The Design and Control of Visual Routines for the Computation of Simple Geometric Properties and Relations

by

Marc H.J. Romanycia

Technical Report 87-34
October 1987

Department of Computer Science
The University of British Columbia
Vancouver, BC, Canada V6T 1W5

This report was submitted as a thesis in partial fulfillment of the requirements for the degree of Master of Science

Abstract

The present work is based on the Visual Routine theory of Shimon Ullman. This theory holds that efficient visual perception is managed by first applying spatially parallel methods to an initial input image in order to construct the basic representation-maps of features within the image. Then, this phase is followed by the application of serial methods – visual routines – which are applied to the most salient items in these and other subsequently created maps.

Recent work in the visual routine tradition is reviewed, as well as relevant psychological work on preattentive and attentive vision. An analysis is made of the problem of devising a visual routine language for computing geometric properties and relations. The most useful basic representations to compute directly from a world of 2-D geometric shapes are determined. An argument is made for the case that an experimental program is required to establish which basic operations and which methods for controlling them will lead to the efficient computation of geometric properties and relations.

A description is given of an implemented computer system which can correctly compute, in images of simple 2-D geometric shapes, the properties *vertical, horizontal, closed,* and *convex,* and the relations *inside, outside, touching, centred-in, connected, parallel,* and *being-part-of.* The visual routines which compute these, the basic operations out of which the visual routines are composed, and the important logic which controls the goal-directed application of the routines to the image are all described in detail. The entire system is embedded in a Question-and-Answer system which is capable of answering questions of an image, such as "Find all the squares inside triangles" or "Find all the vertical bars outside of closed convex shapes." By asking many such questions about various test images, the effectiveness of the visual routines and their controlling logic is demonstrated.

Contents

List of Figures

Acknowledgements

I would like to thank Prof. Alan K. Mackworth for his advice, financial support, and supervision throughout the planning and writing of this thesis. I am also very grateful to Dr. David Lowe for his many constructive comments, and to Heesoon Bai, Jordan Brooks, and Ron Rensink for reviewing earlier versions of this work. François Jalbert was invaluable in assisting me to prepare the thesis with LaTeX. Finally, I wish to thank the members of my family for their encouragement and caring.

Chapter 1

Introduction

In order to successfully interact with the world we need to correctly judge what the world contains, we need to make these judgments quickly, and we need to make these judgements with a physical mechanism which has finite storage and speed limits.

This thesis describes an attempt to apply these constraints of correctness, efficiency, and finiteness to the problem of visual perception. It has resulted in the development of a working computer system as an answer to the question of what methods we can use to *efficiently find the geometric properties of 2-D shapes and their geometric relationships in an image.*

The work presented here is based on the *Visual Routines* theory of Shimon Ullman. In this theory, efficient visual perception is managed by first applying spatially parallel methods to the initial input image in order to construct the most basic representations of features within the image, such as orientation and termination. Then, this phase is followed by applying serial methods, called visual routines, to the most salient items in these and other subsequently created maps.

This work follows the tradition of Marr (1982) in attempting to clarify the computational restrictions underlying vision. Thus, although a working system is demonstrated, it was not built as an engineering project to suit some practical needs. Also, although an effort was made to maintain a correspondence

with what is currently known about the human visual system, the system is not presented as a model of the human system. The philosophy backing this work is that much of fundamental value can be learned by creating and testing computationally inspired models.

The domain in which the system searches for properties and relations is the 2-D world of simple geometric straight-edged shapes, such as squares, triangles, and line segments. The properties and relationships which the system successfully processes are *centred-in, closed, connected, convex, horizontal, inside, outside, parallel, part-of, touching,* and *vertical.*

The central goal of the thesis is to show in detail how a small set of visual processing operations can be combined to compute a large set of visual properties and relations. A secondary goal is to show how a certain architecture and control strategy effectively manages the images, the routines, the focus of attention, the memory of what has been processed in the image, and so on, in order to enable the system to demonstrate its competence with visual properties and relations. The primary contribution of this thesis is its account of the principles and reasoning behind a working visual routine based system.

There are two other points that are noteworthy about this research.

First, in this work it is the properties and relations that are important, not the specific objects bearing them. Thus, this work is not directly concerned with object representation. Although a form of representation for the geometric objects is used to enable the recognition of objects, this representation is crude and is not central to the thesis. However, it is of interest that this representation is visual routine based. This serves to show that it is possible to define objects in terms of the visual routines used to confirm their presence. Although this approach to object representation may not be typical, it may have application wherever objects are defined in terms of their properties and the relations of their parts. This is certainly true of geometric objects like squares and triangles.

The second noteworthy point is that the visual routine system is tested by embedding it in a question-and-answer system and not in a recognition system.

An argument is made for the advantages of taking this route. The question-and-answer system allows one to conveniently test the visual routine manager. Examples of the types of question that can be correctly answered are "Find all the triangles inside squares" and "Find any three instances of a vertical bar outside a simple closed curve."

Chapter 2

Visual Routines and Attention

2.1 Visual Routines

2.1.1 Ullman

Visual routines are proposed by Shimon Ullman as a means of accounting for how abstract shape properties and spatial relations can be derived from early visual representations. The early (also called 'base') representations are uniform retinotopic maps describing the most basic properties such as depth, orientation, colour, and motion at a point. These representations are typically derived directly from the image without taking into account any high level information about what may be in the scene. Hence they are created by "bottom-up" processes. In the human visual system the base representations are thought to be computed in a spatially parallel fashion.

Suppose we wish to compute properties in the image such as which figures are inside others, whether a figure is closed, whether two points are connected, or whether some curve is longer than some other. Ullman argues that although it is possible to compute these properties and relations in a parallel fashion over all possible objects in the scene, this would in general be inefficient. Rather, we want methods which operate only on the most relevant items in the image. These methods are what he calls visual routines.

For their operation visual routines may make use of parallelism. For ex-

ample, bounded activation (colouring-in a region starting from a seed point and working out to any boundary) is a basic visual routine operation which is likely to be implemented using a form of parallel spreading activation. However, bounded activation is most likely to be initiated from only one or at most a few seed points at any one time. So we can anticipate that visual routines themselves will be primarily applied sequentially to at most a few points in an image at any one time.

To avoid indiscriminate application of visual routines to every point in the image, we anticipate making use of higher level knowledge to control the placement, selection, and sequencing of the appropriate routines. For example, while playing golf we may be interested in applying curvature-detection routines to white patches of colour.

We may note that the concept of visual routine implies both the existence of basic, elemental, non-decomposable operations and the existence of higher-level routines which may be composed from these basic operations and other high level routines. Also implicated is the existence of intermediate data structures created by applying visual routines to the base representations. Ullman calls these intermediate maps "incremental representations." They are both output from visual routines and potential input to visual routines. By assembling sets of routines and/or operations and by using incremental representations, Ullman anticipates that the visual system can compute an unbounded variety of shape-properties and spatial relations.

Visual routines are ideally suited to deducing the computational restrictions surrounding the task of visual perception. They form a natural language for dissecting visual tasks into their component tasks, and at the same time they translate naturally into standard computer terminology. This latter feature should make them easy for us to build and work with on conventional computers. It allows us to draw on our large body of programming experience to suggest possible algorithms and approaches, and allows us to make direct use of existing theoretical results when we analyze the complexity of the visual routine algorithms. Lastly, if we approach the design of the routines in a theoretical

task-first fashion (as opposed to working from the constraints of a specific vision task or a specific computer implementation), then we may feel we are truly studying the computational task of visual perception and not just writing ad hoc programs to do vision.

Ullman's Five Basic Operations

In Ullman's inaugural paper (Ullman, 1984) he suggested five basic operations out of which visual routines could be assembled.[1]

1. Shift of processing focus — This is a family of operations which move the location at which attention is directed. The reasons for the move may be voluntary and goal-related or else involuntary and feature-related. The need for such operations is clear. Because all points in an image are not equal in interest, we don't want to initiate many operations simultaneously over the image, for this could result in interference. For example, if we are determining whether a point is inside some region and we plan to colour-in the region, then we ought only to initiate colouring from the given point.

2. Indexing to a point of interest — This operation is related to the former. Here we seek a means of singling out a point so that we can then shift our attention to it. This operation determines which point is the odd-man-out in an image. Koch and Ullman (1984) describe a fast algorithm which performs such an odd-man-out indexing operation. It employs a pyramid network which for an nxn digital image computes the maximum or minimum intensity point of the image in $O(\log(n))$ steps using $4xn^2$ processors.

For the human case, indexing is probably available for all the preattentive features that psychologists have isolated.(Treisman, 1985; Julesz, 1984) These include colour, intensity, size, motion, stereo disparity, orientation, termination, and possibly curvature and closure. Because a unique point possessing one of

[1] In chapter 4 below an alternate set of basic operations is presented and a comparison is made between it and Ullman's.

these features can be quickly brought to our attention, we say these features "pop-out".

3. Bounded Activation (Colouring) — This operation is a spreading activation initiated from a given point or curve and terminating at some boundary. We could use this operation to compute the inside relation and also to separate figure from ground. There is a problem with what is admitted as a boundary because broken curves can sometimes act as solid boundaries. Ullman suggests a means of performing bounded activation which involves two maps, the map to be coloured and the terminating boundary map. The colouring could be done by a grid of cells spreading the activation to their inactive immediate neighbours except when such a neighbour is in the boundary map.

4. Boundary Tracing and Activation — This operation is similar to the preceding one, but it operates on curves rather than regions. We could use it to scan along curves checking for breaks and to check whether two points are connected. It is the operation we would use when reading a map to find whether a road or river connects two cities. Because contours are so important to vision, it is highly desirable that a basic mechanism exist for following along them.

Again, there are problems with broken contours. This operation must somehow determine when it is appropriate to treat broken lines as unbroken curves.

5. Marking — This operation is a form of memorization. It records and keeps track of points that have been indexed or to which attention has been addressed. Some such mechanism is imperative if a visual routine is not to keep returning to the same salient points and repeating identical operations. A simple form of marking would just "switch off" every point visited.

Marking makes it possible to count salient features. It is also very convenient to use while searching an image for instances of a feature, lest we keep noting the same instances. When tracing contours, it would be useful to know if we

had returned to our starting point, which would indicate that the contour forms a loop.

Marking is also invaluable in building high-level property maps, such as maps of shape descriptions or high-level property locations. For example, it allows us to construct a map of all the squares in an image or a map of where objects are inside others. Thus, as we move our focus of attention over local portions of an image and uncover non-basic properties, we can build up a map of these higher-level properties by marking their location in the original map or some other corresponding map.

Similarly, marking can be used to integrate images obtained by moving an eye or camera over a scene too large to be captured in one image. Each local image, or at least its essential components, can be copied – that is, marked – into some internal description of the entire scene.

These are the five basic operations Ullman suggested. Visual routines would be composed from these and other operations. They could also contain other visual routines, but all would eventually ground out to basic operations. However, there is more to the problem of using visual routines than just assembling basic operations, as we shall see presently.

Ullman's Control Issues

One important control problem is that of deciding which sequence of routines is appropriate for a particular input. For example, if we are trying to find a red vertical bar in an image, we could first look for vertical bars and then test for redness. Or else we could first look for red patches and then test for form and orientation. Which method turns out faster will depend on the relative frequency of vertical bars and red things. If there are fewer red things, then we would be wiser to search the red colour map first. Otherwise, we should search the vertical orientation map first.

There are also general management problems, such as how we store, execute, and test visual routines. Sometimes we will want to use skeletal guidelines and fill in specific routines as the processing unfolds. At other times we will want to save time and apply a complete sequence of routines that worked once before. And on truly novel problems we may have to ignore existing guidelines and build and test new arrangements of routines.

Visual routines, as Ullman has described them, are a means of approaching the study of intermediate-level vision – the realm of representations between the local image measurements and those of objects. Consequently, there are also issues that deal with how they interface to the low-level and high-level domains: how do they operate on the base representations; how do general visual processing goals call on them; and how do object representations link to them?

2.1.2 Jolicoeur et al.

Recently, other researchers have begun working in the visual routines framework. Jolicoeur, Ullman, and Mackay(1986) have performed psychological testing of people's ability to trace along curves. They presented to subjects images of two intertwined curves with two x's, sometimes on the same curve and sometimes on different curves. The x's were placed at the same retinal eccentricity but at varying distances apart as measured along the curve. The subjects were asked to judge whether the x's were on the same curve or not. The results were consistent with the hypothesis that our vision system possesses a basic curve-tracing operation. For x's on the same curve, the time to correctly judge increased roughly quadratically with separation distance. In these experiments, half the stimuli were presented too quickly for eye movements to be initiated, while the other half allowed ample time for eye movement. The results were similar for both cases, and, surprisingly, the latter experiments did not result in significantly longer response times. These results suggested the presence of a very rapid internal tracing process which may be used even when it is possi-

ble to track the curves with the eyes. The internal tracing scan rate averaged 40°/sec. Additional experiments were done where subjects were asked to judge whether a curve with two x's on it had a gap in it between the x's or not. The results were similar to the first experiments.

2.1.3 Koch and Ullman

Koch and Ullman (1984) have already been mentioned for their odd-man-out indexing algorithm. In that work they also propose a mechanism for inducing a shift of the processing focus. This is simply to induce a decay in the activation of the maximally active unit whereupon the indexing algorithm will automatically select a new point, which it then shifts to, induces another decay, and so on. Such a mechanism would always shift to the next highest global maximum.

Koch and Ullman report that for humans the shift of attention is not always to the next global optimum, but instead is influenced by both the proximity and the similarity to the previous point. In people the shift of attention is biased to nearby points and to points that are similar to the last point. Koch and Ullman suggest simple modifications to their scheme to enable it to behave similarly. An activated unit could simply enhance the activation of its nearby units to induce the proximity bias. For the similarity bias, a maximally active unit would activate all the active units in the feature maps that shared features with this active unit. Thus, for example, if a point which is selected by the odd-man-out mechanism is red and part of a vertical edge, then all (or maybe just the nearby) units in the red and vertical-orientation maps would have their activation increased. This would increase the chances of the next point selected being red or part of a vertical edge.

2.1.4 Pylyshyn et al.

Pylyshyn (1987, 1988) summarizes the arguments he and his colleagues have been advancing to support the existence of a kind of basic indexing operation in human vision which they call FINST indexing. A FINST is a reference or index

to a feature or feature-cluster on the retina. There may be several FINSTs simultaneously active at any one time. FINSTs have the property of remaining attached to a feature or cluster even when it moves over the retina. By this means a group of FINSTs is able to indicate the 2-D spatial relations among image features independent of the actual retinal location of the features. Also, FINSTs afford the vision system the ability to compute spatial relations among features before actually evaluating the properties of the specific features being pointed to.

The main empirical support for the FINST hypothesis is the set of visual tracking experiments Pylyshyn and his colleagues have performed. Their subjects could accurately track at least four, and at times five or six, moving points, even in the presence of distracting points. The points were moving sufficiently quickly and there were sufficiently many distracting points, so that, if the tracking were managed by a single attention mechanism serially switching between all the points, then errors in tracking would very likely have resulted.

The FINST hypothesis is most relevant to visual motion studies. For fixed images, however, the principle application of FINSTs is in simultaneously maintaining pointers to several features in an image, thereby making available all these features at once to some predicate or relation evaluation function. For example, if we are to judge the collinearity of several points, then we need to have these points readily available. The same goes for the rapid counting of objects (called 'subitizing'). Indeed, the ability of people to rapidly count up to six objects lends support to the idea that FINSTs are being used for the purpose of relation evaluation.

Pylyshyn argues that for evaluating relations FINSTs are a significant improvement over the mechanism proposed by Ullman. But this is debatable. In Ullman's scheme a single focus of attention would be serially moved to each relevant point. Each point would be marked, and then the set of marked points could be processed by the relation evaluation function. So, in the end, with Ullman's scheme the function gets the same set of simultaneously marked points.

The biggest impact the FINST hypothesis makes is in implying that significant features in an image can be marked in parallel prior to and apparently without the need of the focusing of attention. Mahoney and Ullman, to whom we now turn, propose mechanisms which in parallel build "image chunks" prior to being processed sequentially by means of attention. This appears consistent with the FINST hypothesis.

2.1.5 Mahoney and Ullman

Mahoney and Ullman (1988) have made an elegant refinement to the original visual routine proposal. They argue for the presence of an image-chunking stage in between the stage of parallel base representation building and the stage of serial application of primitive operations. This new stage would quickly assemble a variety of simple chunks or subsets of the image, which could then be treated as units by the later serial processing stages. By treating a number of spatially co-extensive image components as a unit or chunk, considerable savings can be had over standard pixel-by-pixel processing. For example, by preprocessing a curve into curve-segment chunks, the boundary tracing operation can thus trace from chunk to chunk, which, if the chunks are large, is much faster than tracing pixel-by-pixel.

Mahoney and Ullman describe three applications of chunking, one for each of the basic operations of boundary tracing, region colouring, and indexing.

