Active Knowledge

VISION FLASH 53

by

Eugene C. Freuder

Massachusetts Institute of Technology

Artificial Intelligence Laboratory

Robotics Section

October 1973

Abstract

A progress report on the work described in Vision Flashes 33 and 43 on recognition of real objects. Emphasis is on the "active" use of knowledge in directing the flow of visual processing.

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-0362-0005.

Vision flashes are informal papers intended for internal use.

This memo is located in TJ6-able form on file VIS;VF53 >.

A Note to the Reader

Well, I thought it was about time to write something down again.

This paper should provide some material for those who wanted more details, or more examples, than they found in Vision Flash 43 (VF 43). See particularly, sections 3 and 6. However, I do not expect to really satisfy anyone in this regard. Moreover, this paper adds whole new parcels of hand waving.

Any way you look at it, it is difficult to write up work in progress. You do not want to go into details on uncertain explorations, but you do not want to completely ignore important issues either. As a result this paper will probably vacillate between cryptic and incoherent.

Nevertheless, it is healthy to write things down periodically. Good luck.

1. Introduction

This is a further progress report on a system whose motivations are found in VF 33 and origins in VF 43.

This paper will focus on the organization and use of knowledge in the system. We discuss the analysis of concepts and their procedural implementation. Briefly, we propose a "hierarchical relevance" analysis which is implemented in a data base of conjectures. Associated programming modules serve to establish the conjectures and guide the processing.

Examples are drawn from current code for recognizing a "tube" (fig. 1). The examples, and the system description are oversimplified and incomplete. One of the most encouraging features of the system is that concrete mechanisms readily suggest themselves for dealing with added complexities. However both the complexities and the mechanisms to handle them are largely theoretical at the moment. It is far too easy to get lost in theory and never complete a program, or paper. I shall try to confine myself here to a description of basic mechanisms, as exemplified by current code. I will not succeed; but I shall try.

You will recall from VF 43 that my basic interest is visual recognition in a realistic environment. Also that my approach derives from the heterarchy concepts of Minsky and Papert, and is summarized in the following pseudo-equation:

$$Pn+1 = f(GK, PK(n)) \quad .$$

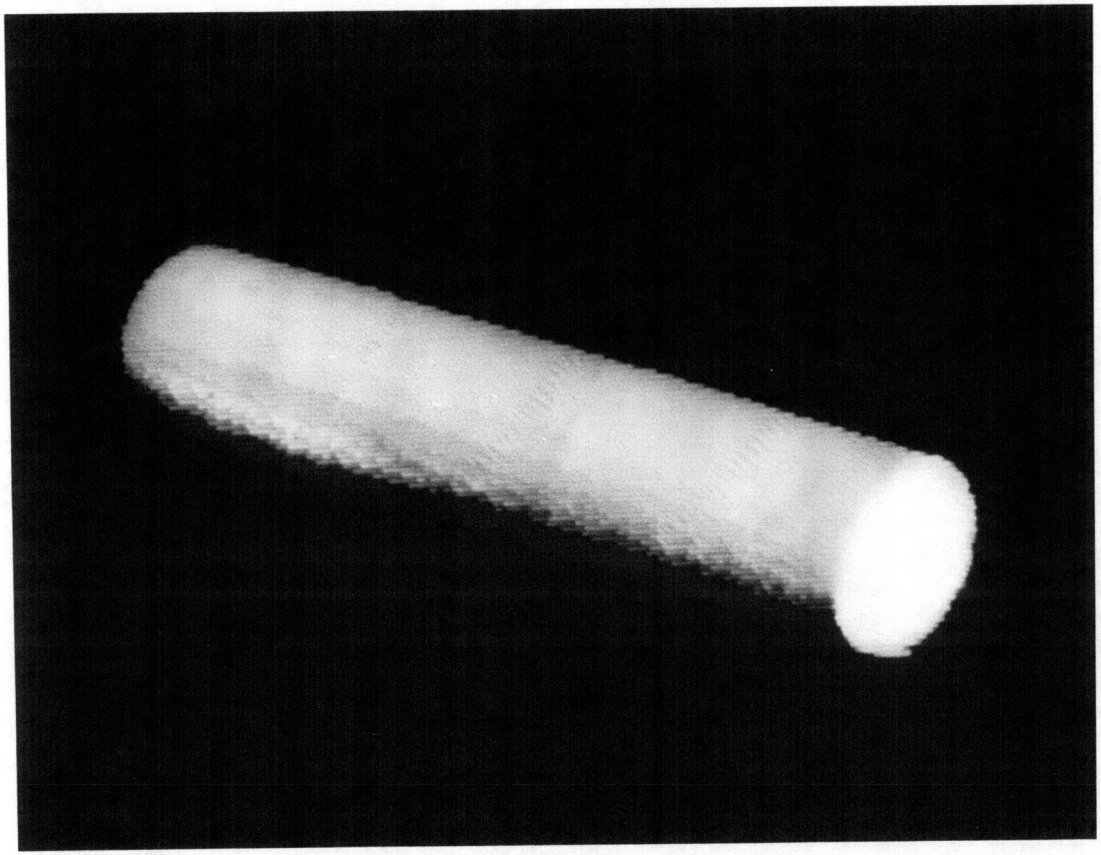


Figure 1

The n+1st processing step is determined by the interaction of our general knowledge (GK), and the particular knowledge (PK) that we have already gathered from the specific scene through the first n processing steps.

In VF 43 I began to analyze this "function" in terms of "suggestion" and "advice".

I also indicated that the representation of both GK and PK in our data base was important, insofar as it could facilitate the interaction refered to above. More generally, we need to represent knowledge in a form which allows and encourages its "active" use in the direction of the recognition process.

This "active knowledge" is to be distinguished, for example, from the passive use of knowledge in a pattern recognition or model matching scheme. These require us to gather all the particular knowledge about a scene we can, then attempt to elicit recognition by comparison with general knowledge imbedded in paradigms or models. However neither the general knowledge, nor the particular knowledge as it is received, is actively used to assist the knowledge gathering process—the most difficult task.

We will use the two-pass "build a description, match against models", approach as a straw man throughout this paper.

In my system the knowledge gathering process coincides with the recognition process.

I will call the system SEER, so I will not have to keep saying "the system". Possibly the name has some punnish

appropriateness.

The form in which knowledge is represented may serve to permit or prevent, encourage or deter its active role.

For example, a mathematically oriented model encourages us to view a relationship like "A next-to B" as a predicate to be tested on A and B.

I want to view "A next-to B" as providing advice on how to find B if I have found A--or A if I have found B. "A next-to B" also should suggest that "X" might be B, if I discover A next-to X.

Once we have an analysis of our knowledge which discovers and encourages its active roles, we can transfer it to a programming system which <u>institutionalizes</u> concrete mechanisms for implementing these roles. Here again, the form in which knowledge is imbedded in a system can encourage or deter its active function.

2. Knowledge Structure

The first thing we need to do is analyze our knowledge in a form which facilitates suggestion and advice. The analysis we make is basically a very simple one.

When people ask me how I describe objects, they generally seem to be thinking about methods for describing curvature, or elegant schema for specifying physical structure and semantic features.

