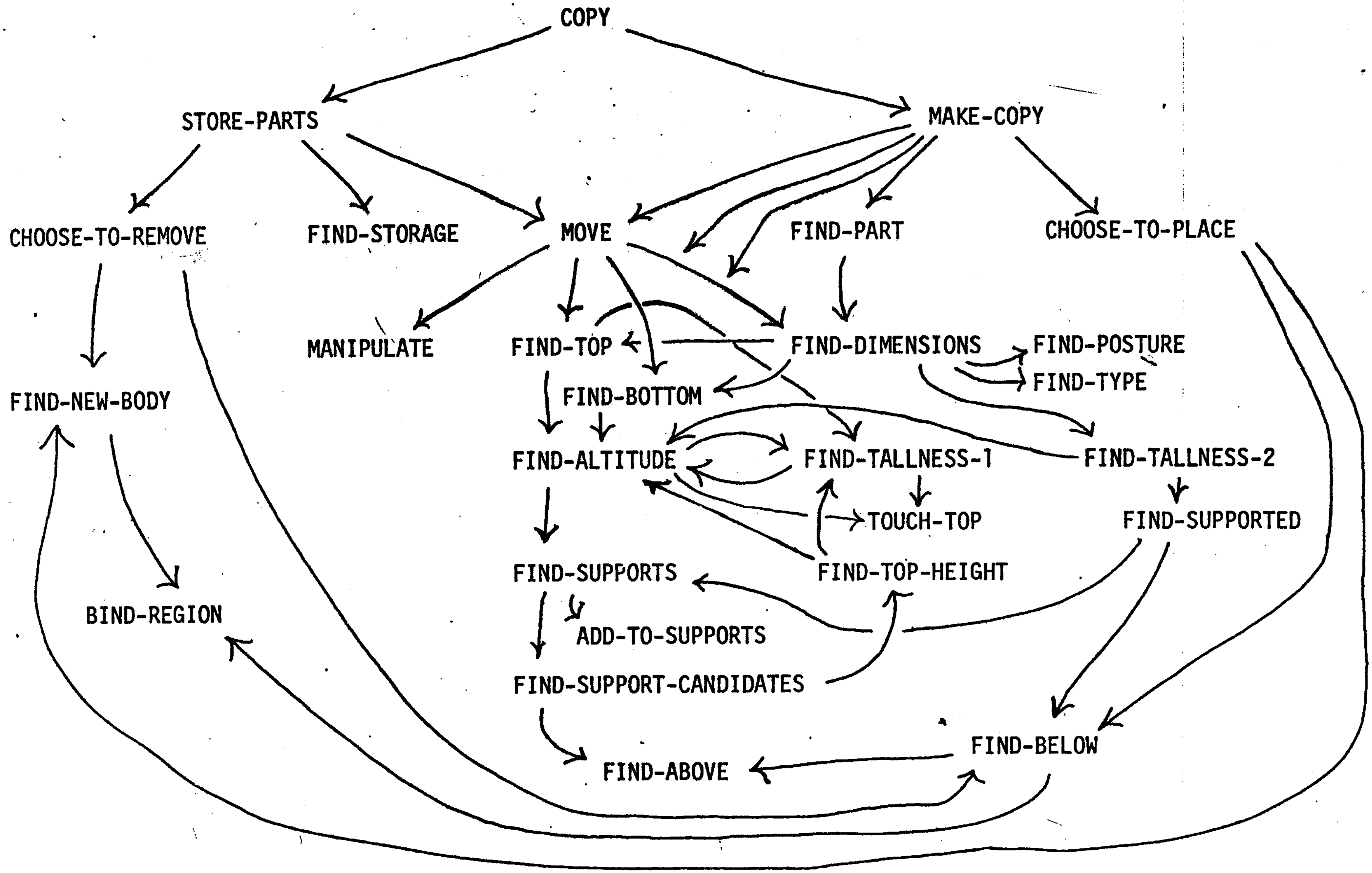


Figure 23
The vision system.

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

STORE-PARTS

To disassemble a scene and store it, STORE-PARTS loops through a series of operations. It calls appropriate routines for selecting an object, finding a place for it, and for enacting 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  B2)
(B2   IS-A BODY)
```

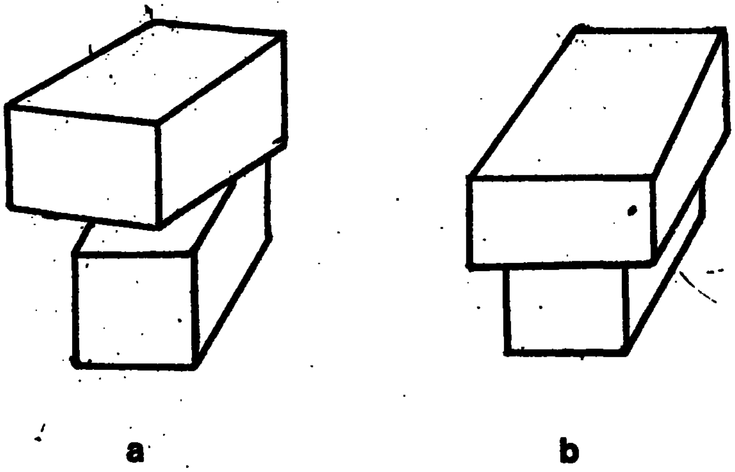


Figure 24
A source of spare parts and a scene to be copied.

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 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 unbodied 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 locations, and dimensions are gathered up. Then MANIPULATE interfaces to the machine language programs driving the arm. After MOVE succeeds, STORE-PARTS makes an assertion of the form:

(B12 IS-A SPAREPART)

FIND-TOP

The first task in making the location calculations is to identify line-drawing coordinates of a block's top. Then FIND-TALLNESS and FIND-ALTITUDE supply other information needed to properly supply the routine that transforms line-drawing coordinates to X Y Z coordinates. Resulting assertions are:

```
(B1  HAS-DIMENSIONS  (2.2  3.1  1.7))
(B1  IS-AT  (47.0  -17.0  5.2  .3))
```

Where the number lists are of the form:

```
(< smaller  x-y plane dimension >
 < larger >
 <tallness>)

(< x coordinate >  <y>  <z>  <angle>)
```

The x y z coordinates are those of the center of the bottom of the brick and the angle is that of the long x-y plane axis of the brick with respect to the x axis. Two auxiliary programs make assertions of the form:

```
                         STANDING
(B12 HAS-POSTURE         LYING          )



                         CUBE
                         BRICK
(B7       IS-A           STICK          )
                         BOARD
```

wherever appropriate.

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 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 objects 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.

```
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 1963) (Herskovits and Binford 1970) (Griffith 1970) (Horn 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 Brick," 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 Scemata) 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, B. 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, B. & 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 Shladows 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.

Sussman, G., Winograd, T., and Charniak, E., "MICRO-PLANNER Reference Manuel," A.I. Memo. 203A, Artificial Intelligence Laboratory, M.I.T., December, 1971.

Rattner, M., "Extending Guzman's SEE 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.

Wichman, 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.