For boundary tracing, they define their chunks as regions which contain a single segment of a curve. The chunks are built up iteratively into binary trees by merging adjacent regions which are compatible; that is, every curve in one meets every curve in the other at a boundary common to both regions. At the end of the chunking phase, each remaining root node represents a single curve segment. The leaf nodes are the initial elementary regions for the segment as well as an assortment of adjacent empty regions.

For region colouring, a similar algorithm is proposed. This is a form of bottom-up quad-tree method, but details are not given.

For indexing, Mahoney and Ullman propose the parallel computation of crude and approximate measures of salience. Examples of local configurations that could draw interest are curve-junctions, regions of free-space, local parallel structures (flow patterns), local proximities, local isolations, bilaterally symmetric regions, and basic shape configurations such as bars, arcs, circles, and ellipses. They stress that the computations be crude and approximate, and that they provide a quick summary description which can then be explored more thoroughly by applying attention.

Mahoney and Ullman hypothesize that the human visual system describes an image at two distinct levels. The crude level is composed of "figural chunks" which are approximate shapes with approximate measures of salience. The refined level is the more traditional spatially-focused and detailed level of description. They suggest that the detailed information is spatially indexed by the crude information. The crude level is responsible for our more nondescript, subjective, and qualitative sense of the overall arrangement of an image.

The figural chunk hypothesis is fascinating and valuable. No algorithms describing how it might be accomplished were given, although Mahoney and Ullman did suggest that basic shape configurations, such as ellipses and arcs, could be computed by the parallel and spatially uniform application of numerous template matches.

2.2 Preattentive Feature Maps

The second line of research work that this thesis draws inspiration from is the psychophysical work supporting the existence of two stages of visual processing: the preattentive and the attentive.[2] Ullman's visual routines framework is also rooted in this work.[3]

[2] cf. Julesz (1983, 1984, 1987) and Treisman (1985, 1986b).

[3] At this point we might address this question: of what value are human studies to a computational investigation? In theory one might be able to deduce a priori the type of algorithms and representations needed to solve the vision problem. Such a theory would need to specify a long list of assumptions about the type of world to be visualized. Whether or not the assumptions accurately reflect our world would be difficult to assess without subjecting the theory to

One key psychological finding is that when certain features are isolated in a sea of distractors, they can be detected in constant time regardless of the size of the image or the number of distractors. These so-called "pop-out" features include colour, direction of motion, velocity, orientation, size, length, width, intensity, binocular disparity (perceived depth), and end-point termination. Other features – such as number, inside-outside, curvature, and intersection – are thought by some to pop-out.[4] Other features definitely do not. These latter include all shape properties (circle, square, etc.), conjunctions of any two features (e.g., red and vertical)[5], parallelism, convergence, connectedness, closure, and T-juncture.

The pop-out phenomena exhibit a number of interesting complications. First, there is an asymmetry between testing for the presence of a pop-out feature and testing for its absence. In all cases, only a feature that is present pops out. Checking for the absence of a feature requires a serial scan.

Secondly, there are different pop-out effects for different values of the same

---

empirical tests. So, in effect, the person who wants to undertake a computational investigation by means of a priori deduction is faced with an experimentalist research path: inventing reasonable assumptions, testing these in a working system, revising these assumptions, retesting, and so on. The only way to avoid this is to somehow guarantee that one's assumptions are right in the first place, but it is doubtful that this is possible.

So, given that one cannot avoid system-building and experimentation in this endeavor, the question becomes whether one can get any hints as to how to proceed. Naturally, our own human system suggests itself on the grounds of its having evolved over a long time in a competitive environment. Of course, there is the risk of being side-tracked along an accidental approach to vision, one characterised by the accidental constraints introduced by our protoplasmic basis or the need to develop out of a single cell. But at these initial stages of vision science it would be wise to stick to the tried and true – that is, to mold our theories to the human model. There is still plenty of room to introduce computational considerations into a human inspired model. One needn't just copy the human system verbatim (not that we know enough yet, anyhow); one can try to justify it computationally, or else introduce computationally inspired alterations.

Ullman (1986) discusses ways in which computational studies could contribute to biological vision science.

[4]Treisman (1985) thinks that inside, outside, and curvature all pop out. Julesz (1984) says that the intersection of elongated blobs pops out, and he suggests that the counting of groups with less than six elements may pop out. Treisman (1985) says that neither does.

[5]Apparently there are exceptions to this rule. Nakayama and Silverman (1986) report that both binocular disparity and colour, as well as binocular disparity and motion, pop-out. The subjective perception of their subjects is that the pop-out occurs in one of the two depth-planes presented.

feature. For example, a vertical bar amidst bars tilted slightly off vertical does not pop out, but the reverse does. Similarly, for curvature, a straight line amidst curved one's does not pop out whereas the reverse does.(Treisman, 1985)

Another peculiarity is that in the case of the vertical versus tilted bars, this effect is largely affected by the perceived frame in which the image is found. When the target bar's orientation matches the frame's orientation, there is no pop-out.(Ibid.)

Julesz and his co-workers have discovered evidence for a variable diameter spotlight or aperture of preattentive vision (to be distinguished from the spotlight of attentive vision which we discuss in the next section). The diameter of this spotlight is inversely related to the ability to discriminate differences in preattentive features. For example, assume we have an image region in which are found two texture regions one of which has bars oriented 45° apart. And further assume that we can just barely manage to preattentively discriminate these two regions. Then, in order to achieve a similar degree of pop-out discrimination for an image with two regions where the bars are now only 10° apart, the image will have to be substantially smaller.

Julesz and his co-workers also report that it is possible for subjects to narrow the aperture of preattentive vision by "looking more closely" at a smaller portion of an image. They report, however, that it takes much longer to perform such a discrimination. Julesz suggests that this is because the preattentive system is performing some form of search, but it must be emphasized that this is not a serial search. Julesz(1984) also reports that when narrowing the aperture of preattentive vision, all the spatial-feature dimensions are identically affected. For example, when the threshold of discrimination increases for orientation, it will also necessarily be increased for spatial distance.

What do all these pop-out phenomena tell us about the computations performed by our vision system?

The only reasonable explanation for constant-time properties of pop-out phenomena is the use of a parallel-search algorithm. The purpose of rapid

search in parallel must be to enable salient points to be noticed more quickly. Anything that pops out must hence be salient in some way to subsequent vision processing.

Treisman explains that the asymmetry in presence/absence search is due to a positive signalling of the presence and location of a feature, and to no signalling of a feature's absence. This explanation is simple and satisfying.

The different effects for different values of the same feature are explained by Treisman as due to recording a feature value by measuring the difference between it and some standard. When the value happens to equal the standard, the difference is zero, and so the feature is not signalled. The standard value can be controlled by global factors such as the tilt of the image frame. This is plausible. An alternate explanation is that the standard – for example, the perceived orientation of the frame – has the effect of filtering out bars that are aligned with it. A simple way to test these two hypotheses would be to measure the degree of pop-out with target orientations progressively further from the frame orientation. If the pop-out effects increase monotonically with increasing difference from the standard, then the difference hypothesis would be supported; otherwise the filter hypothesis would be supported. The filter hypothesis is somewhat more reasonable. It would be advantageous for an organism to be able to filter out irrelevant orientations or velocities. It is difficult to see the wisdom in introducing a uniform bias on a feature; why should 60° off apparent vertical be more significant than 30° off apparent vertical?

Finally, let us consider the variable spotlight of preattentive vision reported by Julesz and his co-workers. How could decreasing the diameter of the input region increase its resolution and its computing time? Numerous mechanisms might yield this result. One possibility that explains the increased resolution and increased time delay is that the smaller preattentive spotlight must be extended more slowly and sensitively in order to discriminate the finer feature differences. This would be necessary in the presence of noise – perhaps that produced by the microsaccades – in order to integrate sufficient input over time to make the signal detectable.

2.3 The Focus of Attention

The psychological literature on the subject of visual search and attention is vast. Hurlbert and Poggio (1985, 1986) review some of the recent psychophysical and physiological work in this area from the computational perspective. Their review suggests that the role of attention in human vision is not well understood although there are some interesting theories. Some of these are recounted here.

Based on their pop-out and illusory conjunction experiments, Treisman and her co-workers propose a theory which maintains that a single focus of attention can be directed at will (by means of high-level goals) or by the presence of discontinuities in preattentive feature maps.[6] The discontinuities, however, do not directly inform the attention mechanism of their location. At best, they can only signal the fact that they exist; for example, that red is present. What the discontinuities do instead is to register their location in a special master location map. When the attention mechanism interrogates a point in this master location map, it may then retrieve the details about what features are present at this location in the image. The purpose of the spotlight of attention is to integrate information from all around the image into a single consistent percept. Our awareness of what is in the scene is a product of both this bottom-up information being integrated from around the image, and of our top-down expectations.

Julesz (1984) believes in a similar role for attention, namely, the role of enabling the combination of texture primitives ("textons"). Only within the attention spotlight is form recognition possible. Julesz goes on to describe some other interesting properties about the focus of attention. First, a shift of attention (at about 50 msec per shift) is about four times faster than a saccadic eye movement, indicating that the shift of attention operates independently of eye movements. Secondly, the shift occurs for all image sizes – for those falling entirely within the fovea as well as for those covering a large portion of the visual field. Thirdly, for size of image, the resultant spotlight of attention expands or contracts appropriately to attend to the relevant texture difference.

[6] Treisman, 1985, 1986a, 1986b; Treisman & Gelade, 1980; and Treisman & Schmidt, 1982.

The diameter of the spotlight can be as narrow as a few minutes of an arc. Finally, Julesz observes that in the absence of a pop-out region to attend to immediately, the spotlight appears to move about randomly until it finds a target of interest.

Hurlbert and Poggio report on several findings involving monkey attention. Apparently, monkey cells in the inferior parietal lobe (area 7), in V4, and in IT respond differently depending on whether or not the animal attends to a visual feature or cue. The researchers of these findings, however, seem very far from establishing the exact mechanisms responsible. There is as yet no confirmed area of the brain held to be responsible for managing visual attention. This contrasts with the preattentive feature maps, where brain regions have been found that respond preferentially to specific orientations, colours, and motions.

Chapter 3

Computing Visual Properties and Relations

In this chapter we examine in the abstract the problem of computing visual geometric properties and relations. We deduce constraints on the solution. These constraints are heeded in the implemented system which is described in the next chapter.

3.1 The 2-D Geometry World

The goal of this thesis is to find efficient methods of computing the geometric properties and relations which can be found in images. To this end it is wise to choose a visual domain rich in such properties and relations, but not so rich as to overwhelm us with the sideline issues of noise, occlusion, complex shape representation, lighting, non-step edge types, variable line widths, perspective distortion, motion, and texture. Such a world is the 2-D simple geometric shape world of circles, squares, triangles, etc., made of lines of uniform width and intensity, and appearing on a uniformly black background. The following common properties and relations are among those possible in this world:

Properties:

a) shape properties: convex, concave, open, closed, continuous, broken, and symmetric (axially, rotationally).

b) quantity and possession of specific features: terminators, straight edges, curves, inner regions, concavities, and extrema/zeros of curvature.

c) possession of specific feature values for features of quantity or degree: orientation, curvature, density, intensity, numbers of vertices, angle, length, width, area, perimeter, diameter, position of centroid, and a variety of functions composed from the above values (e.g., ratios, differences).

d) arrangement and density pattern of features: relative position of centroid, endpoints, and zeros of curvature.

Relations:

a) collinear, curvilinear, parallel, intersecting, quantity, inside, outside, beside, touching, aligned, centred, between, over, under, right-of, left-of, NNW-of, and so on.

b) $>$, $<$, and $=$ for the assorted numeric feature values. (see c) above.)

c) equality under transformation: rotation, scaling, translation, smoothing, filtering, and so on.

Of course, it is possible to define more complex properties and relations using standard mathematical operators and logical connectives. Thus we could define ratios and differences of two values, and also conjunctions of properties, negations of properties, and so on. For example, we could define a rhombus as an object with the composite property of being a simple closed curve and being composed of four equal length line segments.

3.2 The Adequacy of a Visual Routine Language

3.2.1 Descriptive Adequacy : Completeness

The basic operations from which visual routines are composed, as well as the procedural instructions which allow one to string together the basic operations

and to otherwise manage their execution, can be thought to comprise a high-level computing language. We call such a language a *visual routine language* (VRL).

If a VRL for the 2-D geometry world is descriptively adequate[1] – that is, descriptively complete[2] – it must enable us to compute all possible properties and relations (such as those from section 3.1 above) for all possible configurations of objects in that world. This is a tall order. If one were to try to prove descriptive completeness, it would be necessary to ensure that the 2-D geometry world could be defined formally and in a recursive fashion. It is doubtful that our concepts of grouping, of subpart arrangement, of relative position, etc. could stand that sort of precision. Even if they could, the proof of completeness would be an enormous undertaking.

An alternative to a formal proof of descriptive completeness is an argument by means of a sufficiently representative set of instances. This route gives up on any formal guarantee of completeness, but it does have the advantage of tractability. Of course, with this approach we still face the problem of deciding what constitutes a sufficiently representative set. Presumably such a set would contain a variety of properties and relations from each of the catgories outlined in section 3.1 above.

Lastly, a third approach to the problem of proving a VRL to be descriptively complete is the argument from Turing Machine equivalence; that is, we show that a VRL can compute anything a general purpose computer can, which, by appeal to Church's thesis, means it can compute anything computable at all. Although there is some comfort in such a proof – it means one hasn't left anything computationally crucial out – it is of no help in showing that the VRL

[1]Mackworth (1987) gives criteria defining descriptive and procedural adequacy for visual representations.

[2]There are at least two meanings of 'completeness' when applied to a formal language. The first refers to the language's ability to describe all that it was intended to describe. This is the sense in which it is used here. This type of completeness is a form of descriptive adequacy. The second meaning refers to the ability of a computation strategy to apply the language in all the fashions it was intended to be applied. This type of completeness is a a form of procedural adequacy.

allows one to *naturally* and *efficiently* compute all the properties and relations in the 2-D geometry world, which is what we really want to know.

In chapter 4 below the VRL which was invented as part of this thesis is described. No attempt is made to prove this VRL descriptively complete. In any language, descriptive adequacy is typically a tradeoff with procedural adequacy. Later we shall see, by means of numerous examples, that our invented language is capable of representing a variety of properties and relations, and so it does have a modicum of descriptive adequacy. The properties and relations that can be expressed in our VRL are *inside, outside, vertical, horizontal, centred-in, closed, connected, convex, parallel, part-of,* and *touching.* These properties and relations were chosen on the grounds that they were common and basic properties which people find important and which they compute easily.[3] Visual routines may in fact only be useful for computing such basic properties and relations. The more complex properties and relations (eg., those involving many numeric and logical operators) may best be handled by higher level processes. If this is true, then it may well be that descriptive completeness is a relatively unimportant characteristic of VRLs.

3.2.2 Procedural Adequacy : Pragmatics

Where computers and computation are concerned, there tend to be two sorts of pragmatic issues: *resource issues* – time, space, processor size, available tools, and such – and *convenience issues* – ease of use, conceptual elegance, flexibility, and such. Each of these issues defines competing criteria, and tradeoffs must

[3]Lowe (1985) discusses in depth the problem of assigning significance to perceptual grouping properties and relations. He concludes that, for the real world, viewpoint invariant properties and relations are highly significant. These include collinearity, curvilinearity, cotermination of curves, curve crossing, parallelism, convergence to a common point, equal spacing of collinear points or parallel lines, and the creation of virtual lines from the alignment of terminators. In the case of humans, because of the perceptual significance of these feature grouping properties and relations, we can suspect that many of them are computed by parallel preattentive operations rather than by the slower attention guided visual routines.

Viewpoint invariance is a good criterion to use in ranking the significance of visual properties and relations. It would be valuable to have other criteria as well.

be made amongst them. The goals and constraints placed on the system determine what tradeoffs are made. In developing the system, priority was given to resource issues because they more closely constrain any solution to the vision problem.

Resources

The resource issues are simpler to assess than the convenience issues. Complexity studies can give us a handle on resource costs, whereas convenience studies rely on the weaker subjective methods of psychology. What we want in order to assess the procedural adequacy of a VRL, however, is not just a list of algorithms to analyze and compare with regard to their resource complexity. We would like some assurance that we are on the track of the *best* algorithm. For this we need an analysis of the task itself – the task of computing with a VRL the properties and relations in the 2-D geometric world.[4] We now begin such an analysis.[5]

Image Representation

The task of computing properties and relations in images of simple 2-D geometric shapes starts with the representation of the image. Some finite representation is needed to permit computation. We have two options: a list of the symbolic descriptions of the objects in the scene, or a finitely sampled topographic representation of the scene. Since someday we want our algorithms to be applied in real environments and to make use of real sensors, and also, since we want a correlation to the human condition, then we choose topographic image representation. However, we also wish to have some of the advantages of the symbolic description. We will see a way to mix the two shortly.