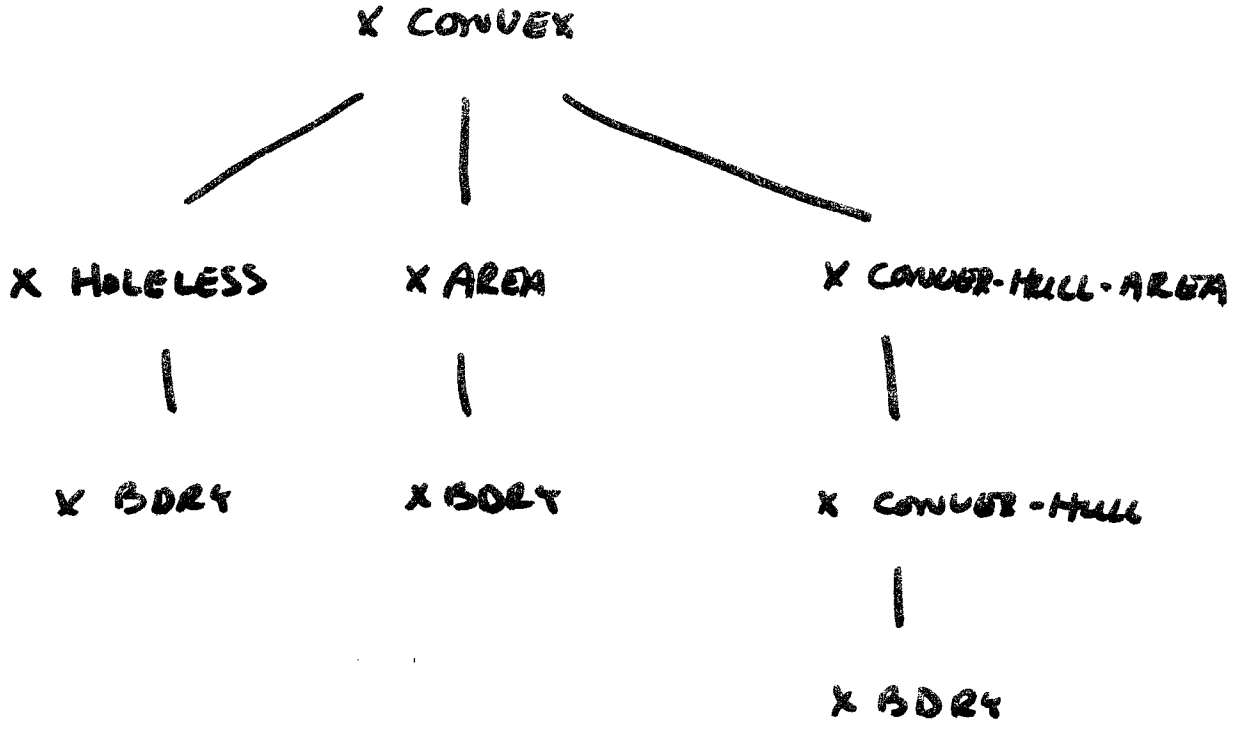
These are important issues, but not, I feel, the central one. It is not really that difficult to make the distinction between a hammer and a screwdriver. Elegance or fine detail is not essential.

Barrow and Popplestone's "cup recognizer", for example, a pioneering effort, did not fail to recognize a hammer because their description of a hammer was not rich enough, or did not specify a hammer completely enough. Rather the system could not make use of its description of a hammer when it was needed—when the scene was being perceived.

We concentrate, therefore, for the moment, on the aspect of our "descriptive" knowledge most directly relevant to its active use. A truly "procedural" description.

Figure 2 presents a possible, simplified, organization of the concept "convex".

One process for determining convexity involves finding the area of the region, R, and the area of its "convex hull" (a



where the link in : P — Q

indicates that Q is-relevant-to P

concept of my own which is probably equivalent to the standard one, i.e. the smallest convex region containing R). If these areas are close enough, it is fairly likely the region is convex. I also require convex regions to be without holes.

This "definition" or process for establishing convexity is reflected in figure 2. There are two basic aspects of this structure to notice.

First, the analysis is based upon "relevance" or "possible relevance". "Relevance" is a suitably vague concept whose dimensions we shall come to know by example. I could define it by indicating that "x is relevant to y", if the presence of x in the data base for a scene, might suggest that we look for y in the scene. However, this definition would prove to be rather circular.

It is important here to note that relevance is not limited to the necessary and sufficient sub-parts, sub-properties or "defining conditions" of a concept or object. We do not seek any "minimal" set of defining conditions. In fact, we hope to be able to use redundancy and alternatives. Relevant sub-elements may include factors of advice or context. We shall examine these possiblilities a bit further in later sections.

Second, the analysis is hierarchical. Convexity involves three major items; these, in turn use...

How exactly this analysis is organized hierarchically, and what elements are singled out as nodes, are decisions that can only be understood in terms of our overall aims, and programming

mechanisms.

Most importantly, this structure must facilitate our suggestion mechanisms. Therefore commonly used sub-processes are isolated as "nodes", so that we may suggest all their relevant "superior" processes. Processes are also isolated in order to permit suggestions to be made in precisely the proper place in the structure. For example, if we discover a boundary we need not immediately suggest convexity, we can test if the region is without holes first.

But the organization will also serve other purposes when transferred to our programming structure.

It will facilitate sharing and interaction of information.

The hierarchical organization directs the flow of processing, in serial and parallel streams.

Most concepts will be complex enough to organize alternatives into parallel "methods" for establishing the concept.

These various complexities are best understood in the context of the remaining sections.

3. Programming Structure

In this section we discuss the manner in which a hierarchical relevance analysis is implemented, in code and in the course of processing a scene.

This is a straightforward section; at least the first half is. We describe the basic mechanisms and how they operate. That this operation is of a rather different nature from normal programming structure should be obvious. However we will save most discussion of the character of what is operating here for later sections.

3.1. Modules on Datums

3.1.1. Introduction

Every concept or predicate that can appear as a "node" in a hierarchical relevance analysis, has an associated set of program modules. There are five primary types: conjecture (CONJ), establish? (EST?), establish (EST), FAIL? and FAIL modules.

When an instantiation of a predicate is hypothesized, e.g. R69 CONVEX, region 69 convex, it is placed in the data base as a Conniver "datum", and marked as a "conjecture". All conjectures go on the suggestion list with an appropriate priority. If the conjecture is later verified it is marked "established". It is removed from the suggestion list, but remains in the data base.

The program modules operate on these datums. More precisely the set of program modules associated with a datum's

predicate act upon the datum. Thus the CONJ module "CONJ-CONVEX" acts upon the datum R69 CONVEX.

The essential operation of the system consists of applying modules to datums. We will now discuss how and when these modules get applied and what functions they serve.

3.1.2. CONJ

When the monitor chooses to execute a conjecture for the first time, the appropriate CONJ module is called to act upon the conjecture. E.g. CONJ-CONVEX acts upon R69 CONVEX.

There are two standard simplest forms of behavior that a CONJ module may exhibit.