[4]Mackworth (1987) discusses the difference between task or problem complexity and algorithm complexity. Task complexity is defined as the lowest possible complexity of any algorithm for the task. There can be many possible algorithms of varying complexity for any one task.

[5]As we analyze each stage in vision, we will identify those methods we deem most procedurally adequate for performing that stage. Then in our analysis of subsequent stages we will assume we are employing the solution to the earlier stage. Without this methodology the analysis would quickly explode into numerous competing paths.

It was mentioned above that we desire to avoid imaging complexities in choosing the 2-D geometric world so that we can concentrate only on the key issues in efficiently computing properties and relations. One such complexity is the sampling scheme used to go from a continuous to a digital image.

Sampling involves at least two issues: where to sample and how to sample. Regarding where to sample, a uniform tessellation makes sense since all portions of our 2-D geometry world are equally significant. Simplicity will then dictate using a regular polygon for each cell, leaving us with three tile options: squares, equilateral triangles, or hexagons.[6] The second issue we discuss is how to sample. This is the problem of how to compute the value of each cell from the continuous image. Regardless of the tessellation used, the problem with sampling is *aliasing*. At some orientations a continuous straight line will cross the cells in such a way as to generate a jagged effect. Also, there is a broadening effect on the width of the lines, which can be very substantial in cases where the lines are narrower than the cell diameter. These aliasing effects can be reduced by using finer tessellations, but then computational effort increases substantially as well. The solution opted for is to bypass the sampling stage altogether and to create pixelated lines of constant intensity directly, using techniques well known in Computer Graphics.[7] Using these symbolic lines has the advantage of both preserving connectivity and keeping lines at a uniform width of one pixel on average. Regarding the issue of tessellation type, although each has its own advantage, the square grid was chosen because of its conceptual clarity, its long tradition in computer vision, and its direct correspondence to the array data structure.

Base Representations

Now that we have an image representation, we can turn to resource issues concerning how best to make use of it. The problem of efficiently computing properties and relations dictates that parallelism be used as much as possible.

[6]Ballard & Brown (1982) and Horn (1986) discuss the relative advantages of different tessellations.

[7]For example, see Hearn & Baker (1986).

Parallelism can be used whenever simple local properties exist. The types of local information in an image are simply the infinitude of patterns of variation which can exist in a region surrounding any point. Any pattern of variation is informative to some extent. The questions we must ask are the following: Which patterns are easiest to compute? Which are most likely to occur in the image? And which correlate most heavily with "significant" states of the organism (e.g., survivally important ones)?

One of the simplest types of local computation is the single orientation gradient measurement, which is the directional derivative. This is the measure at a point, for a given orientation, of the change in measured values across the region in that orientation. To avoid instabilities in computing this measure for a point, we must weigh most heavily the contribution of points nearest this point. And for peak stability this weighting must follow a Gaussian distribution with the mean at the measuring point. (The choice of sigma influences the diameter of the region of relevance.) We can compute this weighted directional derivative by cross-correlating the image function with the directional derivative of the Gaussian.[8]

Other patterns of variation which are relatively simple to compute and yet are quite informative include the following:

a) *the rate of change of the measured variation in all directions* – This can be accurately computed by cross-correlating the image with the difference of two Gaussians. Varying the two sigmas allows one to vary the sensitivity of the measurement to lower and higher frequency gradients.

b) *an off-on-off bar pattern in a particular direction, where the bar is of a particular width* – This allows one to locally indicate the presence of a bar element of a particular width. It can be computed by cross-correlating the image with the product of a 2-D Gaussian and a 1-D Gaussian along the desired orientation. The sigma of the first Gaussian controls the length of

[8]Ballard & Brown (1982) and Horn (1986) are two prominent Computer Vision textbooks where definitions of these terms can be found.

the local bar segment and that of the second controls its width. Of course, in practice we would restrict ourselves to a finite set of orientations and widths.

c) *an endpoint or terminator pattern* – This pattern can be computed by combining the computation for b) above with a perpendicular directional derivative along the axis of the terminating bar. In this way we can detect a bar terminating at a small edge. An alternative computation, which measures relative isolation and thus can also be used to detect isolated points, is to cross-correlate the image with a 2-D Gaussian from which a constant value has been subtracted.

d) *the various edge crossing patterns* – Those with low-order vertices, which can be seen in Figure 3.1, are apt to occur often in the 2-D Geometric world.[9] One way to compute each of these properties locally is to apply the bar-pattern strategy mentioned in b) above for the appropriate orientations. This is in effect template matching. However, given all the orientations to be permuted, the number of templates to be computed goes up exponentially with the order of the vertex. We shall discuss a more efficient approach shortly.

e) *the various arcs of curvature* – Again, template matching is an obvious and quick way to compute each of these, although this could be expensive in the number of computations to be performed simultaneously in each locale, especially if we are given a large number of orientations and arc-radii of curvature.

We can be fairly confident in saying that the above are the kinds of information that ought to be calculated from the initial image, given their usefulness, basis on locality, and relative ease of computation . The only problem we will encounter is the large number of separate computations needed for complex patterns, due to the large number of possible permutations of the positions of

[9]Walters (1986) summarizes psychophysical evidence that people preferentially process such curve crossing patterns over non-crossing patterns.

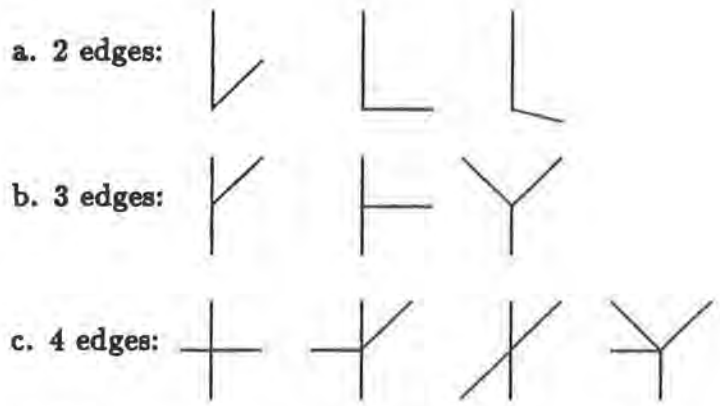


Figure 3.1: Edge-crossing patterns with vertices of low-order

their parts. If we cannot afford extensive processing at each locale, but we still want to recognize each such informative pattern, then we are forced to do some of our computing in serial. The basic idea behind intelligently serializing the process is to find abstract classes for subdividing the input patterns and then to use membership in these classes to define the pattern. In practice we must find at least two parameters that can categorize all the input patterns, find simple means to compute the values for these parameters in a first stage of processing, and then pass these parameters to a second stage where they can be reassembled to "look-up" the pattern found. For example, in order to compute the angle patterns of Figure 3.1.a, we can categorize these patterns by the minimal angle ($< 180°$) and the orientation of the bisector. If we can efficiently compute these abstract features, then we have reduced the number of computations at each locale from $O(n^2)$ to $O(n)$, where n is the number of orientation categories.

We have just seen how to quickly compute many simple but useful local properties. The next question we must ask is this: are there any non-local properties that would be useful to compute in parallel? These properties could not be guaranteed to occupy a locale of any fixed size. For example, we might want to determine the presence of shapes, locate all connected objects, locate all closed objects, or locate all collinear points directly. To do this we must introduce an extra stage of processing. We may also have to introduce a large

number of processors to handle all possible configurations.

Because a non-local property can occupy any expanse, it requires a computation insensitive to size. This requires either an enormous number of special purpose recognizers – one for each legal instance – or else a more intelligent approach based on an analysis of the non-local property into sub-properties. Since the non-local property is insensitive to size, its sub-properties must be either a) not locally computable themselves or b) locally computable but can appear any distance apart. The a) case is no solution since it only pushes the problem on to the sub-properties. In the b) case the locally computable sub-properties can signal their existence and location, and thereby, in a Hough Transform fashion, they can vote for the existence and location of their parent property. The second stage of processing must then take these signals and votes and confirm the location of the parent property. For example, to find all the collinear points in an image we can have each point vote for all orientations of lines that go through that point. If any line has had three or more votes, then we take that line, intersect it with the image, and thereby recover all the collinear points that voted for it. It is uncertain whether all non-local properties can be computed quickly by such means. Undoubtedly the complex shape properties would be more difficult to handle. However, we have made our point: with a few layers of processing even some non-local properties can be computed reasonably quickly.

Intermediate Representations

We now continue the analysis and ask ourselves what we ought to compute from these new topographic maps of basic information. Of course, we wish to apply the same criteria of usefulness and efficient resource usage.

First, we can consider applying the same local variation detecting operations to these new maps themselves. And we could of course try this repetitively. However, this would be pointless for most images from the geometry world. It would be very useful in the few cases where the alignment of features of the objects is itself significant, as in Figure 3.2. Thus we may wish to allow a second

a. terminators forming a line:

b. crossings forming a curve:

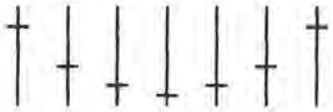


Figure 3.2: Examples of basic features that form significant shapes

application of the initial pattern computations to their initial output.

The first thing we would want from these initial topographic maps of basic features is hints about which features are related to one another to form objects, and what objects are formed thereby.[10] There are two types of such hints. First, there are hints intrinsic to the maps themselves. If two features are coincident, then they are likely to belong to the same object. If they are relatively nearby or connected by a line segment, curve, or region, then they are also somewhat likely to belong to the same object. The same holds if they have similar orientations, intensities, etc.[11] The second type of hint suggests the type of object the feature belongs to. For example, right angles suggest rectangles. With sufficient "extrinsic" hints such as these, a representation for the simple object in question could be called forth to verify the hints. The extrinsic hints index into high level knowledge, whereas the intrinsic hints can be applied without such knowledge.

Thus many weak constraints are being simultaneously applied both within maps and from maps to simple-object memory and back. We can visualize the

[10]The purpose of Mahoney and Ullman's "image chunks" (see section 2.5) may be to provide hints such as these.

[11]These intra-map hints could be quite elaborate. For example, within a single map, say of 2-edge vertices, we could learn that angles whose bisectors intersect are likely to be part of the same object.

whole process as a network settling down to a segmentation of the image into its separate connected parts or feature groups. This is a first-pass settling, one that is not likely to be revised. However, we should allow for further higher-level processing to reinterpret the initial feature groupings and thereby to force a resegmentation.

In the simple 2-D geometry world, this feature-grouping stage of segmentation is somewhat simplified. Because we disallow noise and use uniform shading of shapes, we can group all features that are connected. This leaves only the cases of overlapping shapes. If we keep overlaps to a minimum, this phase of the problem can be effectively solved by a connected region-labelling algorithm.

Once elementary segmentation has occurred, we can proceed to determine properties of, and relations among, the elementary groupings. How can this be done in a resource-efficient way?

If we review the large list of properties and relations we hope to compute, we see that many of them cannot be computed efficiently by means of parallel algorithms. There are many types of convex or closed shapes, many quantities of things, many instances of right-of, on-top-of, etc. There are too many such instances to allow one algorithm to search for each instance. Therefore, at least for these properties and relations, we need serial methods. The seriality need not be very extensive. As we saw earlier in the case of collinearity, a few stages of parallel algorithms were all that was needed. The example of collinearity should encourage us to look for similar approaches to computing other non-local functions. The hope is that at most a few stages or steps of parallel operations would suffice for computing each of them. This idea fits well with the concept of visual routines – the idea that a few basic operations applied serially can compute these same visual functions.

At this point in our progress at developing the best resource using algorithms by analysis of the task to be performed, analysis must give way to intuition and inspired experimentation. Beyond what Ullman (1984) has already said on the subject, there is no easy way to deduce which basic operations are best or how

they are to be strung together. Such an approach would be like trying to deduce what the optimal instruction set and architecture are for a digital computer. What is needed are inspired design and test experiments – the acquisition of a body of experience from which sound judgments can be made on how to proceed in doing computational vision. The system which has been built and which is described in the remaining chapters of this thesis is a small contribution to the body of experience needed.

Convenience Issues

The procedural adequacy of our VRL rests partly on its efficient use of resources and partly on its elegance, simplicity, and utility in the eyes of the experimenter. To the extent that the visual system must *itself* design and test new visual routines to suit new image situations and to learn new properties and relations, it too is apt to benefit from a convenient VRL.[12] We can determine what convenience features should go into a VRL by analyzing the task at hand, viz., the task of designing and testing of visual routines, and their placement into production.

The visual routine programmer and system developer needs to be able to perform the following tasks quickly and conveniently:

1. Create, edit, copy, save, and run routines.

2. Define subroutines; that is, treat a group of routines as though it were a single routine. He will also need to pass parameters to such routines and have values returned. These subroutines will need the option of suspending themselves prematurely if certain conditions are not being met. Subroutines should be nestable and recursively callable.

3. Conditionally apply a routine. Apply a routine repeatedly until some condition is met.

[12]For these reasons, convenience issues will become more central to visual routine studies when the problem of learning is addressed.

4. Monitor a routine during execution, stop it and examine it.

5. Create, edit, copy, save, and process images. We should include the following abilities:

   - to retouch individual pixels
   - to globally change all pixels with such arbitrary properties as colour, position, context, and all the logical combinations of these
   - to apply standard image transforms
   - to merge two or more images

No doubt, other features would add to the convenience of the language, but if the VRL designer can supply these basic capabilities, he will have done well.

3.3 The Rationale for a Question-and-Answer System

So far in this chapter we have been looking at the problem of efficiently computing visual properties and relations. We have discussed: the image world; the base and intermediate representations useful to derive for this world; the basic operations out of which a VRL can be composed; and the convenience features we would like in a VRL.

Here we briefly address the metaproblem of how best to go about building a system to test visual routines. The position argued is that a question-and-answer approach offers several advantages over the more conventional recognition approach. Others wishing to design and test visual routines may profit from considering this argument.

Visual recognition is fundamentally a process of inferring which of many possible interpretations of an image is the correct interpretation. A recognition environment can be defined in terms of an image domain and a language for describing the domain. A specific task will require choosing which descriptions

in the language are correct for a given image. Assuming that we have a language with objects, properties, and relations, then the interpretation will be based on several activities:

1. indexing from features in the image into the set of legal object descriptions,
2. confirming that the indexed objects for a given feature are consistent with the interpretation of neighbouring features,
3. testing the objects found and noting the properties they possess,
4. indexing from the object locations into a set of legal relations, and
5. confirming that the indexed relations indeed hold.

After all these have been performed the system can return a list of sentences in the language describing the objects, properties, and relations which were recognized in the image. This list will count as the interpretation of the image.

Activities 1, 3, and 4 can be very time-consuming, and they are independent of the task of testing visual routines that compute properties and relations. Activity 2 is also independent of this task, but it is necessary for confirming the presence of objects. Since we would like to avoid performing activities that are unnecessary, we would like a means to test visual routines which does not require activities 1, 3, and 4. A question-and-answer approach offers this means.

If the questions are not so unspecific as to ask "what is in the image?", but instead specify what objects, properties, and relations are being sought, then activities 1, 3, and 4 are eliminated. All that remains is searching for the specified objects, properties, and relations. The problem of indexing from all features to all possible objects is eliminated, and there is no need to try out all possible properties and relations. These are the principle reasons why we have decided to test our visual routines in a question-and-answer system as opposed to a recognition system.

Chapter 4

Design of the System

In this chapter we describe the system which has been developed and implemented. We start with an overview and then give details.

4.1 System Overview

The system is interactive. The user works at a video display station, entering commands at the keyboard and watching the results on the screen. He can easily create, edit, and manipulate the images. He can manually apply single basic operations or whole visual routines to the image. He can ask questions about the image using a simple query language. He can save intermediate results, edit a visual routine, and then restore the results to test the revised routine. In short, the development environment works well. Sample inputs, queries, and responses can be seen in Chapter 5.

The typical flow of processing is as follows:

1. A test image is input.
2. Base representations are built.
3. Questions are posed about the image.
4. The system runs from a few seconds to a few minutes, and replies with a sequence of responses, one for each instance of the object or relation

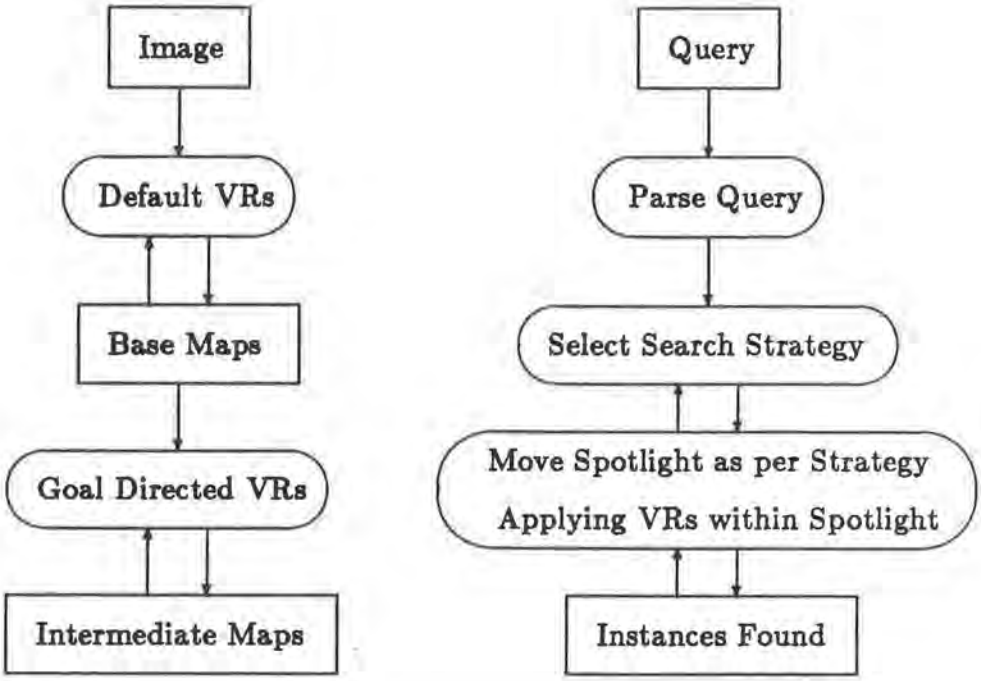


Figure 4.1: General Data Flows

found. Relation instances are displayed by simply showing the two objects so related.

The key components of the system and their interrelations are depicted in Figure 4.1.

4.2 Key Data Structures

Each map, be it an image, a base representation, or an intermediate representation, is composed of 32x32 pixels using 64 shades of grey. This is not so small as to preclude sufficient detail, and not so large as to impede response time. Maps are treated as units, and can be thought of as registers in a conventional computer, indexed by location in high speed memory. Thus, we can copy maps from register to register, apply an operator to the map inside some register,

and so on. Currently we get by with about sixty registers[1], although there is no fixed limit. Currently, the registers are used as follows:

- 12 for base representations, 8 of which are for orientations 22.5° apart
- 10 for intermediate representations
- 40 for utility

The stack can hold any number of registers, limited only by the computer's memory. The stack is used as a kind of short term memory for storing temporary search results. It enables us to backtrack to a previous decision point in a search. Local and final search solutions are stored as lists of maps.

4.3 The Visual Routine Language

4.3.1 The Basic Operations

Basic operations are compiled PASCAL procedures.[2] Each basic operation is called by issuing a single assembler-like command to an interpreter. A visual routine is simply a string of basic operations.

The following is an annotated list of the basic operations currently available. Within each category the operations are sorted by frequency of use. The frequencies are measured by counting the occurrences of each operation in the set of 36 active visual routines. Appendix A contains the detailed logic for several of these commands.

[1] If forced, we could probably get by with about 30 registers. But 60 is not an excessive number, even if we are modelling the human system. Although only 18 or so orientation maps and 10 or so other feature maps have been identified in humans, one can argue that the utility maps have not yet been discovered because, by nature, their contents are more transient and varying. Hence they would be harder to test for and to recognize using current single cell recording technology.

[2] The operations which work on image arrays are parallelizable and would benefit from implementation on a parallel machine, such as the connection machine (Hillis, 1985; Little, 1986). The parallelism is simulated in PASCAL.

Image Manipulation Operations

Let A, B, C, and R stand for $32x32$ image arrays. Let *connectivity-type* be 1 or 2, standing for 4-connectivity or 8-connectivity on a square grid, respectively. Let *mask-name* be one of the following: line, grow.a, grow.b, or end.pt. Let *orientation* be an integer from 1 to 8, standing for each of the orientations 0°, 22.5°, 45°, ..., 157.5°.

Freq	Mnemonic	Parameters
92	mov	A,B [,n]

Move registers A, $A+1$, ..., $A+n-1$ to B, $B+1$, ..., $B+n-1$. The default for n is 1.

47	bop	A, operator(+ - * / < >), B, R

Binary Operation: $\forall i,j$, $R_{ij} \leftarrow$ *(A_{ij} operator B_{ij})*. +,-,*,/ are the standard numeric operators; < is min and > is max. The frequency breakdown by operator is: +(14), -(33)

33	cbop	A, operator(+ - * / < >] [), B, R

Conditional Binary Operation: $\forall i,j$, *if* $A_{ij}=0$ *or* $B_{ij}=0$ *then* $R_{ij} \leftarrow 0$ *else* $R_{ij} \leftarrow$ *(A_{ij} operator B_{ij})*. +,-,*,/ are the standard numeric operators; < is min and > is max; (A_{ij}] B_{ij}) is A_{ij}; (A_{ij} [B_{ij}) is B_{ij}. The frequency breakdown by operator is:]&[(32).

30	set_all	A, constant

Set all values of A to the constant.

19 sp_act_l_a connectivity-type, A, B, R

Same as sp_act_lin only the process is repeated until R no longer changes. This instruction is unnecessary but convenient, as it runs much faster than the interpreted equivalent: until_nc A : sp_act_lin 1/2 A B R

15 sp_act_lin connectivity-type, A, B, R

$R \leftarrow A$. $\forall i,j$, *if* $A_{ij} = 0$ *and (any 4/8 connected neighbour of* $B_{ij} > 0$*) then* $R_{ij} \leftarrow B_{ij}$.

8 compete_3 A,B,C,R

$\forall i,j$, $R_{ij} \leftarrow 0$. $\forall i,j$, *if* $B_{ij} \geq A_{ij}$ *and* $B_{ij} \geq C_{ij}$ *then* $R_{ij} \leftarrow B_{ij}$. This is a specialty routine used for selecting one among neighbouring competitive orientations.

6 mk_convex A,R

$R \leftarrow A$. $\forall i,j$, *if* $A_{ij} = 0$ *and (*A_{ij}*'s eight neighbours are active in such a way as to suggest* A_{ij} *is a concavity), then* $A_{ij} \leftarrow 1$.

5 conv3 mask-name, A, R, orientation

Cross Correlate with a 3x3 mask indexed by the given orientation. The masks are prepared off-line and stored in memory prior to use. Appendix A describes the mask computations.

5 oddman_out A,R

$\forall i,j$, $R_{ij} \leftarrow 0$. *For one* i_0, j_0, *and* $\forall i,j$, *if* $A_{i_0 j_0} \geq A_{ij}$ *then* $R_{i_0 j_0} \leftarrow A_{i_0 j_0}$. I.e., R is empty except for a single maximum point from A.

5 spread connectivity-type, A, R

$\forall i,j$, $R_{ij} \leftarrow A_{ij} + \sum$ *(the 4 or 8 connected neighbours of* A_{ij}*) divided by 5 or 9, respectively.*

4 kop k, operator(+ - * / < >), A, R

$\forall i,j$, *if* $A_{ij} = 0$ *then* $R_{ij} \leftarrow 0$ *else* $R_{ij} \leftarrow$ *(k operator* A_{ij}*).* +,–,*,/ are the standard numeric operators; < is min and > is max. The frequency breakdown by operator is: +(2), <(1), >(1).

3 most_actv A, B, R

If $\sum A > \sum B$ *then* $R \leftarrow A$ *else* $R \leftarrow B$, i.e., make R the more active of A or B.

3 detotal connectivity-type, A, R

$R \leftarrow A$. $\forall i,j$, *if all of* A_{ij}*'s 4/8 connected neighbours are > 0, then* $R_{ij} \leftarrow 0$.

2 deunit connectivity-type, A, R

$R \leftarrow A$. $\forall i,j$, *if* R_{ij} *has no active 4/8 connected neighbour then* $R_{ij} \leftarrow 0$, i.e., zero all isolated cells.

2 unitize connectivity-type, A, R

$R \leftarrow$A. Using 4 or 8 connectivity, reduce each connected region of R to a single point.

2 centroid A,R

$\forall i,j$, $R_{ij} \leftarrow 0$. *Compute* x_0y_0 *as the centroid of the non-zero points in A.* $Rx_0y_0 \leftarrow$ *(the average intensity of the non-zero points in A).*

Control Operations

17 exit_on_z A

If $\forall i, j$, $A_{ij} = 0$ then, if we are within a loop, exit the loop and record the control variable's value in parameter SYS; otherwise, exit the visual routine immediately.

14 call visual-routine-name [,parameters]

Call the said routine while passing along the parameters.

13 exit_on_nz A

If $\exists i, j$, $A_{ij} > 0$ then do the same as for *exit_on_z* above.

10 do control-variable, start, stop, step

The typical *do loop* statement: while start $\leq$ stop perform the following expression (single line or begin/end block) and increment control-variable by step after each pass.

8 until_nc A, [,n]

Repeat the following expression (single line or begin/end block) until A stops changing, or until n passes have occurred. The default n is 50.

5 push A,B

Push registers $A, \ldots, B$ onto the stack.

5 pop A,B

Pop registers off the stack into $A, \ldots, B$.

2 if_z A, any-instruction

If $\forall i, j$, $A_{ij} = 0$ then perform the instruction.

2 if_nz A, any-instruction

If $\exists i, j$, $A_{ij} > 0$ then perform the instruction.

Utility Operations

- `draw A, draw-mode(p, v)`

Display register *A* at the terminal. If draw-mode = *p* then draw an intensity image; otherwise, display *A*'s values.

- `pause`

Stop execution and ask the user to hit any key to continue.

As we can see, the most frequently used operations include: moving registers, spreading activation, intersecting registers(cbop]&[), amalgamating registers(bop +), clearing registers, subtracting out parts from a register(bop -), and exiting a routine on some condition. We interpret the popularity of these operations to be a sign that they are intrinsically important. We must of course be cautious in drawing such a conclusion because it is not easy to separate out which operations are essential to a task and which are accidental features of a programmer's style or of early system design decisions. However, even though an author's programming style will set the frequencies of the operations he uses, that the operations he uses frequently are very useful to him must be given some weight. It would help to answer the question of which operations are truly important if other researchers were to build systems with their own brands of basic operations. We might then learn, by comparing our sets of operations, which operations are essential and which are not.[3]

A little thought can suggest why these particular operations may be important. Data has to flow from stage to stage; so moving maps is likely to be

[3]Preston (1981) surveys image processing languages that have been written for parallel machines. Some of these languages have operations similar to our basic operations, even though they were not written with the visual routine paradigm in mind. This reflects the common need to compute properties and relations in images. We might begin a program of comparing basic operations in search of the universal ones by looking at what these image processing languages have to offer.

important. Spreading activation reflects the basic need to follow paths and respect boundaries. Intersection is essential for finding common influence. Amalgamation is needed for comparison and relating to take place. Clearing maps is important for separating tasks so that a former result does not confuse a new and unrelated assignment. Removing parts is important for focusing attention on something else. And, conditional quitting is necessary to save ourselves wasted effort whenever it becomes clear that there is no point in continuing a routine.

At this point it is instructive to stop and compare our list of basic operations with Ullman's. The two operations we more or less have in common are: spreading activation and indexing to the odd-man-out.[4] Ullman makes no mention of such mundane operations as moving maps or clearing them. He also makes no mention of less mundane operations like intersecting or amalgamating maps. This may be a mistake on his part. The point of taking a computational view is to make clear the exact nature of the computations involved. It is valuable to specify every detail of the computation. If two features are to be compared, then we will need operations for bringing them together. If we are someday in position to test whether visual routines operate in the human visual system, then we will want to know exactly what basic operations we are trying to correlate with brain processes. Since the more complex operations may rely on simpler ones, it will be very valuable to know what these simpler operations might be.

Let us now see how these basic operations are actually assembled to perform interesting tasks.

4.3.2 The Visual Routines

There are certain deep logical problems which confront the designer of visual routines. Before we can get on with writing routines to compute properties and

[4]Our single form of spreading activation can be applied to curves as well as regions. In conjunction with a few other basic operations we are thus able to simulate Ullman's basic operation #3, bounded activation, and his basic operation #4, boundary tracing.

relations, we have to settle on the logical status and logical interrelationships of objects, properties, and relations. Some tough questions that arise are:

- What is an object independent of its form and properties? Is the form of an object a property?
- Are objects single connected things, or can they also be groups of things?
- Properties and relations apply to objects. But, can properties and relations also apply to properties and relations? For instance, a property of the relation *greater than* is asymmetry, and properties of colours are their brightness and hue.
- Do we admit relations of arity greater than two, or can we get away with defining these in terms of properties and binary relations? For instance, we could define COLLINEAR(dot1,dot2,dot3) as COLLINEAR(GROUP-OF(dot1,dot2,dot3)); and BETWEEN(A,B,C) as WITHIN(A,GROUP-OF(B,C)).

The solution to these problems is motivated by the desire to avoid effort and complexity. Accordingly, the response to each of the above problems is as follows:

- Objects are forms with properties; hence, the form of an object is not a property. To confirm the presence of a form, you start with a location or region, and try to construct a shape of that form within or near that starting point. For a property, there is no question of going beyond the input since the location of what we must look at is given.
- Currently, there is no allowance for the definition of group objects. For instance, there is no means to handle a row of dots or a square made of little triangles.
- Properties and relations only apply to objects.

- Only binary relations are admitted.

Within these limits, however, there is much room left for expressing ourselves.

We will now examine all the visual routines that were written for computing properties and relations and for recognizing objects. There are other visual routines that were written but do not appear here. Some of these are used internally by the search logic. For instance, a routine was written for computing the parts of an object which do not touch any other object. This is useful for deciding what to trim from index maps after an object has been found. Others are used for generating the base maps. All of these additional routines can be found in Appendix A.

Each routine is called with a certain parameter passing convention. For relations, the convention is to pass the register numbers of the registers which contain the original image, of two registers containing the forms or structures[5] to be tested, and of a register containing the union of both these structures. The convention is that, if the structures fail the relation test, then the register whose number is passed as parameter #6 is to be set to zero; otherwise, it is set to the union register.

The reader may disagree with an interpretation that is given here to a particular property, relation, or structure. There are many definitions of 'inside', 'outside', 'touching', and so on. In order to keep things manageable, one popular definition was chosen to represent each term and a clue to the specific meaning is given in parentheses wherever necessary. One difficult problem for Visual Routine Science will be accounting for the many subtle and context dependent shades of meaning that visual properties and relations can have.

[5] I use the term 'structure' henceforth as it has fewer alternate meanings.

Relations

Each partial relation routine below is to be completed by placing the following command skeleton around it:

```
{%1 = img reg        {Input
{%2 = struct1 reg    {  ''
{%3 = struct2 reg    {  ''
{%4 = union reg      {  ''
{%5 = match reg      {Output
set_all %5 0         {clear answer in case of exit-fail
   . . .
[operations unique to routine]
   . . .
exit_if_z/nz         {exit if relation(struct1,struct2) is not true.
mov %4 %5           {o.w. return the union as the answer
```

1. **Inside** (within the convex hull of)

```
mov  %3 29           {store struct2 in 29
until_nc 29
  mk_convex 29 29    {make struct2 into a convex blob
cbop 29 [ %2 28      {intersect with struct1
exit_on_z 28         {if nothing in common, then not inside
bop %2 - 28 27       {o.w., compare the intersctn with the original
exit_on_nz 27        {if any dif, then st1 is not inside convex st2
```

2. **Outside** (not entirely Inside)

```
mov  %3 29           {store struct2 in 29
until_nc 29
  mk_convex 29 29    {make struct2 into a convex blob
cbop 29 [ %2 28      {take intersection
exit_on_nz 28        {if something in common, then st1 is not outside
                     {the scope of st2
```

3. **Touching** (crossing or immediately adjacent)

```
mov %2 20
spread 2 20 20       {create immediate neighbourhood of struct1
cbop 20 ] %3 21      {and see if it intersects struct2
exit_on_z 21         {if not, then not touching
```

4. **Centred In** (centroids intersect precisely)

```
centroid %2 20
centroid %3 21
cbop 20 [ 21 20    {intersect centroids
exit_on_z 20       {if they don't overlap, then not centred
(Note, we could loosen our standards on what stands as being
 centred by smoothing either centroid before the intersection)
```

5. **Connected**

```
mov %2 20
sp_act_1_a FG_PARM 20 %1  {generate the entire structure connected
                          {to pt-reg
mov %3 21
cbop 20 [ 21 22      {intersect it with struct2
exit_on_z 22         {if nothing in intersection, then not connected
```

6. **Parallel**

```
( Note, we might first want to ensure that both str1 and str2 are
{ bars; but here, for efficiency, we remove these tests.
{ Hence we must be sure to only query :  "... bar parallel to bar"
{-----------------------
{mov %2 20           {test str1 = bar
{call bar 20 20 21 22 23
{exit_on_z 23
{mov %3 20           {test str2 = bar
{call bar 20 20 21 22 23
{exit_on_z 23
{-----------------------
mov %2 20
do y 1 8             {find str2's orientation
  begin
    cbop 20 ] y 21 {intersect with orientation map y
                   {and set all values to those in strc1
    bop 20 - 21 21 {remove all y oriented edges from strc1
    exit_on_z 21   {exit loop when orientation found (sets SYS = y)
  end
{-----------------------
cbop %3 ] SYS 22     {now verify that SYS is str3's orientation too
bop %3 - 22 22
exit_on_nz 22
```

7. **Part of** (a proper subset of)

```
cbop %2 ] %3 20      {find portion of part shared by whole
bop %2 - 20 20
exit_on_nz 20        {exit if part contains extras not in whole
cbop %2 [ %3 20      {find portion of whole shared by part
bop %3 - 20 20
exit_on_z 20         {exit if the whole is not greater than the part
```

Properties

The parameters passed to properties are somewhat fewer. We only pass the image, the image boundary, and the structure. The convention is that, if the structure does not have this property, then the register whose number is passed as parameter #4 is to be set to zero; otherwise, it is set to the structure register.