The module may suggest "pursuing" all further conjectures that might help establish the current one. E.g. CONJ-CONVEX acting on the conjecture R69 CONVEX, makes the following further conjectures: R69 HOLELESS, R69 AREA, R69 CONVEX-HULL-AREA. These conjectures are placed in the data base and on the suggestion list.

If the module is at the "lowest" level, it will simply execute some "black box" calculation, and call an EST module upon success (or a FAIL module upon failure).

3.1.3. EST

An EST module gets executed whenever a conjecture is established. (We will see how higher level conjectures get established in a moment.) For example, when R69 CONVEX is established, EST-CONVEX is applied to the datum.

The standard simplest behavior for the EST module to

exhibit involves "proposing" further suggestions. The EST module knows what the established datum is possibly relevant for, and places further conjectures in the data base accordingly. For example, EST-HOLELESS acting on R69 HOLELESS would propose R69 CONVEX.

The reader will recognize the EST modules as a mechanism for implementing the basic form of "suggestions" introduced in VF 43.

The CONJ mechanism places normal "downward" processing in the hands of the same Suggestion List/Priority System/Monitor apparatus. (This consistency of approach may have been encouraged by an early conversation with Jeff Hill.)

All very well you say, but when does anything actually get computed around here (aside from low level CONJ's)? There is room in CONJ's and EST's for additional hair, but the greatest burden falls on the EST? modules.

3.1.4. Links

I must first explain that when CONJ And EST modules place new conjectures in the data base, these conjectures are "linked" to the datums which inspired them. That is, R69 CONVEX is linked to R69 HOLELESS, regardless of whether the former pursued the latter, or the latter proposed the former. R69 HOLELESS is thus noted as "relevant to" R69 CONVEX. These "links" are simply the "possibly relevant" links discussed in section 2 and figure 2.

The CONJ and EST mechanisms, then, place instantiations of

our hierarchical relevancy analysis into the data base, and before our control structure. (There need not be a simple 1-1 correspondence between our analysis of the knowledge structure and our implementation of the programming structure.)

Of course when a suggestion is pursued or proposed it may be found to be already in the data base. It may be already linked or even established. This provides a mechanism for "sharing" facts and conjectures, for interacting with established data and parallel investigations.

3.1.5. EST?

Whenever an EST module changes the status of a datum, D, from "conjecture" to "established", it activates the EST? modules for all the conjectures immediately "above" D in the relevance structure. These "superior" conjectures may be ones which were already present and linked to D, or which were only just now linked or created by the "proposing" efforts of the EST module.

An EST? module generally checks to see if certain relevant conjectures have been established yet and if so may perform some computation on the now available results.

EST?-CONVEX acting on the conjecture R69 CONVEX will check if R69 HOLELESS is established. It will also check on R69 AREA and R69 CONVEX-HULL-AREA. If these have been established, it will compute their absolute difference and compare this with some threshhold.

If the two areas are close enough, the EST? then succeeds

and invokes EST-CONVEX on R69 CONVEX. If some of the necessary conjectures bearing on the decision had simply not been established yet, the EST? would have returned to try again another day. If some tests fail, and there are no further alternatives available to the EST? (for example, an EST? is even capable of pursuing new conjectures), the EST? will call a FAIL module.

3.1.6. FAIL?—FAIL

There are also FAIL? modules, and FAIL and FAIL? modules are rather analogous to EST and EST? modules. A FAIL module will change the status of a datum to "failed" and call FAIL? modules for the conjectures immediately above it in the relevance structures. FAIL? modules check failure conditions for a datum D and call a FAIL module for D if failure is demonstrated.

FAIL modules can also propose new suggestions as do EST modules. And "failed" datums remain in the data base, so marked for future use. (All data is subject to some sort of eventual "garbage collection", of course.)

3.2. Example

Figure 3 presents a printout of the program's reaction to the conjecture R1 BDRY. (Incidentally, the boundary is placed on the property list of the conjecture when found.)

SEER is operating here solely within the context of the knowledge illustrated in figure 2. Because of the almost degenerate nature of this limited structure, everything

```
WORKING ON THE CONJECTURE R1 BDRY
R1 BDRY ESTABLISHED
R1 BDRY PROPOSING R1 HOLELESS
PLACING R1 HOLELESS ON SUGGESTION LIST
R1 HOLELESS ESTABLISHED?
R1 HOLELESS ESTABLISHED
R1 HOLELESS PROPOSING R1 CONVEX
PLACING R1 CONVEX ON SUGGESTION LIST
R1 CONVEX ESTABLISHED?
NOT YET
R1 BDRY PROPOSING R1 CONVEX-HULL
PLACING R1 CONVEX-HULL ON SUGGESTION LIST
R1 CONVEX-HULL ESTABLISHED?
R1 CONVEX-HULL ESTABLISHED
R1 CONVEX-HULL PROPOSING R1 CONVEX-HULL-AREA
PLACING R1 CONVEX-HULL-AREA ON SUGGESTION LIST
R1 CONVEX-HULL-AREA ESTABLISHED?
R1 CONVEX-HULL-AREA ESTABLISHED
R1 CONVEX-HULL-AREA PROPOSING R1 CONVEX
R1 CONVEX ALREADY PRESENT
R1 CONVEX ESTABLISHED?
NOT YET
R1 BDRY PROPOSING R1 AREA
PLACING R1 AREA ON SUGGESTION LIST
R1 AREA ESTABLISHED?
R1 AREA ESTABLISHED
R1 AREA PROPOSING R1 CONVEX
R1 CONVEX ALREADY PRESENT
R1 CONVEX ESTABLISHED?
R1 CONVEX ESTABLISHED
THAT IS IT

NIL
```

described in figure 3, actually occurs without further reference to monitor or priority system. But for this same reason, it is perhaps a good example of the actions of the programming modules.

3.3. Hair

I have presented a data base that treats facts, failures and program elements in a uniform network structure.

I have described a programming structure that functions by applying modules to conjectures in this data base.

SEER never demands that much be done, it just makes a lot of helpful suggestion. A good boss.

This basic structure is not too hairy, I hope.

If you like hair, however, there are enough hooks here to open up a wig salon.

I really do not want to get into a discussion of unmotivated and unfinalized hair. However, perhaps I should just indicate a few of the problems and possible solutions. I do maintain that there is enough structure here so that specific programmable solutions arise easily to confront theoretical or practical problems.

The most motivated of these problematical issues arise in attempting to program actual knowledge into the system. We have to develop specific mechanisms or techniques for dealing with part-whole relationships, for example, to allow parts or whole to be analyzed in any order and results to interact. These are often, in some sense, technical issues.

More basic demands for hair generally are related to a felt need for more "serial" control of the processing.

These needs can often be met by suitable organization of our hierarchical relevance structure. However, this may get awkward at times.

In particular the process of generation, of arguments for predicates, or candidates for objects, sometimes seems to call for more serial control. As Sussman and McDermott point out, it is sometimes painful to generate an entire set of possibilities at once. We would prefer to generate one, test it and if it does not meet our needs return for another. Sometimes also we have a sequence of operations, where it is difficult to instantiate the later steps, until the earlier ones provide arguments to apply them to.