Each partial property routine below is to be completed by placing the following command skeleton around it:

```
{%1 = img reg       {Input
{%2 = boundary reg  {  ''
{%3 = struct reg    {  ''
{%4 = match reg     {Output
set_all %4 0        {clear answer in case of exit-fail
   . . .
[operations unique to routine]
   . . .
exit_if_z/nz        {exit if property(struct) is not true.
mov %3 %4           {return the object in %4 to indicate success
```

1. **Horizontal** (the dominant orientation)

```
mov %3 20
cbop 20 [ 5 21          {intersect with horiz orientation map (=5)
mov 21 22
do y 1 8
  begin
    cbop 20 [ y 23      {intersect with orientation map y
    most_actv 23 22 22  {keep most active orientation map in 22
  end
bop 22 - 21 22
exit_on_nz 22           {if horiz component(21) ~= most
                        { active orientation(22), then exit
```

2. **Vertical**

```
(Same as Horizontal only use ''cbop 20 [ 1 21'' for line 2 in order
to intersect the structure with the vertical orientation map (=1).)
```

3. **Closed** (is a simple closed curve)

```
mov %2 20
set_all 21 1
bop 21 - %3 21      {create inverse image
sp_act_1_a BG_PARM 20 21  {sp_act out from bndry within the inv.img
bop 21 - 20 22      {remove activated part, leaves inner parts
```

```
exit_on_z 22          {if no inner regions, then not closed
unitize BG_PARM 22 23     {reduce each inner region to a single pt.
oddman_out 23 24      {find one such point
bop 23 - 24 23        {remove it
exit_on_nz 23         {if more than one inner region exist, then
                      {the struct is not a simple closed curve
{---- now test that no protrusions exist
mov 22 25             {need inner regions map computed above
sp_act_lin FG_PARM 25 %2  {spread activate out twice to include
sp_act_lin FG_PARM 25 %2  { the object portions adjacent to the
                      { inner regions and any small protrusions
bop %3 - 25 26        {remove all this from the object
exit_on_nz 26         {exit if (large) protrusions exist
```

4. Convex (no concavities & needn't be closed)

```
mov %3 20
until_nc 20
  mk_convex 20 20   {create convex hull of object
detotal 1 20 21     {remove interior of hull
bop 21 - %3 22      {remove object, leaves edge of hull in concavity
exit_on_nz 22       {exit if such an edge exists
```

Structural Forms

The parameters passed to structure routines are rather different from those passed to properties. They include: the image; the image boundary; the spotlight of attention, which is usually a single point from an index map that the type of structure is expecting (e.g., squares expect corners in the spotlight); and the index register itself in the event that the routine sees fit to remove some points that needn't be examined any further. For output, each routine always fills the structure register with whatever relevant structure it has found under the spotlight. The match register is filled with either nothing, if the sought structure was not found, or with the structure, if it was found.

Each partial structure routine below is to be completed by placing the following command skeleton in front of it:

```
{FIND any Structure attached to the pt in pt-reg
{%1 = img reg       {Input
{%2 = boundary reg  {  ''
{%3 = pt  reg       {  ''
{%4 = index reg     {Input/Output
{%5 = struct reg    {Output
```

```
{%6 = match reg     {  ''
set_all %6 0        {clear in case of exit-fail
. . .
[operations unique to routine]
```

Base Map Features

1. Terminator

2. Crossing

3. Corner

4. Concavity Vertex

```
{For all four of the above:
{the indexed base map is the one corresponding to the feature
{So, for terminators, replace MAPID below with 9; for crossings,
{replace it with 10, and so on.
sp_act_1_a 2 %3 MAPID {recover entire feature attached to spotlight
mov %3 %5             {return it as the solution
mov %5 %6
```

Background Regions

5. Inner Region

```
{the indexed base map is concavity vertex
mov %3 20
sp_act_1_a FG_PARM 20 %1 {sp_act in img (backgrnd: hence conn-4)
mov 20 %5           {return the region found
cbop %2 ] 20 21     {intersect with boundary
exit_on_nz 21       {if intersects bndry, then not an inner rgn
mov 20 %6
```

6. Concavity

```
{the indexed base map is concavity vertex
mov %3 20
spread BG_PARM 20 21      {widen index point
cbop 21 [ 0 21      {recover portion in image near concavity vertex
sp_act_1_a 2 21 0  {recover entire parent object nearby
{------  First test that region is not an inrgn of parent
push 20 21
push 50 55          {save regs
set_all 50 1
bop 50 - 21 50      {create inverted image of parent object
mov %1 50
mov %2 51
mov %3 52
```

```
mov %3 53           {set up parms
call inrgn 50 51 52 53 54 55
mov 55 25           {save reply
pop 55 50
pop 21 20           {restore regs
mov 25 %5           {prepare to return whatever was found
exit_on_nz 25       {if the region is a closed inner region, then
                    {it is not a concavity, so exit
{--------------------------
{-- now recover concave region
mov 21 22
until_nc 22
  mk_convex 22 22   {find entire convex region about the object
bop 22 - 21 23      {remove object (leaves concave regions)
mov %3 %5
sp_act_1_a FG_PARM %5 23  {recover the concav region attached to
mov %5 %6           { the index point; return it as the answer
```

Foreground Structures

7. Square (vertical & isolated)

```
{the indexed base maps are corners, horizontal, and vertical
mov %3 20           {isolate the
sp_act_1_a 1 20 %1  {struct attached to pt_reg; 4-c bec of vert sqr
mov 20 %5           {prepare to return the isolated
                    {struct as the struct found
{-------------------------
{ The basic strategy here is to start from the corner and
{ trace both vertically and horizontally until we reach the
{ end.  Then we trace horizontally and vertically from those
{ end points until we reach a new set of endpoints.
{ If those endpoints are the same, then we have found a square!
{-------------------------
set_all 21 0        {interim path trace stored here: init to 0
mov %3 29           {set up sp_act_lin start point for vert pass
mov %3 26           {save start point
mov %3 25           {set up sp_act_lin start point for horiz pass
mov %3 22           {save start point
{--- First vertically/horizontally
until_nc 29
  begin
    mov 28 27          {record last frame
    mov 29 28
    sp_act_lin 1 29 1  {sprd act out vert; trick: use 4-c
    mov 24 23          {record last frame
    mov 25 24
    sp_act_lin 1 25 5  {sprd act out horiz; trick: use 4-c
  end
bop 29 + 21 21
bop 29 - 27 29      {compute last point(s) activated
bop 26 - 29 26
exit_on_z 26        {if last=start, then exit-fail
bop 25 + 21 21
bop 25 - 23 25      {Ibid for vert path
bop 22 - 25 22
exit_on_z 22        {if last=start, then exit-fail
{--- Next, horizontally/vertically
```

```
mov 29 26          {save start points
mov 25 22
until_nc 29
  begin
    mov 28 27          {record last frame
    mov 29 28
    sp_act_lin 1 29 5  {sprd act out horiz; trick: use 4-c
    mov 24 23          {record last frame
    mov 25 24
    sp_act_lin 1 25 1  {sprd act out vert; trick: use 4-c
  end
bop 29 + 21 21
bop 29 - 27 29      {compute last point(s) activated
bop 26 - 29 26
exit_on_z 26        {if last=start, then exit-fail
bop 25 + 21 21
bop 25 - 23 25      {Ibid for vert path
bop 22 - 25 22
exit_on_z 22        {if last=start, then exit-fail
{--- finally test that two final termination points are the same
cbop 29 ] 25 28
exit_on_z 28
mov 21 %5
mov 21 %6           {return the square!
```

8. Triangle (isolated)

```
{the indexed base map is corners
mov %3 20
sp_act_1_a 2 20 %1   {recover the full isolated object in 20
mov 20 %5            {prepare to return the isolated
                     {struct as the struct found
{--------------------------
{ First check that only three terminator regions exist
cbop 20 ] 11 21       {intersect with corner map
unitize 2 21 21       {8-conn, make each term rgn a single point
oddman_out 21 22      {pick one terminator
bop 21 - 22 21        {remove it
oddman_out 21 22      {pick another
bop 21 - 22 21        {remove it
exit_on_z 21          {if there were only 2 terminators, then exit
oddman_out 21 22      {pick another
bop 21 - 22 21        {remove it
exit_on_nz 21         {if there were > 3 terminators, then exit
{--------------------------
mov 20 27
call closed %1 %2 27 28 {verify that the object is simple closed.
                     {Closed uses temp regs 20-26; so using
                     {27 28 is a dangerous but efficient hack;
                     {we should really use a push/pop sequence.
exit_on_z 28         {if nothing returned, then obj is not closed
{--------------------------
mov 28 %6            {return the triangle
```

9. Line Segment (Bar)

```
{the indexed base map is the input image
mov %3 %5                {prepare to return index pt as object found
mov %3 20
do y 1 8                 {for each orientation
  begin
    cbop 20 ] y 21       {intersect with orientation map y
    sp_act_lin 2 21 y    {sprd act :8-conn, in this orientation
    oddman_out 21 22     {find one point
    bop 21 - 22 21       {remove it
    exit_on_nz 21        {exit loop if some remain
                         { sets SYS parm to y
  end
exit_on_z 21             {if none found active, then only a dot here
sp_act_1_a 2 21 SYS      {recover object in this orientation
cbop %1 ] 21 %5          {renormalize to image intensity
mov %5 %6                {return the bar
```

10. Dot

```
{the indexed base map is the input image
mov %3 20
sp_act_1_a 2 20 %1  {recover attached object
mov 20 %5           {return the struct found
cbop %3 [ %1 21     {set index point to image intensity
bop 20 - 21 22      {remove it from the struct found
exit_on_nz 22       {if any remain, then the struct was not a dot
mov 20 %6           {return the dot
```

11. Isolated Arbitrary Object (IAO)

```
{the indexed base map is the input image
mov %3 29
sp_act_1_a 2 29 %1  {recover the object attached to the index pt
mov 29 %5           {return it as the found struct
mov 29 %6           {return it as the IOA
```

The reader may wonder why we bother with so many structure types when the goal of this thesis is to compute properties and relations and is not to do object recognition. The point of being able to recognize several objects is to enable us to test our control logic in the presence of objects having common features, such as edges and corners. Hence, we have two types of closed object – square and triangle; two types of background region – concavity and inner region; and two types of simple 1-D shape – bar and dot. The feature objects are available for free, and we should be able to recognize them, given that we can index to them.

It is not a failing that we artificially define our squares and triangles to be isolated and the squares to be not tilted. It saves us from having to spend undue effort rotating the image or removing lines in the hope of finding the object. In the opinion of the author, objects should be remembered as a series of prototypical views, with recognition managed by template matching, and visual routines play no central role in this. The reasons it was decided to do recognition by means of visual routines are the following: the routines were readily available and using them would save the additional programming effort of a recognition system; it was desirable to see if visual routines were powerful enough to do it; and this approach might clarify the relationship between certain types of defined object and the routines used to confirm the presence of the defining ingredients. This last reason is particularly applicable in the case of geometric objects. For example, a parallelogram is hard to define prototypically, but is relatively easy to define in terms of its number of corners, its being a simple closed curve, and its having two instances of parallelism. Likewise, there are many triangles, but all have three corners and are simple closed curves. So, in these cases the object representation is likely to have "hooks" into the visual routine calling mechanism. Thus, by doing visual routine based object recognition, one can explore the interface between object representations and visual routines.

4.4 The Question-and-Answer System

In Chapter 3 we saw arguments for why we would do well to apply our visual routines to *interrogating* images – for specific objects' properties and relations – rather than to *recognizing* whatever the image contains. Here we briefly sketch the design for a query language which is to facilitate such interrogation. We provide motivation for the design by first looking at the interrogation task the language is to solve.

First, let us look at the sort of question we ask when we search for an object on which to focus. Here are some examples.

A.1 What objects are in the scene?

A.2 What is the distribution pattern of objects in the scene?

A.3 How many objects are in the scene?

A.4 Are there any objects in the scene with some property or some combination of properties? How many of these are there, and what is their distribution?

A.5 What are the most salient, striking, peculiar, or interesting objects in the scene?

A.6 Does a specific relation hold between any two objects?

A.7 Do some objects in the scene share properties?

A.8 What objects, properties, and relations are present in some subportion of the scene?

Then, let us look at the sort of question we ask when we want to learn more about an object or objects which is or are already brought to our attention.

B.1 What are all the object's properties?

B.2 How many objects are there in this group?

B.3 Which properties of this object are striking, peculiar, or interesting?

B.4 Does the object have some specific property or some combination of properties?

B.5 Does some specific relation hold between any two objects in this group?

B.6 Are there some properties which the objects have in common?

B.7 Does some logical combination of properties and relations hold for the members of this group of objects?

This list is no doubt incomplete, but it will allow us to infer the general features we want in our language.

We notice the following about our questions:

- Question A.1 is the general recognition request all over again. Hence, we will prefer to disqualify such questions from being in our language, and request that at least some restrictive search information be provided in the question, be it a specification of a structure, of a property, or of a relation. We disqualify questions with the character of "what is this?" and only admit questions with the character of "what meets such and such conditions?" Hence, we disqualify question A.2 as well. The most general question we will allow ourselves to ask for our 2-D domain is: "what arbitrary isolated objects are there in the scene?" There is no object recognition required to answer this; all we need is knowledge of what it means to be isolated.

- Questions A.3, A.5, and B.3 are bipartite questions. For question A.3, we must first find all the objects and then count them. For question A.5, we must at least cursorily consider all the objects, sort them by some standard, and then return those highest on the list. Likewise, for question B.3, we must find all the applicable properties, and then sort and evaluate them. We should be on the look out for such questions and, for clarity's sake, decompose them into their constituent queries.

- All the A Questions can be phrased in the form: "find me some X with the following properties and/or relations." In contrast, in all the B questions, the objects to be considered are provided for them. These questions can be phrased in the form: "give me information about X_0", where X_0 is known. Our language will therefore have two basic question forms – the FIND form and the GIVE information form. The FIND form is concerned with objects, and the GIVE form, with properties and relations (Ps&Rs).

  To manage the distinction between known and unknown objects, our language will need constants and variables. In order to disambiguate the

variable scopes, we will also need parentheses and/or scope conventions. In addition, we will need to type the constants and variables. The variables for the FIND requests range over image objects. Those for GIVE requests range over Ps&Rs. A FIND request will return a list of the objects, the ordered pairs of objects, the ordered triples of objects, etc., depending on the number of quantifiers used. Similarly, a GIVE request returns a list of Ps&Rs. An Object-constant can be a single object or a list of objects. A P&R-constant can be a single property or relation, or it can be a list of these, so long as the members are all of the same arity. The idea of admitting lists is to enable us to ask complicated questions by assembling a sequence of simpler ones. The results of one query can thereby be made into the constant terms of another.

- Questions A.3, A.4, and B.2 seek quantitative information. So we will need a counting function which we can apply to a list of objects, properties, or relations.

- Questions A.5 and B.3 ask us to rank the values of properties. So we will need a sorting function which we can apply to a list of property values for objects, properties, or relations. Indeed, after introducing counting and sorting, we might as well admit any list-processing function.

- Question B.7 requires the expression of a logical combination of properties. Hence, we must have a convenient and complete set of logical operators.

There are no doubt other useful features we should include in a good Question-and-Answer language, but those above represent a good start.[6] Regretfully, little of the language which is sketched here was implemented. The only command that was partially implemented was the FIND command. Only as much of it was implemented as was necessary to achieve the goals of testing VRs

[6]We might acquire good features from the languages developed by researchers on Picture Grammars (see Nake and Rosenfeld (1972) and Kaneff (1969) for a representative sample). For instance, Stanton (1972) describes a language, RAMOS, which was built to recover structures from images. Among other commands it possesses a FIND command similar to the one described here. It also includes descriptions for points, lines, terminators, triangles, and squares.

and their control logic. Completing the language and providing visual routines which can answer every query within the language would bring us a long way toward a complete set of visual routines and control strategies. How do we visually count objects, sort objects, process lists or groups of objects, and manage logical combinations of properties? If we are to apply visual routines to these tasks, and if testing is to be thorough, then we will need a language in which we can express all the myriad variations of the task.