I should give some examples, but I really do not want to get into this here. As I say, more specific control of the sort offered by the Conniver "tag" mechanism may sometimes prove useful.

Which leads up, of course, to the observation that Conniver tags can be rather naturally added to the operation we have described.

Normally when we want to apply an EST? module, for example, to a datum, we call on a program called EST? to cons together the call to the appropriate EST? module. Now once this EST? module has been applied to the specific datum, we have a process, that can be tagged and saved, after the Conniver

fashion. The tag can be saved as an EST? property on the datum's property list. Now the next time we apply EST? to the datum, it will check the property list, find an EST? property, and go to the tag to resume the interrupted process. Thus we can have several instantiations of the same EST? module hanging alive in the data base.

A quasi-example of how this feature might be used. We may not desire, or be able, to pursue, i.e. add to the data base, all the sub-conjectures of a conjecture, C, at the original application of the CONJ module to the datum. Perhaps one sub-conjecture provides an argument for the others, or we just do not want to get involved with the others unless some "triggering" datum is found first. (Similarly this trigger could be the only datum capable of proposing C from below.) One way of dealing with this is to use tags. There are ways of putting this hair in the CONJ, EST? or EST modules.

Some sub-conjectures could get "pushed" the first time; upon their success others could get pushed.

In fact, we could obviously do this sort of thing in the EST? modules even without tags. And there all sorts of other possibilities, ranging from methods of organizing the hierarchical relevance analysis to utilizing the priority and monitor system, or even implementing "EXEC" modules.

Basically, I tend to favor placing distinct stages of processing in distinct datum nodes and modules as far as possible. I think this approach is clearer to think about and

program. It protects against insularity, i.e. prevents us from getting tied up in a parochial process for long without interacting with the "real world". In summary it may be more conducive to the kind of active knowledge interaction we are hoping for.

However, I intend to let real problems in visual processing motivate theoretical problems and advise the nature of appropriate solutions, as much as possible.

Another whole field of hair would involve "second order" hacking on the network of conjectures. Cutting out contradicted conjectures. Restructuring investigations dynamically. Making non-local changes in the network, i.e. not just on nodes immediately above or below a current node.

Well, I hope that is enough programming detail and hand-waving hair, for now. The remaining sections will discuss some of the features of the system, and present a few examples.

4. A System

The first point I want to make is that SEER is a system.

The Vision Group has produced several good examples of heterarchy in action. These date back to Minsky and Papert's original suggestions for proposing missing lines that reflected work by Blum and later Wolfe and others, and were implemented by Freuder for the Copy Demo.

Shirai and Wizard use heterarchy to guide tracking. Finin and Lozano have several extensive examples of higher level guidance or presumption.

All these, however, are basically isolated examples.

Consider the parallelogram heuristic in Wizard.

The assumptions, of a rectangular parallelipiped environment are implicit. There is no provision for making them explicit. If the parallelogram attempt succeeds the assumptions are not reinforced. If it fails they cannot be questioned or alternatives taken. Assumptions cannot be added or removed based upon experience with the scene.

Perhaps we have a wedge? Can we propose a line to complete a triangular face?

Well yes, of course, a certain amount of additional heterarchy "hacks" could be added to Wizard. However without a systematic structure to receive and organize these additions the program would soon fall under its own weight. Like the "limited logic" programs for natural language that Winograd criticizes,

they would become impossible to program, or else remain hopelessly crude.

The problems are such, that we do need to be able to make additions incrementally; but to a system that can receive them: new objects, new views, new difficulties. We cannot hope to write a "complete" program from the start. Not only is there an effectively infinite variety of objects; not only does each have a number of different "views". Each object can also appear in a wide variety of contexts or instantiations, different lighting, surface texture, etc. And then there are the endless combinations of objects into scenes.

There are far too many possible interactions for the programmer to individually consider every time he adds a new facility, or considers a new scene. The system must provide some automatic mechanisms for facilitating these interactions and tailoring the processing to the scene.

It should be fairly obvious (hah) how the mechanisms of SEER meet these challenges to provide a system, in which a parallelogram hack can coexist happily with a resistor hack, and both can be invoked at appropriate times with appropriate priority.

The examples used in section 6 may serve to illustrate this claim further for the skeptical.

A few other features of a good system. (Did someone mention SEER?)

It provides standard formats that make some tasks routine,

and lifts other routine burdens entirely from the user.

It provides concrete mechanisms for heterarchical interactions and "institutionalizes" and encourages forms of heterarchy like advice.

It offers hope of being able to assimilate advances in areas like shape description on the one end, and "frames" on the other. In the meantime it makes best use of crude understanding.

Let us take, for example, the question of multiple views. This is a longstanding visual description problem: do we need a separate description of each distinct view of an object, or is there a single model that can be perturbed properly to handle any view?

How does SEER handle multiple views?—asks someone in the back of the hall.

Well, first perhaps I should distinguish the "system" from its present state of knowledge.

The system provides ways of dealing with knowledge, enabling knowledge to interact with a scene to produce suggestions and advice that guide processing.

If someone discovers a satisfactory "unified" description of objects, presumably this representation could be used by the system. In the meantime, the system is particularly adept at handling information in modular form, and so makes good use of the distinct descriptions we must provide of distinct views. (If really pressed, I would be forced to admit, to my future

chagrin no doubt, that I should not be surprised if no satisfactory "unified" model exists; and SEER is thus all that further ahead of the game with its modular capabilities.)

What means "modular"? Well, we can treat each view of an object as a separate "method" of recognizing or establishing the object. These approaches can be pursued serially or in parallel. They can be proposed individually or in groups, by previous results in the processing of the scene. They can be programmed separately, added incrementally, at different times, to the system. And yet SEER mechanisms facilitate the sharing of information and aid between the methods, and benefit from partial successes or failures, to direct attention to the winning approach.

In short SEER facilitates programming and makes the best interactive use of the disparate knowledge it has.

I have not discussed the Priority and Monitor aspects of the system. This is because I do not intend to discuss them. These are beyond the scope of this paper. I only wish to point out that they provide additional hooks, for means to deal with additional complexities.

5. Parallelism

One of the obvious features of SEER is its parallel aspect. (I will not debate the meaning of parallel here.)

The "data base" consists of a network of current conjectures, established conjectures and failed conjectures. These interact, and SEER may move about among them according to priority and interest.

The dissection of concepts into a relevance network of communicating sub-elements, permits SEER to go away and return, or "multiprocess". (The primary role of this network is in the suggestion structure, of course.) If desirable, the Conniver tag mechanism can be employed to further facilitate this type of processing. However the representation of our programming elements by existing structures in the data base, can make "frame-saving" unnecessary.

Now I am not going to attempt here to proclaim the superiority of a parallel facility. But perhaps I should provide some motivation, some indication that parallel processing structures are worth studying in a visual environment. It may be that SEER will say a little about the desirablility of parallelism, or the nature of parallel and serial processes, which are really needed where. At least in vision. Some negative results if nothing else. For now let us at least present some suggestions that parallel structures <u>might</u> have a role to play in visual processing.

At the "lowest" levels, there are, of course, neurophysiological motivations for studying parallel structures in vision. Bert Horn's latest results in perceiving "lightness", suggest a parallel implementation. He is working closely with Dave Marr, who has neurophysiological theories with some parallel features.

My work in region finding, and other work before me, has a parallel flavor.

On an even higher level, we have Dave Waltz's efforts. I feel that his program can be viewed as carrying forward all possible conjectures, about his class of scene predicates, in parallel.

Most notably, this did not result in an exponential explosion. Rather as more of the scene was viewed, and more types of scene predicates, more classes of knowledge were understood and pursued, the process converged to the correct analysis.

For those who still protest that it must be wasteful to pursue so many conjectures, let me point out that the only systematic alternative, our straw man, model-matching approach, must be worse.

Let me present a semi-serious "proof" of this, a "proof" that heterarchy works.

We have to consider first of all the alternative. It may seem very wasteful to make a lot of suggestions in parallel. However, our straw man processing, build a description and model

match, would really in some sense have to compute everything. All possible properties and relationships have to be computed since we do not know which might be needed to match some model.

At least in SEER, we have some hope of not computing everything.

Consider a worst case simple model of the system in operation.

A region R has property p1. This proposes that the region satisfies predicates in the set S. So we now have to pursue all the properties P of all the members of S. But these are not necessarily all the properties that could be tried for R; and would be tried in our straw man system. They at least have some likelihood of success. There may be properties Q that nowhere coexist with property p1; why check for them?

Further, remember these properties P are not really pursued all at once in parallel. They are pursued according to a priority and monitor system. And our hierarchical structure is organized so that we proceed upward in appropriate small chunks, and discriminate quickly anong large sets of possibilities.

As we determine more properties of R, more suggestions are made; but we really cut down our possibilites, in a Waltz-like way, as we learn more and proceed upward.

We proceed upward in stages: p1 and p2 establish q1; q1 proposes r1, which pursues q2; q1 and q2 establish r1;... A jagged procession, two steps forward, one step back.

But in many cases we intersect, contradict and halt.

"Round" suggests a lot of things, "red" suggests a lot of things, but only items in their intersection, e.g. apples not fire-engines, remain. Fruit bowls, not fires, will be proposed next. Furthermore the conjecture that a round, red region is an apple will have a higher priority than a conjecture that it is a fire engine, simply because we have two pieces of evidence for it.

For a simple example of how hierarchical organization can affect efficiency and order of processing, apart from priorities, consider again the analysis of convex expressed in figure 2.

Now, as the analysis stands, the discovery of the boundary starts three upward proposals which eventually lead to convexity.

We could instead organize convexity as shown in figure 4.

Now suppose we find the area of a region R somehow, without involving the boundary, or a proposal that R is holeless. (Regions in the initial data base, for example, have their area calculated as they are grown.) If we program the arrangement shown in figure 2, area will propose convexity, which will pursue holeless, area (have), and area of convex hull. Now this means that holeless will probably get tested on the region. (There are, of course, other variables, like order placed on the suggestion list, operation of monitor, etc.) When area of convex hull is found and compared with area, we may find that the region is not, after all, convex. Our holeless test was

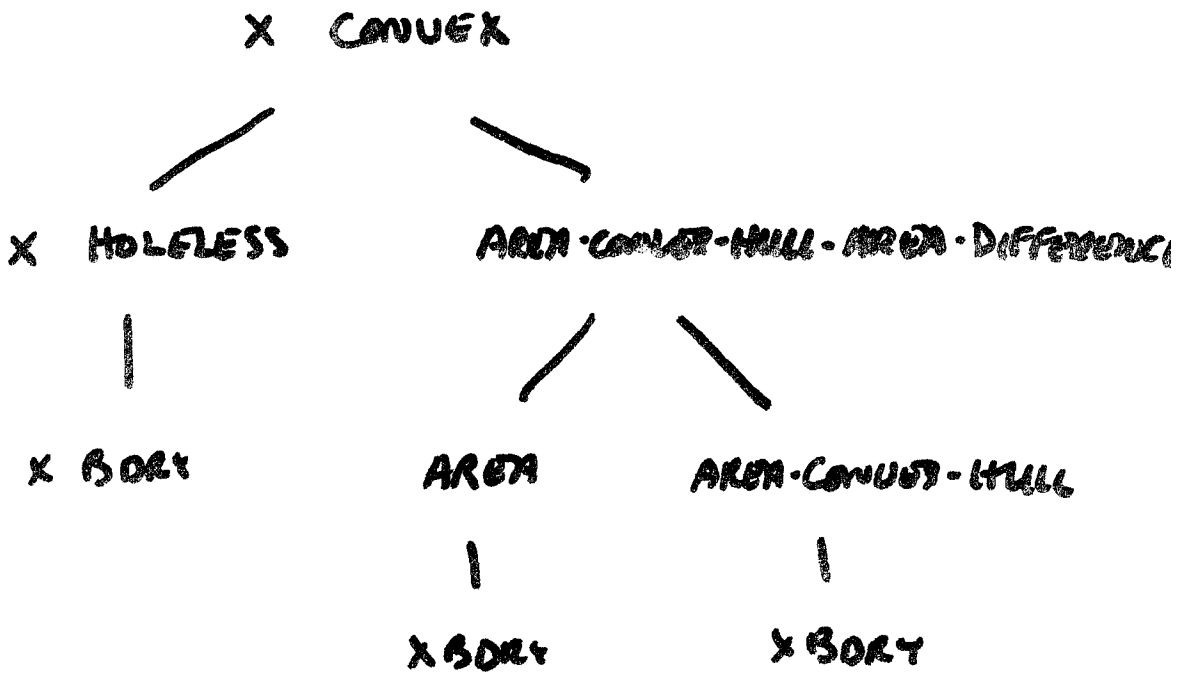

X CONVEX
X HOLELESS
AREA-CONVEX-HULL-AREA-DIFFERENCE
X BOX
AREA
AREA-CONVEX-HULL
X BOX
X BOX

Figure 4

"wasted". (Actually in SEER, it would hopefully have suggested other possibilities.)

If we had programmed the organization in figure 4, area would have suggested area-convex-hull-area-difference. The area, convex hull area comparison would have been made first, and the holeless test not made unless the comparison succeeded.

Ah, but, you say, that is nonsense. What we want to do, is not make the costly areas comparison until the trivial holeless test has been made.

The point is that our method of processing, while it may seem at first to propose endless possiblities, in fact has means of ordering and cutting down alternatives.

A straightforward "build up a description" system, has no advantage "a priori". In its simplest form, in fact, it computes everything, where our approach at least cuts down a bit. One, of course, could organize a "build a description" method in various ways to cut down processing. But one can do that in SEER, and perhaps more naturally and automatically.

The fact that some of these methods for guiding flow of control more intelligently may involve serial mehanisms, does not bother me. The hierarchical aspect of the analysis SEER uses, is a serial mechanism. The suggestion list is a serial list. The system is not really parallel.

It does have some definite parallel and multiprocessing flavor. In some respects this is almost a side effect of facilitating the suggestion and advice interaction of active

knowledge.