The precise grammar of the query language currently developed is as follows:

```
                query : find-object-query | find-object-pair-query
    find-object-query : FIND quantifier object
find-object-pair-query : FIND quantifier object object : relation
            quantifier : ALL | ANY-n
                     n : 1,2,3,...,100
                object : structure[/property[/property[/property]]]
             structure : TERM | CROSS | CORNER | INVTX | INRGN |
                         SQR | TRI | BAR | DOT | X
              property : HORIZ | VERT | CLOSED | CONVEX
              relation : INSIDE | OUTSIDE | TOUCH | CENTRED |
                         CONNECT | PARLL | PARTOF
```

So, for example, to ask the question, 'find all vertical line segments', we would actually type at the keyboard, 'FIND ALL BAR/VERT'. Likewise, 'find any three closed and convex inner regions inside any square' would be entered as 'FIND ANY-3 INRGN/CLOSED/CONVEX SQR : INSIDE'[7].

The logic of the Question-and-Answer system is simple but adequate:

1. Parse the query.
2. Pass the parsed query to the FIND routine.
3. Display all the successful finds returned from the FIND routine, and save these finds in memory for possible user examination.

[7] Any combination of lower and upper case is accepted.

4.5 Control Logic

In this section we describe the control executive which manages the user's queries and manages where and when visual routines are applied.

In Ullman's visual routines framework there are two basic operators which together control the spotlight of attention. These are the "index to a point of interest" and the "shift processing focus" operators. The former does the important work of deciding where to focus attention. Koch and Ullman (1984) discuss three criteria for deciding where to focus attention: the criterion of maximum peculiarity, or "oddest man out", in some feature map; the criterion of proximity to the previously indexed point; and the criterion of similarity in feature values to the previously indexed point. These are valuable but are incomplete as an account of how to manage the search which is required to intelligently answer the queries in our query language. From experimental experience the author has found that the following problems arise. These problems are sufficiently universal that any method for controlling attention would likely face them.

1. The problem of deciding which search strategy to use when searching for an object – We could search randomly; or search with guidance from heuristics, such as the proximity and similarity heuristics mentioned by Ullman; or we could search by following a specific search pattern, e.g., a raster scan, or a pattern that spirals out from the centre. Figure 4.2 illustrates situations where different search strategies would be advantageous. Raster scan search suits Figure 4.2.a because we want to find *all* instances of an object in an image of evenly spaced shapes. Proximity search suits Figure 4.2.b because right angles occur back-to-back in images of crossing lines. And, random search suits Figure 4.2.c because we only have to find one instance of an object in an image where they are plentiful.

The problem of deciding on a search strategy may be faced repeatedly during a search, not just upon initiating the search. Portions of an image may benefit from different strategies. Also, we may want to change strategy mid search if

a. Raster scan search is suited to the following image for query 'Find all squares':

b. Proximity search is suited to the following image for query 'Find any two right angles':

c. Random search is suited to the following image for query 'Find any one square':

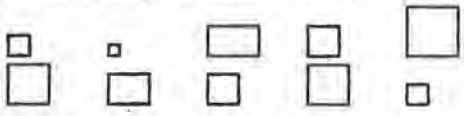


Figure 4.2: Situations warranting different search strategies

we find we are making no progress.

2. The problem of spotlight diameter – The reason why we need to focus attention at all is that we cannot afford to apply visual routines uniformly across an image. Yet it is clear that we want to apply some basic operations over an area, and not just at a point. So the problem becomes, what is the optimum size area or spotlight within which we can apply a routine? Also, does this size depend on the problem to be solved?

3. The problem of which feature map we should be indexing within – For example, in the query of Figure 4.3 we would be best off searching the vertical bar map rather than the terminator map. Ullman (1984) also mentions this problem.

4. The problem of different search formats when trying to find a single object as compared to finding a pair of related objects – This problem arises because relations and objects are intrinsically different sorts of thing. Finding an instance of a relation requires finding objects first; whereas, in the 2-D

For the query 'Find all vertical bars' we are better off searching the vertical map over the terminator map.

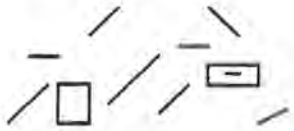


Figure 4.3: Example illustrating the value in choosing the right map to search

a For the query 'Find all triangles outside squares' we may want to find triangles and squares first before computing the outside relation.
b. For the query 'Find all right angle triangles inside vertically oriented squares' we may want to compute the inside relation first.

Figure 4.4: The advantages of different task schedules

geometry world at least, objects have existence independent of any relations they enter into.

5. *The problem of task scheduling* – For example, when trying to find a relation, should we first look for the objects, verify their properties, and then see if they are related? Or, should we instead look for general objects, confirm the relation between or among them, and then confirm that these general objects indeed have all the desired properties? In Figure 4.4 we see examples of when it is advantageous to use each strategy. The scheduling problem also arises when trying to find an object with several properties. In what order should we test for the properties? The task scheduling problem is addressed by Ullman with his suggestion of "skeletal guidelines" (Ullman, 1984).

6. *The problem of what to remove from a map in which we are indexing, after we have found or failed to find something at the indexed point* – Provided that we are not working with a reflexive relation, we can certainly remove the

If we are searching for isolated triangles and we find p1 is not attached to one, then we may legitimately remove p2, p3, and p4 from the index map we are searching.

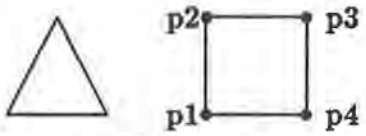


Figure 4.5: The removal of unneeded points from an index map

indexed point; but if we are clever, we can also remove some nearby points that are no longer worth indexing to. For example, in Figure 4.5, when we find there is no isolated[8] triangle attached to point p1, we can remove points p2, p3, and p4 as well.

7. The problem of duplicate examinations – No matter how carefully one trims the search maps after each search step, situations can still arise where the search sequence brings you back to a previously examined object. This is because one cannot legitimately trim all points from each object examined, and also because objects are often larger than the spotlight radius, and so one can return to them from a different route. In the case of relations, it is likely that, having considered R(obj1,obj2), one will then encounter R(obj2,obj1). If R is symmetric, this is a waste of time.

8. The problems of what to do when a search pass fails – Should we just quit; try a new search strategy; use a new resolution scale; rotate the image; lengthen lines; or otherwise modify the image?

Our response to the above eight control problems is as follows.

[8]I define an object to be isolated when it is in no way embedded within any other shape and has no other appendages than those defining it.

Problem 1: Selecting a Search Strategy

It is clear that selecting a search strategy must be done early, and then be open to reconsideration after some searching has been done. Accordingly, the control logic does exactly this. There are two loops. Within the first loop we select a strategy, and within the second one we carry out the last strategy selected.

```
1. While we have not yet succeeded & the situation is hopeful
2.   While the search is proceeding well & we have not yet succeeded
3.     Select search strategy
4.     Move spotlight as per strategy
5.     Search for instance of item, starting at spotlight location
       (This may entail moving the spotlight as well.)
6. Done.
```

In the current implementation we have not addressed the problem of which strategy to select. We have allowed for the addition of new strategies, but the only one we use is the simple odd-man-out strategy without heuristics. It is adequate for the size of image we handle. For large images in which "needle in a haystack" problems can be presented, the patterned searches would be useful. As for the heuristic approaches, it is not clear exactly when they would be useful, although, as we saw in Figure 4.2, there are particular images and questions for which they do help. For the entire class of 2-D geometry world images we cannot think of any obviously useful heuristic.

Problem 2: Choice of Spotlight Diameter

The solution to the problem of choosing a spotlight diameter will likely emerge only after we have acquired much experience working with visual routines. Accordingly, our interim solution to the problem is to allow for the freedom to experiment by leaving the control of the spotlight diameter up to the individual routines. Sometimes a routine will apply a basic operation over the entire image. For example, this is done as a preprocessing stage prior to computing the INSIDE relation. Most other times the routines will expand the spotlight out to cover the relevant region. For an object this relevant region is its convex hull.

To facilitate this freedom of control, what the system passes to visual routines is an initial spotlight which is the point or region it is currently attending to and which the routine can modify as it sees fit. For visual routines of structures, such as *square* and *bar*, this initial spotlight is a single point, usually the last point the system indexed to. The routine then expands the spotlight to cover the object it hopes to find. For visual routines that compute properties, such as *closed* and *convex*, this initial spotlight is larger; it covers the object to be evaluated for the presence of this property. For visual routines that compute relations, such as *inside* and *connected*, this spotlight covers the subimage (of the input image) which contains only the two objects to be tested for the relation.

Problem 3: Which Base Map Should We Index Within

For this problem our guiding principles ought to be to use those maps most likely to contain evidence of the objects we are seeking, and to use those maps least cluttered by the presence of objects which we do not seek. Accordingly, we solve problem 3 in the following manner. Upon receiving a query the system takes note of all the objects and properties mentioned. For each of these it looks up in a table the base maps which indicate the presence of these objects and properties. It thus constructs a list of base maps which are pertinent to answering the query. Its only concern then is to choose a map which minimizes search. Given that it does not know how many distractor objects may be in the image, one good piece of advice is to guarantee some lowest upper bound on the required search by choosing a map with lowest overall activity, and this is just what the system does.

This heuristic rule – use the base map which has least activity – works well, although it does mislead us whenever the relative activities between maps are not directly comparable. For example, a long vertical bar will activate only two terminator map points but many vertical map points. If we are looking for a vertical bar amidst a number of horizontal ones, our heuristic might have us

If we are searching for a horizontal bar, then, because there are fewer points active in the terminator map than the horizontal map, we will search inefficiently.

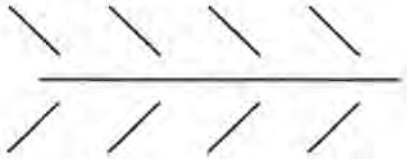


Figure 4.6: An example showing how the minimum-activity heuristic can mislead

look at the relatively inactive terminator map instead of the vertical map, as it should (see Figure 4.6). Therefore, a better heuristic rule would weigh the activities in each map, based on its experience with the domain, before doing the comparison. Such a superior rule was not implemented since the simple rule works reasonably well.

From the control perspective, where should we apply this heuristic rule? It is best to apply it regularly and not just once at the beginning when selecting the strategy. This is because it is possible for situations to change dynamically as one is searching. For example, in Figure 4.7 we see a situation where initially the least active map is the corners map. But as we eliminate some of the long vertical rectangles, the vertical map becomes the least active map and we would save effort if, at that point, we switched to searching the vertical base map.

Accordingly, we choose the map in which to search at this point:

```
1. While we have not yet succeeded & the situation is hopeful
2.   While the search is proceeding well & we have not yet succeeded
3.     Select search strategy
4.     Move spotlight as per strategy
         a. Choose minimally active map
         b. ...
5.     Search for instance of item, starting at spotlight location
       (This may entail moving the spotlight as well.)
6. Done.
```

For the query 'Find all vertical bars inside triangles' the minimal map is initially the corners map. If we employ the minimal-activity heuristic, we will begin searching this map for triangle vertices. If we eliminate a few rectangles, then the vertical map will become minimal and we will benefit by switching to searching it instead.

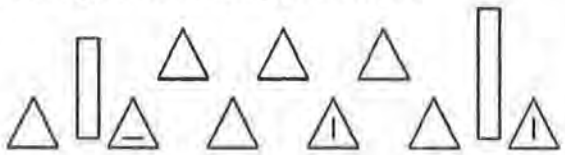


Figure 4.7: An example showing how the minimum-activity map can change as search progresses

Problem 4: Finding Relations is Different from Finding Objects

Now that we have selected the map in which to index, we can get on with the search. At this point control problem 4 presents itself. If we are looking for a relation, we have more things to look for than if we are simply looking for an object. This forces us to devise two logic streams. For objects we must confirm the existence of a shape with properties. For relations we have to do this twice, as well as to confirm the relation between the two objects.[9] This latter task presents additional control and memory management problems because, when we find one of the sought objects, we must remember this and then shift our attention to the new search problem of finding the second object. This second search may employ a search strategy different from the one the first search employs. For example, the second search may take advantage of the knowledge of where the first object is and of the relation which the first object is supposed to bear to the second object. For example, it might now employ a proximity guided search. Moreover, for certain relations – e.g., inside, touching, and connected – the search can be constrained to fall within a certain subimage.

[9]We don't do things necessarily in that order, as we shall see shortly.

We take advantage of this whenever possible by having these relations trigger a visual routine which limits the second search to the relevant subimage. By this means an order n^2 search problem can be reduced to an order n problem, provided there is a good proportion of subimages.

So now our control logic appears as follows.

```
1. While we have not yet succeeded & the situation is hopeful
2.   While the search is proceeding well & we have not yet succeeded
3.     Select search strategy
4.     Move spotlight as per strategy
         a. Choose minimally active map
         b. ...
5.     Search for instance of item, starting at spotlight location
       (This may entail moving the spotlight as well.)
         Case 1: Object
         Case 2: Object - Relation - Object
           a. Find one object
           b. Perform secondary search on the relevant
              subimage for the second object
6. Done.
```

Problem 5: Task Scheduling

We will treat separately the task scheduling considerations for each of Cases 1 and 2.

Case 1, the simpler of the two, is the problem of deciding in what order to confirm the properties and structure of an object. For example, if we are looking for a vertical bar, do we first check that whatever is under the spotlight is vertical, or do we first check that it is a bar? Ideally, we would like to perform all such checks simultaneously, with any failure cancelling all remaining work. However, assuming that resources constrain us to perform one test at a time, then clearly we want to perform those tests earliest which are least expensive and which have the greatest chance of disqualifying the region under the spotlight, thereby saving us from having to do the remaining tests. In the human system, for instance, it appears colour is computed first, preferentially over form. This picture is also complicated by the fact that some properties are dependent on others having been completed. If we want to find a red triangle, and the spotlight of attention is only over one vertex, then we have to recover the entire triangle shape before we test for redness. Otherwise, if the triangle

had two red sides and one green, and the red vertex was under the spotlight, we might accept the whole triangle as red on the evidence of the one vertex. In general, in the 2-D geometry world at least, structures take precedence over properties. That is, before we determine whether an object is closed, convex, vertical, symmetric, etc., we must have computed its isolated form. Later we can verify that this form meets the definition of a triangle, square, or such.

The task scheduling problem for Case 1 is a difficult one. The relative cost of performing a property or structure test, and the chance of the test failing, will doubtlessly vary according to the input. Attempting to solve these problems could amount to a thesis in itself. Therefore, in the present implementation it was decided to make do with a crude solution to this problem. What we do is to avoid all comparisons of relative cost or chance of disqualification, and just to impose an order on the tasks. It was decided to compute the structure first because, as mentioned above, for many properties a minimal structure is a prerequisite. We then compute the properties in the order in which they occur in the query. We make no attempt to order the computations by cost or by chance of failing. However, we did design the program to allow the addition of such logic, should it ever become available. These choices have no big consequences for the current implementation since there are so few properties available that one is hard pressed to find two that one would naturally apply to any one object. The addition of colour, intensity, motion, or depth properties would change this situation.[10]

The task scheduling problem for Case 2 is an expansion on the Case 1 problem. The variety of task permutations is larger in Case 2 because, in computing the properties of two objects, we could conceivably alternate between them, and also because at various points we could try to compute the relation. Whichever way we manage this, we must be careful to keep apart the results of each ongoing computation. Case 2 is also complexified by the addition of a

[10]Our solution to Problem 7, the problem of avoiding duplicate examinations, also makes it advantageous to compute the form before any other properties. Without the form one cannot really compare the object in question with the objects one has previously encountered. As search progresses, these appeals to memory can save one much needless computation.

second search task. That is, whenever one of the objects is found, the search problem changes its nature. The found object becomes a pivotal point around which the search for the second object takes place.

Our solution to the Case 2 problem is to make four types of schedule. As with the Case 1 problem, we decided to ignore the scheduling of property checks. However, experience with early versions of the system revealed that great savings could be had by properly scheduling the relation computation. Let us assume that object #1 is the first object found and that it becomes the pivotal object. Let O_1 stand for the complete computation of object #1's presence; let O_2 stand for the complete computation of object #2's presence; and let R stand for the completion of the computation of the relation between object #1 and #2. Then, the four possible schedules are O_1-O_2-R, O_1-R-O_2, O_2-R-O_1, and R-O_1-O_2. In the last three schedules the relation is given an abstract form to relate before that form has been entirely confirmed to be the desired object. (For example, we could thereby determine that some object is inside some other before actually inspecting closely to see which objects they are.) The choice of schedule is determined by the relative cost of computing the relation as compared to computing the objects. Experience with a variety of test images was the basis for judging the relative cost.

The way we handle the problem of the secondary search about the pivotal object, namely, the first object found, is rather elegant. After finding one of the objects we define a second search task, and then we recursively call the entire FIND algorithm on that task. This task is no longer a task of finding a relation between two objects. Instead it is the task of finding the object not yet found, where this object also has the property of being related to the pivotal object. We manage this by creating a property out of the relation by instantiating one of its variables with the found object. We then simply add this new property to the list of properties to be confirmed. This approach has the advantage of simplicity. There is no need to define new methods of managing a second search within a search, no need to keep separate working memories, etc. Pascal, with its ability to handle recursive calls, does this automatically for us. All that

is needed is manually to Push a few images onto the image stack and then to Pop them back upon returning. Incidentally, we also copy the termination conditions from the main task to this subtask. Thereby, if the required number of objects is found, the subtask will terminate as it should.

Now our control logic appears as follows.

```
1. While we have not yet succeeded & the situation is hopeful
2.   While the search is proceeding well & we have not yet succeeded
3.     Select search strategy
4.     Move spotlight as per strategy
         a. Choose minimally active map
         b. ...
5.     Search for instance of item, starting at spotlight location
       (This may entail moving the spotlight as well.)
         Case 1: Object
    1.     - Confirm structure present at spotlight
    2.     - Confirm properties for this structure
         Case 2: Relation(Object1,Object2)
    1.     - If Object1 is costly to compute and an
               isolated object (i.e., triangle and square)
           then
    2.       Find the isolated region under the
               spotlight; call it X1.
    3.       If Object2 is costly to compute and an
               isolated object
             then (R-O1-O2 case)
    4.         Compute relevant subimage to search
    5.         Perform a new search task to find ALL
                 isolated objects, X2s, having the property
                 of being R(X1,obj).
    6.         If X1 is indeed Object1 then
    7.           While termination conditions are not satisfied
    8.             If X2 is indeed an Object2.
    9.               Accept each R(X1,X2) as a solution
             else (O2-R-O1 case)
   10.         Compute relevant subimage to search
   11.         Perform a new search task to find as many
                 Object2's as are necessary, each also
                 having the property of being R(X1,object2).
   12.         If X1 is indeed Object1 then
   13.           Accept each R(X1,Object2) as a solution
           else
   14.       If Object1 is under the spotlight then
   15.         If Object2 is costly to compute and an
                 isolated object
               then (O1-R-O2 case)
   16.           Compute relevant subimage to search
   17.           Perform a new search task to find ALL
                   isolated objects, X2s, having the property
                   of being R(X1,obj).
   18.           While termination conditions are not satisfied
   19.             If X2 is indeed an Object2.
   20.               Accept each R(Object1,X2) as a solution
               else (O1-O2-R case)
   21.           Compute relevant subimage to search
   22.           Perform a new search task to find as many
                   Object2's as are necessary, each also
                   having the property of being
```

```
                R(Object1,Object2).  Accept each of these
                as a solution
23.       - If Object1 was not found under spotlight, then
24.           Repeat the above, except now look for Object2
              first and Object1 second. (This step is not
              necessary, but it is added to preserve some sym-
              metry between Objects 1 and 2.  Without this step,
              if we don't find Object1 under the spotlight,
              then we move the spotlight elsewhere.  With this
              step, we would instead switch to looking for
              Object2 under the spotlight.  Ideally, both
              objects would have equal chance from the start,
              but this is not possible in this serial model.)
6. Done.
```

Problem 6: What to Remove from the Index Maps as Search Progresses

Inattention to this problem leads to wasted effort because points that were previously evaluated as parts of an object are repeatedly inspected. But an overzealous solution to the problem can have the unwanted effect of trimming potential solutions. The key to solving this problem is knowing what one is searching for and what its properties allow one to get away with while trimming. First, let us consider Case 1 searches only. If one is looking for an isolated object, then, whether or not one finds it, one can remove from consideration all those points connected to the object found within the spotlight. If the object one seeks is not an isolated object, then, when the object is found, all those points that are part of the object and are isolated (ie., that do not abut onto some other form) can be removed. And, if the object is not found attached to any point within the spotlight, then all these points within the spotlight can be removed.[11]

The only additional consideration needed for Case 2 searches is whether or not the relation can be reflexive. If it is not, then as one performs the second search, one had better not find the pivot object. We solve this problem simply enough: whenever we are dealing with a non-reflexive relation, we remove the

[11]Incidentally, all points that are removed are removed from copies of the base maps and not from the original base maps. This is necessary because, if we are searching for a relation, finding one object must not interfere with the search for the other object. We must be able to call up a fresh base map when searching for the other object.

isolated portions of the pivot object from the subimage which is given to the second search task.

Problem 7: Duplicate Examinations

The way to avoid making duplicate examinations of the same object is simply to remember the essential aspects of one's previous activity. Our solution is to remember every successful object found and every failed object found. In the case of relations, if the relation is symmetric, then we also remember every pair of related objects and every pair of objects not so related.[12] If ever we find the same object or pair of objects, we make sure to stop reconsidering it or them immediately. Successes, incidentally, are treated exactly like failures, except that a single flag is set to distinguish them. For Case 1 searches, we find that remembering failures is as important as remembering successes. For Case 2 searches, this holds only if R is symmetric. If R is symmetric (such as *touching, outside, connected,* or *parallel*), then knowing whether R(A,B) is true will tell us whether R(B,A) is true, and hence saves us from having to compute this directly. If R is not symmetric, then we have to go ahead and test R(B,A) independently.

This strategy for handling symmetric relations can at times strain the resources of memory since, for n objects in the image, up to $n(n-1)/2$ pairs have to be remembered. This has not been a problem for the small images we have worked with. On larger problems we could give up on this strategy and only remember the pivot objects.

Now our control logic can be updated to appear as follows:

```
1. While we have not yet succeeded & the situation is hopeful
2.   While the search is proceeding well & we have not yet succeeded
3.     Select search strategy
4.     Move spotlight as per strategy
```

[12]Whether the relation is symmetric or not, we must of course remember the successful relations since we have to report these back to the user. However, we do not need to remember them in order to avoid duplicate examinations. This is done automatically by remembering the pivot objects and, for each pivot object, by locally remembering the second objects which we attempted to relate to it.

```
        a. Choose minimally active map
        b. ...
5.    Search for instance of item, starting at spotlight location
      (This may entail moving the spotlight as well.)
        Case 1: Object
  1.      Confirm structure present at spotlight
  2.      If we haven't examined it before then
            Confirm the properties for this structure
  3.      Remember the object examined
        Case 2: Object - Relation - Object
  1.      Find a pivot object
  2.      If we haven't examined the pivot before then
            Perform a secondary search on the relevant
              subimage for the second object.
              During this second search, if R is symmetric,
              then remember all pairs we attempt to
              relate and do not bother attempting to relate
              pairs that are already on this list.
  3.      Remember the pivot object
6. Done.
```

Problem 8: What to Do When a Search Pass Fails

The final control problem which faces us is what to do in the event we fail to find what we seek. The failure could be indicated to the system by the user's dissatisfaction with a response or by some internal trigger of non-confidence. If the system has confidence in its detection ability, it could simply quit while insisting it has found everything findable. Even people, however, fail to detect things in images, and we would expect our system to perform less well than a person. Therefore, some provision should be made for returning to the assigned task and seeing if a new approach might not yield better results.

The failure to find something can be the result of several things. The image could be noisy; it could be a trick image; or the item sought could occur at a lower scale of resolution. A sought object could appear in an unnatural position or orientation. The system's definition of an object, property, or relation might disagree with the user's. The system might have used a heuristic search strategy that did not guarantee a thorough search. Then again, maybe the system is just defective: it is blind to certain objects or is incapable of reliably detecting certain properties and relations.

For whatever reason a search can fail, the ideal system response would be to diagnose the cause of failure, remove the cause, and then to try again. If the

image is noisy, we could try smoothing the image or lengthening lines to fill any gaps. As a matter of course, we should probably examine the image at a variety of scales and orientations. To handle trick problems we should try not to be misled by context. This could mean artificially removing or distorting parts of the image in the hope of chancing upon the correct "view". If we previously used a heuristic search scheme, then we could return and try a patterned search, one that is guaranteed to cover the entire image, such as a raster scan search. If we keep failing at a particular type of query, then we could simply admit our failing and ask for guidance, clues, or a simpler request.

Unfortunately, none of these methods were implemented. These problems are rather beyond our immediate aims of devising efficient means of computing properties and relations. This control problem is nonetheless a very important one and we should plan for it. The programs were designed to accommodate the future addition of such logic. For now, whenever the initial search stops, the program simply returns whatever has been found.

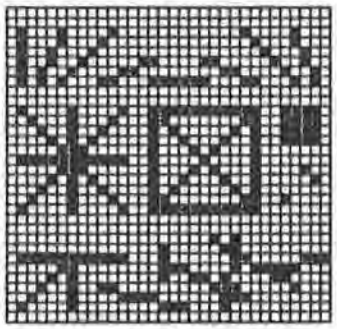
Chapter 5

System Performance and Evaluation

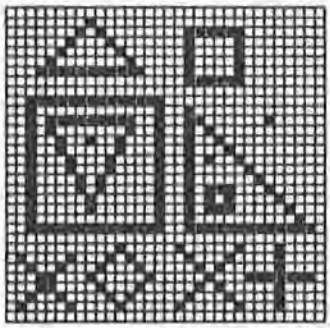
The system[1] has undergone numerous tests in order to evaluate the ability of the visual routine language to compute a wide variety of properties and relations, and in order to evaluate the control logic described in Section 4.D. In this chapter the results of these tests are presented and these results are evaluated.

5.1 Experiments and Results

Privately, the system has been tested on numerous small images. To save space all the test figures have been merged into four images:

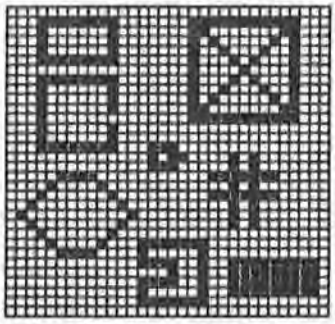


Test Image 1

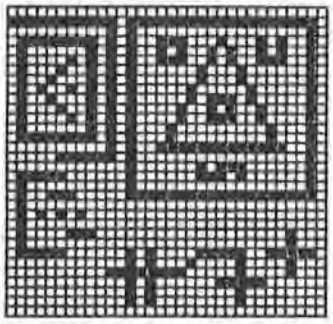


Test Image 2

[1]The program is written in Borland International's Turbo Pascal, version 3.0. The computer on which the system currently runs is a standard IBM pc.



Test Image 3



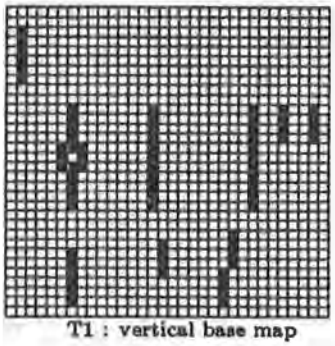
Test Image 4

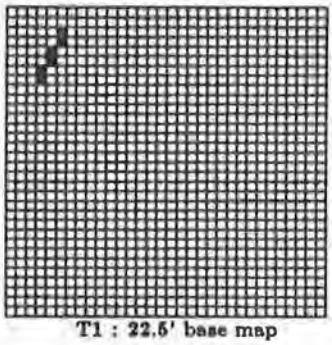
The first image is specialized to test the performance of the base map calculations. The remaining three images contain a variety of figures. Each of these three is subjected to 28 questions. Twenty-two of those questions are each designed to evaluate a particular visual routine. The remaining six are designed to exercise the control logic and confirm that it can handle a mixture of structures, properties, and relations.

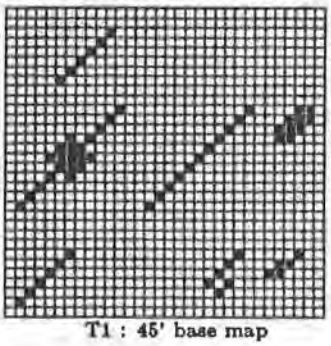
All the queries are of the "FIND ALL" variety since these are more instructive than the "FIND ANY" variety. The system responds to each query by individually displaying each instance found. To save space here all the responses are overlayed into one image. Unfortunately this makes it difficult to separate the responses. A count of the number of instances that were found by the system is provided, and with a little work the reader can compute these himself. If the system makes any errors, these are explained.

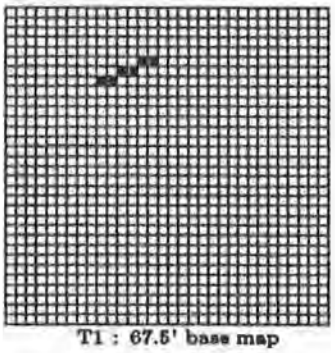
5.1.1 TEST 1 : The Performance of the Base Map Calculations

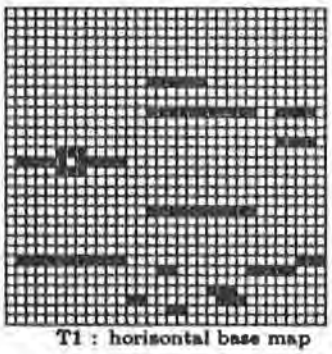
The following base maps were computed:

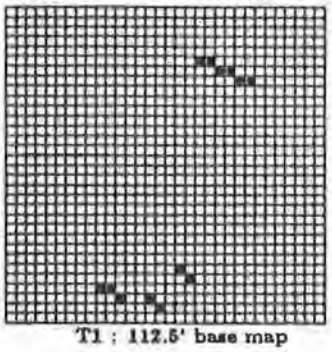

T1 : vertical base map

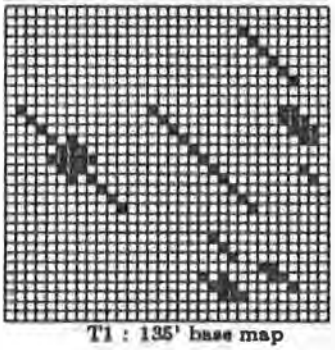

T1 : 22.5' base map

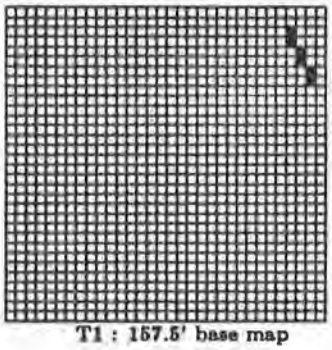

T1 : 45' base map

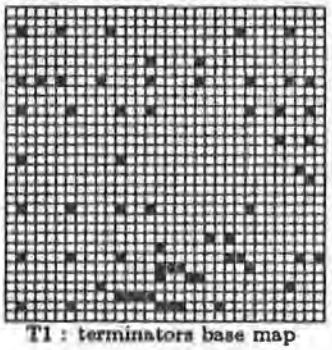

T1 : 67.5' base map

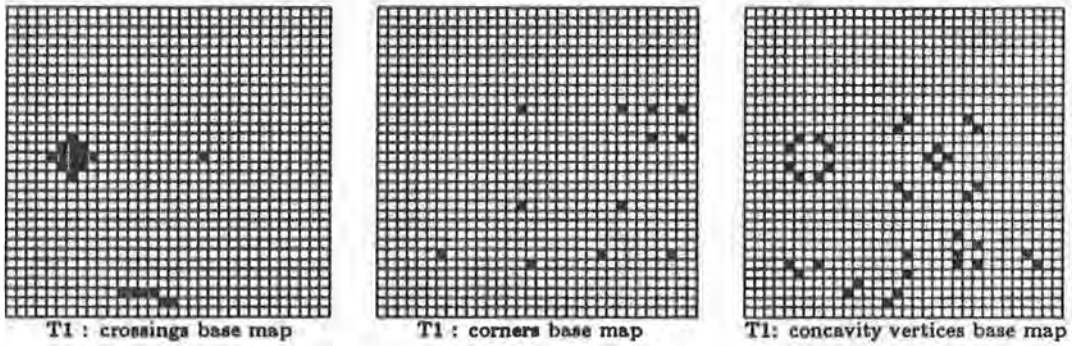

T1 : horisontal base map


T1 : 112.5' base map


T1 : 135' base map


T1 : 157.5' base map


T1 : terminators base map



T1 : crossings base map T1 : corners base map T1: concavity vertices base map

Discussion:

The purpose of the base map algorithms[2] is to give us sufficiently accurate base maps with which we can carry on the business of testing visual routines. To this end we could have appealed to some magic mechanism and created the base maps manually. However, it was felt that generating base maps ought to be a task within the scope of visual routines, and that it would be a good test of their ability. Furthermore, there may actually be times when one wants to create second order maps from the original base maps; for example, as when looking for edges in the terminator maps (see Figure 3.2 above). In those cases it would be nice to have the operations managed by the general purpose visual routines rather than by some special purpose mechanism.

As we can see from our test, the base map algorithms work well enough for our purpose. The oriented edge computations do get confused somewhat in the presence of a block of active pixels, as we can see in the centre of the star. This can be excused, given that there is no real edge present there.