However, if analysis points to definite needs to reintroduce "serial" control, e.g. through the use of tag mechanisms, I shall feel instructed, not disappointed.

Ultimately I hope the parallel mechanisms will show some advantages. If not, I would expect to be open to the criticism that the parallel aspect of the system "gets in the way". However I would hope to rebut this by demonstrating how natural this organization is for "active knowledge".

Basically, we observe that, for the problem of recognizing a visual scene, the data is presented in a more parallel fashion than it is presented for most "higher level" problems of "intelligent" processing.

It is reasonable then to attempt to employ and study parallel mechanisms in this area. If they are to prove useful at all, one would hope they would show up to advantage here.

6. Suggestion and Advice

6.1. The Heterarchy Gap

A.I. research has explored the extremes of heterarchy. The broad idea of heterarchical interaction on the one hand, specific "heterarchy hacks" on the other.

What we need are a sequence of ideas that bridge that gap, and some specific mechanisms for implementing and "institutionalizing" these ideas within a coherent system. These concepts are necessary to fully understand and utilize heterarchy. They would ameliorate the difficulty we face when looking for examples of heterarchy. (For a long time it was essentially: "line proposing—ok, name two".)

VF 43 focused on "suggestion" and "advice" as two useful sub-concepts in the analysis of heterarchy. I will here analyze these, in turn, a bit further; and provide some more concrete examples. Not enough to be sure, but an indication.

One can say: apply heterarchy to the problem. It should prove easier to do this when we can say: heterarchy involves suggestions; one class of suggestions is "part suggests whole"; for this problem that means...; and here is a mechanism for programming it into a system.

6.2. Advice as Suggestion

The reader may have recognized some confusion in the distinction between "suggestion" and "advice" in VF 43. I have now come to recognize that the intended distinction, between

"what to do" and "how to do it", while useful in some ways, is essentially misleading.

Advice on "how" to do something is really advice on "what" method to use.

This observation has more than quibbling implications for SEER. My original approach to advice involved advice "modules" (like the modules in section 3) for giving and/or taking advice. This approach still has some attractions. However, once we recognize that advice can be viewed as a suggestion on what "method" to use for some piece of processing, P, we can treat advice in the same way we do suggestions. We can separate out P and its distinct methods, as nodes in our relevancy structure. The "adviser" is made into a node. The EST module for the adviser suggests the appropriate method for accomplishing P, i.e. advises how to establish P.

It is at the same time natural that the existence of some advice for a method of establishing P, is one of the relevant items which "suggest" the method, in the usual sense of "what to do next"—placing it on the suggestion list or increasing its priority.

Thus suggestion and advice are treated uniformly, through the relevance analysis and suggestion mechanisms.

Notice also that the advantages of the system in terms of interaction and sharing of general and particular knowledge are available to elements of advice.

The remarks I made earlier about the modularity of the

system, with regard to distinct "views" are relevant here. (In fact one piece of "advice" might be which view was suggested.) Different "methods" may be programmed and executed distinctly, serially or in parallel, yet still share and interact. The possibility of advice, improves our ability to choose among these methods. Interactions are still important though; in fact, failure of one method may advise another.

Winston's suggestions in his thesis about moving from model to model in a similarity network should be explored in this context.

6.3. Example

I suspect we are overdue for a "brief word from our sponsor", i.e. a simple example.

Suppose we are trying to recognize the side of a tube, having already identified a region, R, as having the overall properties of a tube "as a whole".

Now the side of a tube has a number of necessary properties. If we were simply given some region and asked to decide if it was a tube-side, "in vacuo", these properties would all have to be checked out. However, some of these properties are implied by the properties of R that we have found already. For example, in finding that the boundary of R is tubelike, we have already established relevant information about three sides of the tube-side boundary.

This knowledge may not be in easily "shared" form. Data base "assertions" that involve R, as a tube, may not simply

"match" needed facts about a tube-side. Some "higher level" sharing is required. We could have a conjecture access a method capable of "deducing" an assertion from such related facts, when a simple data base match failed.

In SEER the presence of the established facts about R advise (suggest) a method for establishing a tube-side for R. This method assumes what is already available, and merely seeks to verify a few more details.

We already have a "profile" of R, segmenting it into two (three-dimensionally) straight surfaces lengthwise (fig. 5). We take additional profiles on either side. Some may be affected by noise, or fall along a line where intensities on side and face are equal. However, say two out of three, should coincide, to enable us to verify a distinct side of uniform length and curvature.

This example, meager as it is, has already raised several issues, which we explore in the next four sections.

6.4. Parts and Wholes

It is useful to distinguish the whole as, if not more than, at least different from, the sum of its parts.

Observations we make about the whole can assist our investigation of the parts or vice versa. Also, of course, one part can give advice to another.

We may find any part, or the whole, in any order, depending on what stands out in the given scene.

In the case of a tube, the end may blend into the

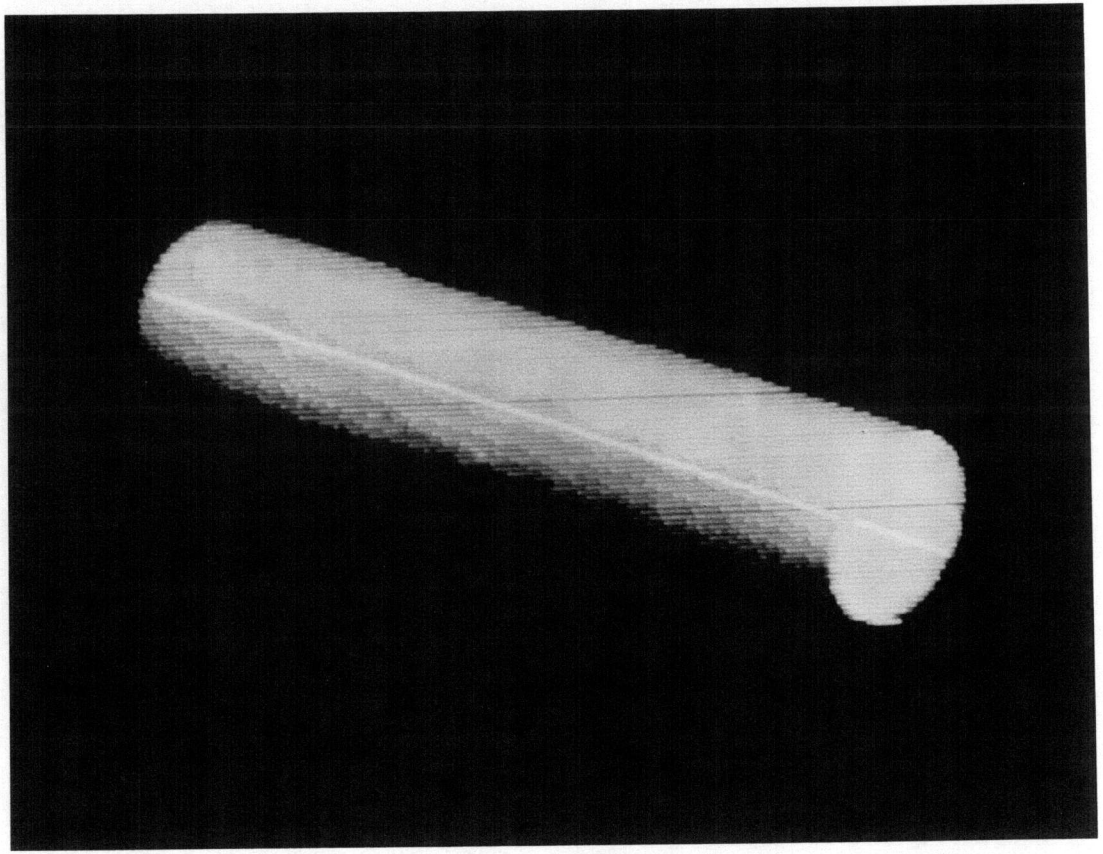


Figure 5

background, causing us to pick out the side first. The end may stand out as much brighter. The two parts may blend together as the whole stands out against the background.

We should be prepared to attack the recognition task in any order, and our tactics at any stage should be advised by what we already know.

It is not simply a matter of one part "standing out". Or even of special features, like highlights, attracting attention. Other parts may be difficult to find, or "missing". Advice and support is needed from the features most easily available.

In particular, "model matching" approaches have difficulty with occlusion, partly because they are oriented toward "whole" models. Our system quite naturally expects to deal with parts that "suggest" wholes before all the evidence is in. Partial or occluded objects can be suggested directly; or the programming modules for the whole can explore occluded alternatives, or attempt to "explain", justify or ignore discrepancies due to occlusion, and other problems. The following sub-section is also relevant here.

6.5. Effects

What we already know about a conjectured object can affect how we continue to analyze it in many ways:

1. What we look for.
2. How much we look for.
3. How we look.
4. How hard we look.

5. How easily we are convinced.

All these effects can be induced by advising appropriate methods for establishing remaining conjectures.

I will not expand upon this outline here.

6.6. Generation

We did not simply have to demonstrate that some region was the side of a tube. We had to find or "generate" the side, and prove it was the side. In some cases this might mean conjecturing that some "syntactically generated" region, e.g. from the initial data base, was the side, and applying tube-side predicates to it. In other cases we generate some "new" region. In any case, we use some of the semantic properties of the desired region, what we are looking for, to guide the search for an existing region or the generation of a new one.

In fact in the course of generationg a region for some object we may use all the properties of the object. Generating the region and establishing its identity merge.

Or, as in the case of the tube-side above, we may not even generate a region in the normal sense. We have no set of boundary points, or any other "complete" description of the tube-side.

In these cases we see most most clearly process of perception and recognition merging as I had hoped they would in VF 33. What we perceive, and the description of the tube side, are no longer unique abstract models. In this scene we first saw the tube as a whole. The tube side was perceived as a few

further region profiles. The "tracks" of these processes form the description of the side.

When we get to the end of the tube, all we now need to establish is a flat region running the width of the tube, and somewhat narrower in the other dimension. A few region profiles suffice to perceive and recognize the end.

To summarize, if the tube as a whole stands out and is perceived first, only a few observations, like a flat region at the end, complete our recognition. In fact, in a line drawing of a tube, merely the presence of the line between the two parts, side and end, together perhaps with line drawing conventions for implying curvature of surfaces, will imply a tube.

The problem of "generation", of regions, or other arguments for conjectures, is also one of the key "technical" issues for a system like SEER.

6.7. Relations

The use of relations, e.g. between parts of an object, as a source of advice, rather than just something to be proven, was discussed in VF 43. We can see from the discussion above that this process is being further explored. The lack of obvious checks above on the relations between parts and between parts and whole, in fact reflects the subtle use of relations in "generation" and advice on "methods". This is their natural "active" role. When description is tied up with the process of recognition, relations cannot sit around as labelled links

between pieces that have not been found yet.

However, I will defer a clearer and extended exposition of their role.

6.8. Under the Assumption

Advice is often a matter of "context". Lighting, for example, or previous experience with the scene, ideas about what objects are present, can advise particular methods. We will say a little more about context in general terms in section 7.

In particular, our system directs us to operate within the context of specific conjectures. We may make some observations about a region, using generalized methods, that suggest it is a tube. We conjecture that it is a tube. When we execute that conjecture we are able to determine if other tube-like features are present by methods which assume a tube context.

If you like, this is a generalization of the verification idea. E.g. line verification can be easier and more sensitive than line finding. We are able to ask "is this x" or "verify that this is x"; rather than the more difficult "what is this".

6.9. Example

Let us explore this notion with an example.

Consider the outer boundary of a tube. Our "straw man" type of processing would act on this boundary in the following manner.

First the feature points of the scene would be found and organized into lines. Then the boundary line would be segmented, into corners, straight lines, "uniformly" curved

segments perhaps, or some more complex description of curved boundaries. Then possible predicates would be tried on the pieces to determine possible relations. The two straight lines would be found to be parallel and of the same length, the two curved segments of the same shape and size and oppositely directed. Finally the built up description of the tube boundary might be matched against models of object boundaries.

All these operations, from line finding on up, are difficult to excute in a general context. We may have serious failures along the line.

In SEER it is likely that when we go looking for a tube boundary, we are pursuing the conjecture that some region is a tube.

We also have, or will look for, symmetry in the region. The two symmetric axes constitute the length and width axes of the region.

If the region has a tube boundary, it will have a straight side parallel to the length axis, and just a little shorter, running through the endpoint of the width axis.

With this position pointed out, we can proceed to "verify" the line in several ways. We can employ a standard line verification or tracking technique. In this case we in fact use a trick involving the boundary of the region. A general method might search for local maxima and minima of boundary points about the center of gravity, with the attendant problems of a local process. Instead we find all boundary points close to the

hypothesized line, and use this set to find the endpoints and verify the existence of the line. (This trick is supported by another property of a tube-like region, convexity. We may already have established convexity, and it may have "advised" this trick, or we can pursue convexity now.)

Now that we have one side, symmetry gives us the other. Symmetry tells us the sides have equal length; and the existence of two symmetric axes implies that the sides are parallel. We already know that they are parallel to the major symmetric axis; that is how we found them. So the relations are taken care of.

The endpoints of the two lines specify the remaining sides. We can quickly verify that one of these is a simple "single" curve, e.g. it has no concavities. The presence, shape and relative orientation of the other end is implied by the symmetry again.

Again this example has raised some interesting issues. In parrticular:

6.10. Levels of Generality

The means we used in the above example have various levels of generality. The trick we used for finding the straight line on the boundary, for example. In some respects it is quite general, depending only on a convex environment. The hypothesized line, however, really depended on our conjecture that we were dealing with a tube.

Symmetry provides an important mechanism for conveying advice. I hope to investigate its possibilities further.

As we have seen, other properties, like convexity, also provide an environment conducive to certain methods, imply or advise certain results.

We can outline now some of the "levels" and classes of advice:

1. Specific Objects

e.g. if it is a tube, then the boundary line should be over there. (I will not mention really specific objects, like 2"x4"x4" bricks.)

2. Classes of Objects

e.g. if it is a parallelipiped, then try to complete parallelograms.

3. Property Classes

e.g. if it is symmetric, than having one side, gives us the other.

4. Environmental Context

e.g. if there are a lot of shadows, expect the end to blur into the background.

We have to work to distinguish further levels and classes, and to extend our knowledge and use of elements of these classes. (Observe that in one sense these levels can be viewed as "context" levels.)

6.11 Suggestion

As I have pointed out, advice constitutes a class of suggestions. We could wait to give advice on a method for doing X until X had been otherwise suggested. However, the presence

of advice is generally considered sufficient to make the suggestion itself, i.e. to add the method to the data structure and suggestion list. This more fully obscures the distinction between "suggesting" what to do and "advising" how to do it.

As a result much of the discussion above, about part-whole relationships, generation, etc., applies rather directly to issues of suggestion. Further detailed discussion of the functioning of suggestions in SEER is really better presented with an environment of several alternative objects to draw on for examples. I will therefore only add here a few general remarks.

The first thing to note is that the Suggestion List/Priority/Monitor system provides us with a general mechanism for making suggestions. Any kind of suggestion and at any time. These suggestions can interact globally in our data base. We do not have to handle everything when it comes up, or fail back through a set of possibilities serially.

The modules described in section 3 provide more specific mechanisms for programming suggestions.

The "relevance" analysis of our knowledge structure provides a very broad approach to the question of "what should suggest what". This is reflected in our data structure links. We do not, for the present at least, require these links to be classed and labelled. The burden of using the links is borne in the organization of the hierarchy structure, and in the programming modules.

However, it behooves us to classify and analyze the types of relevance that may occur for our own understanding. This assists our analysis of concepts to effect the best utilization of our suggestion mechanisms for a particular problem.

Each conceptual node will, of course, have its "defining concepts", necessary and sufficient conditions, as relevant sub-nodes. But we need not be limited to these. Redundant or alternative information can be used. These can be organized under distinct "methods" for establishing the concept.

We should notice that suggestions are also made "downward" in the relevancy structures. We "pursue" not just necessary and sufficient conditions, but "relevant information".

Part-whole and part-part relationships are, of course, important. Again, advice of all sorts also acts to propose suggestions. Contextual observations are relevant.

And we can get into all sorts of real hand waving about causality and other relationships. We could even try to incorporate an idea of time sequence.

To summarize though.

We start with the idea that we want to allow suggestions. The results on the specific scene should interact with our general knowledge to help suggest what to do.

The suggestion list mechanism provides us with a means of doing this. And without following up every suggestion immediately. Rather they can be weighed and reinforced or contradicted.

Partial successes as well as failures are not lost in black boxes. They are available in the data base. There are specific mechanisms for "learning" from them, and generally using them to propose new conjectures.

At the heart of all this is a heuristic insight into one basic manner in which "suggestions" may originate. When a particular fact about the scene is established, general knowledge of the hierarchical relevance of concepts should permit this fact to make relevant suggestions.

The knowledge structure is analyzed to present these relevant relationships, and concrete programming mechanisms are provided to operate these suggestions.

Classes of relevant suggestions are identified.

Finally specific instances are programmed for a given problem.

7. Context

I have alluded to "context" several times above. It is a concept with so many levels, that it is rather difficult to avoid. One can talk of almost any heterarchical mechanism in contextual terms.

In this section, I will merely point to some of the higher level contextual issues, and indulge in some of my wilder hand-waving, just to let you know I am here.

The system does appear to offer possibilities for implementing context mechanisms. Context, as I say, is a broad term, and I will just suggest a few of the areas that I might explore.

Of course, the basic idea of suggestions, is essentially contextual advice, particularly in terms of part suggesting whole. As I pointed out above, we also work "downward" in context, as much as possible.

Beyond this we may consider global factors such as lighting. Have we found a lot of shadows? Well, when we are faced with the need to find the end of a tube, there may be several methods available, one of which is alert for a shadowed face blurring into the background. A "shadowed context" datum will have a relevant link to this method, suggesting it or increasing its priority.

Our past experience with the scene can aid us.

To find the end face of a tube, one method the system may

use involves finding several points along the edge between end and side, or end and background. After finding one or two of these in some general manner we may discover that the former edge is highlighted, the latter is a step. In finding further edgepoints, methods may be advised that are tailored to these types of edges.

On a higher level, if we find several bricks in the scene we can postulate a brick environment, or at least raise the priority of conjectures related to bricks.

"Psychological" concepts like "set" or "predisposition" can be simulated.

The system may expect to see something, or may be told, for example, to "look for a hammer", as opposed to "analyze the scene". We would expect these conditions to make the task easier, and perhaps to result in objects other than hammers being ignored. In fact, the system could simply be instructed only to make or follow up suggestions related to hammers.

This points up one mechanism for implementing context: affecting the set of suggestions; as they are made in EST modules, for example, or as the monitor deals with them.

The priority system can also have a contextual function.

Consider the casual peruser of a scene who suddenly spots what might be a diamond in the rough. His attention is immediately riveted. (I shall avoid other possible male chauvinist examples.) The rest of the scene is ignored as all detailed knowledge about diamonds is brought to bear on the

possibility, with highest priority.

At the highest levels some of Professor Minsky's ideas about frames could perhaps be implemented in this system. I use "frame" here for its contextual implications, e.g. a "frame" for a kitchen, which includes expectations of a refrigerator, a sink, etc. Frames could affect priorities assigned, suggestions made, or monitor functioning. More generally an entire "data base frame" could be shuffled in and out of the current "working memory" for the system.

The distinction between modules and data is already hazy and could be made more so. EST module suggestions, for example, could be read from the data base, or the EST modules themselves assembled when needed from the current frame. Even more vaguely, entire blocks of hierarchical relevance analysis could be pulled in and out as data; the specific scene would instantiate pieces of the current frame's structure interpretively when needed, to form the understanding of the scene.

Well, obviously I do not wish to get involved in a clear analysis of context mechanisms at this stage of my work. Merely to point out possibilities.

8. References

I will take advantage of the informal nature of this paper, not to present detailed references at this time.

The debt to the heterarchy concepts proposed by Minsky and Papert is obvious. The atmosphere provided by Professor Winston and the Vision Group is naturally important.

Clearly a large body of A.I. work conspired to set my thoughts in certain directions, though not necessarily in a conscious way.

Carl Hewitt's basic recognition of the two procedural facets of the implication statement stands out, of course, in recent work, along with Winston's network data structures.

I should trace up through the more recent work of Charniak, Sussman and McDermott, and back to people like Slagle.

I should mention the triggering, slot fitting aspects of the "frame" metaphor, as proposed by Minsky and pursued by Winograd, and more recently Sussman and several others.

I was not consciously working from this metaphor; in fact, I believe many of the basic features of this work were developed before I was aware of the frame model. However, one can see obvious analogies. And like most good "high level" ideas, the frame metaphor will begin to influence many projects, this one included no doubt.

One could attempt to conjure up any similarities to neurophysiological models.

Previous vision "systems" should be compared, from Roberts,

through Barrow and Popplestone, Winston et. al., Falk et. al., Brice and Fennema, etc.

Another chapter to look forward to in the thesis.