Failures also occur for edges whose orientation does not fall into exactly one of the eight categories, as we can see in the lines along the bottom of the image. In such cases part of the line is given one orientation and the other part, a nearby orientation. Part of this problem is due to the low resolution image and the use of 3x3 masks; another part is due to the lack of a good iterative competitive scheme which would have neighbouring active pixels "settle down" within a common orientation map.

[2]You can find the actual base map algorithms in Appendix A.

Of course, the problem with the oriented edges propagates to those base maps which depend on these edges for their own computation. All the non-orientation base maps – corners, crossings, terminators, and concavity vertices – are affected in this way.

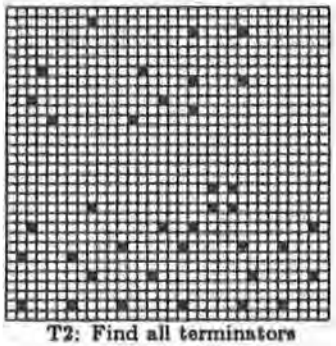
But, if we just remember these mild troubles, we should feel confident in relying on our base maps.

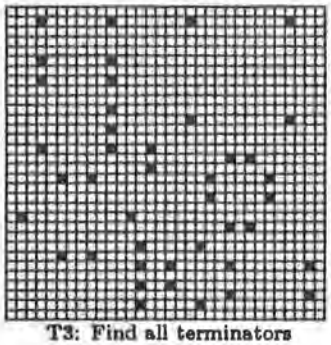
Note: the solid block on the right edge of the image is not a legitimate object in our 2-D geometric world. We were curious to see how the algorithms would behave for such an object. Presumably, passing the image through an edge detector and thereby reducing solid blocks to edge figures would allow us to handle such objects. In section 5.1.3 below, another solid object was included to test the CONVEX and CLOSED visual routines. The routines handle it properly; it is convex but not simple-closed.

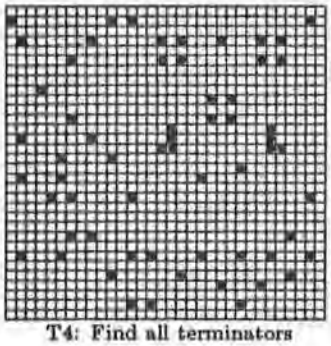
5.1.2 TEST SET 2: The Performance of the Structure Routines

a. Find all terminators

Instances found: T2-37, T3-42, T4-56. All are correct.

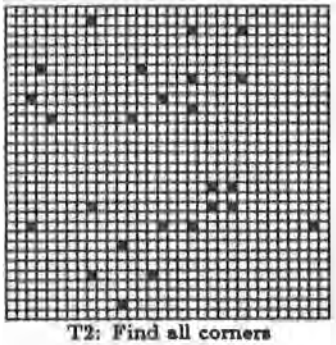

T2: Find all terminators

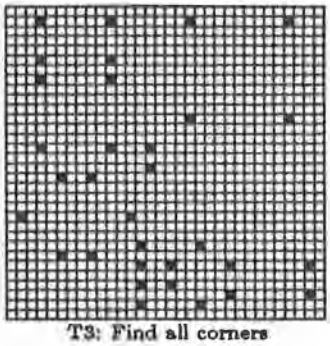

T3: Find all terminators

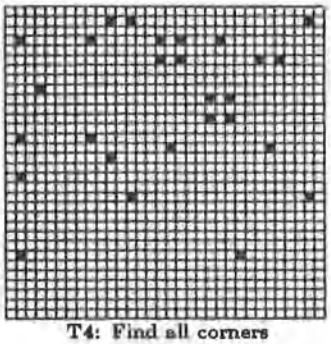

T4: Find all terminators

b. Find all corners

Instances found: T2-25, T3-28, T4-27. All are correct.


T2: Find all corners

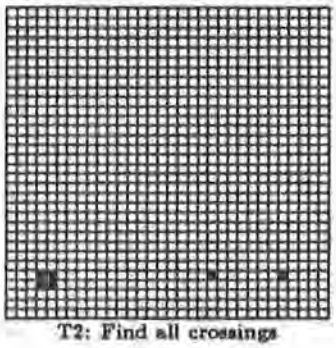

T3: Find all corners

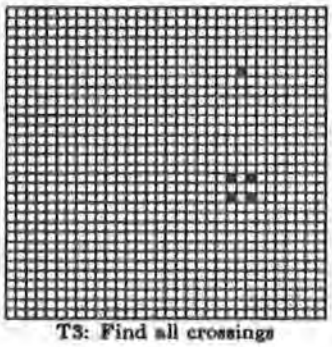

T4: Find all corners

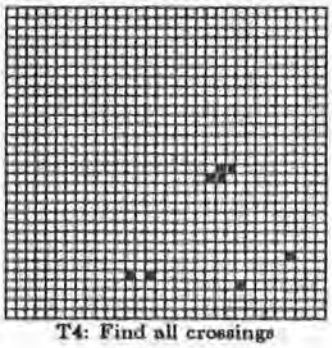
c. Find all crossings

Instances found: T2-3, T3-5, T4-5.

The instances for T2 and T3 are correct. The base map calculations had trouble with the "N" figure of T4.

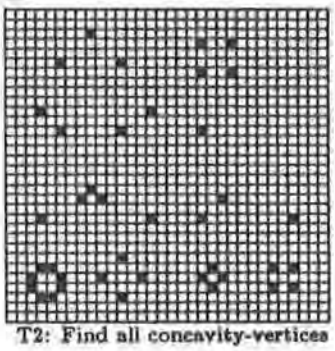

T2: Find all crossings

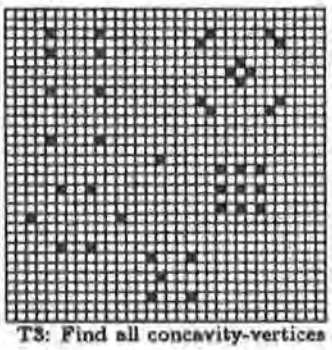

T3: Find all crossings

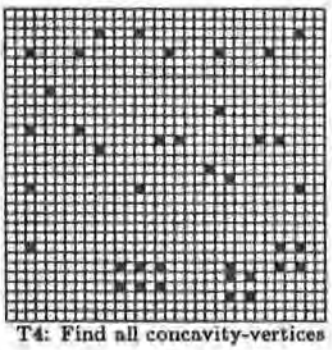

T4: Find all crossings

d. Find all concavity vertices

Instances found: T2-36, T3-48, T4-37. All are correct.

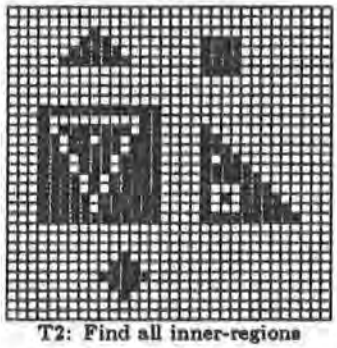

T2: Find all concavity-vertices


T3: Find all concavity-vertices

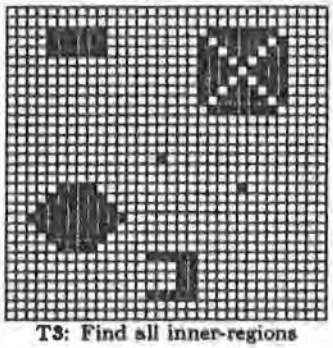

T4: Find all concavity-vertices

e. Find all inner-regions

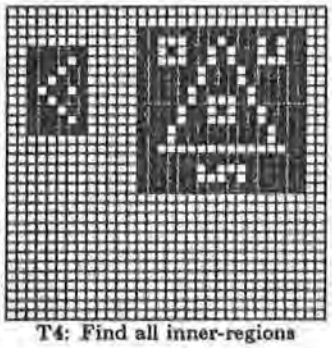
Instances found: T2-7, T3-9, T4-5. All are correct.



T2: Find all inner-regions



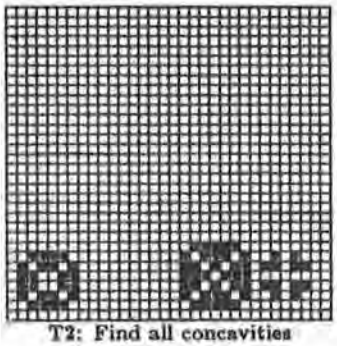
T3: Find all inner-regions



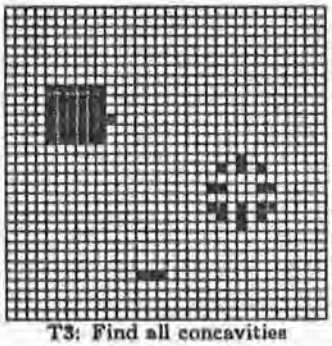
T4: Find all inner-regions

f. Find all concavities

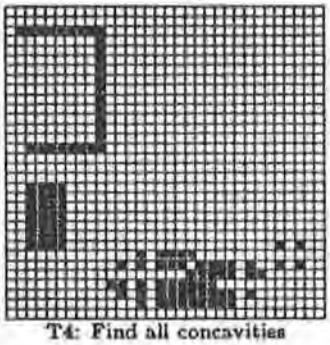
Instances found: T2-12, T3-10, T4-14. All are correct.



T2: Find all concavities



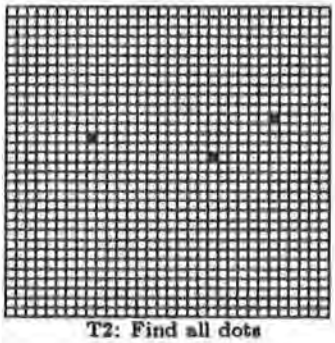
T3: Find all concavities

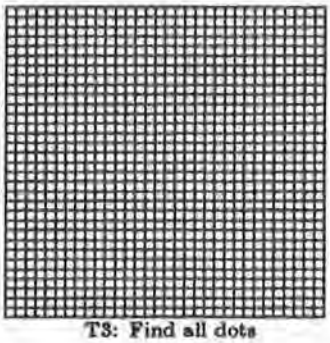


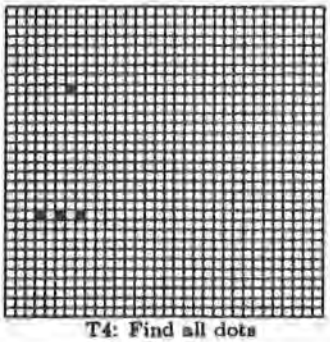
T4: Find all concavities

g. Find all dots

Instances found: T2-3, T3-0, T4-4. All are correct.


T2: Find all dots

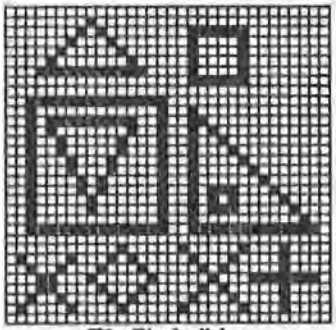

T3: Find all dots

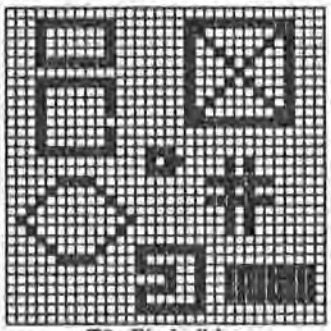

T4: Find all dots

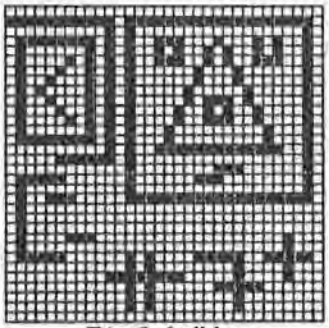
h. Find all bars

Instances found: T2-31, T3-40, T4-44.

The errors made in the base map calculations influence the results here. In T3 the solid region generates five bars: the four outer edges and one large solid diagonal. Also in T3 the little box in the centre generates one vertical box on the left side. The remainder is recognized as a single horizontal bar. In T4 we lose the diagonal part of the bottom-centre figure; the 'N' figure in the square generates one horizontal bar; and the triangle gnerates five bars, two of which are the small vertical bars near the two lower vertices.

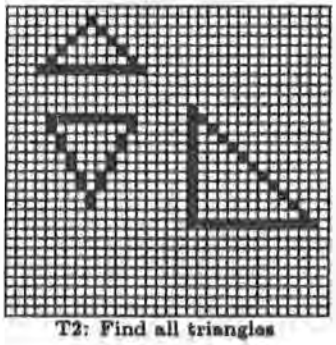

T2: Find all bars

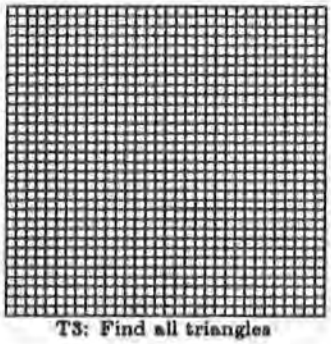

T3: Find all bars

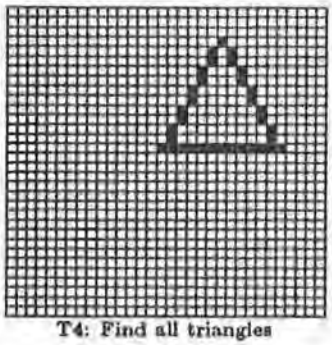

T4 : find all bars

i. Find all isolated-triangles

Instances found: T2-3, T3-0, T4-1. All are correct.


T2: Find all triangles

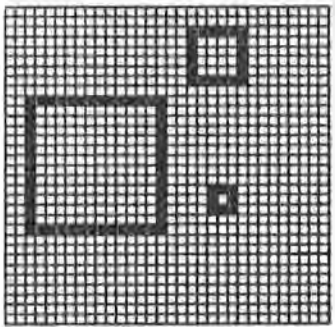

T3: Find all triangles

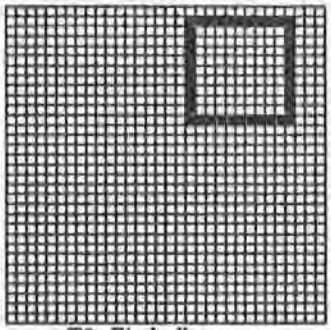

T4: Find all triangles

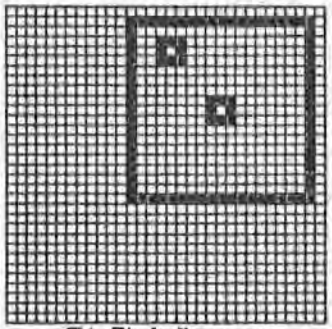
j. Find all isolated-vertical-squares

Instances found: T2-3, T3-1, T4-3.

The square in T3 is not isolated, but was recognized nonetheless. It would be trivial to alter the visual routine which computes squares so that it does not recognize such cases, but there would be little point in that.

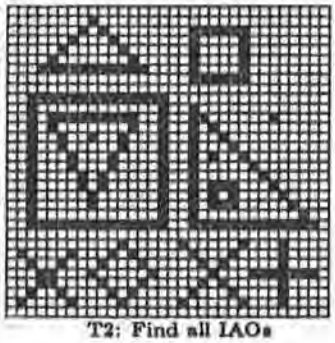

T2: Find all squares

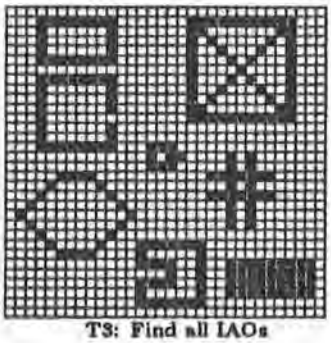

T3: Find all squares

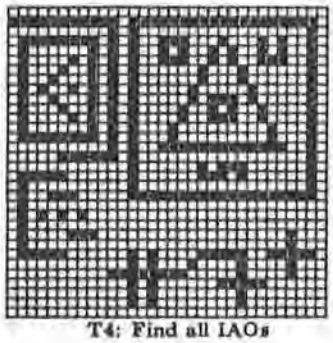

T4: Find all squares

k. Find all isolated-arbitrary-objects (IAOs)

Instances found: T2-13, T3-8, T4-18. All are correct.


T2: Find all IAOs


T3: Find all IAOs


T4: Find all IAOs

Discussion on the structure routines:

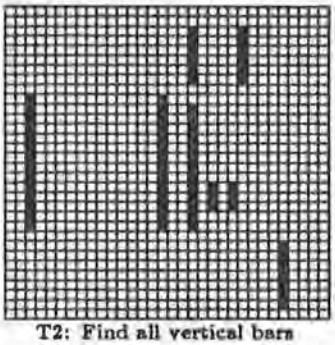
The preceding batch of tests thoroughly exercises each of the visual routines used to confirm the presence of an object or a feature. It is important that we be able to trust our basic object finding logic before moving on to test the property and relation finding logic. Allowing for errors in the base maps, all the queries were correctly answered.

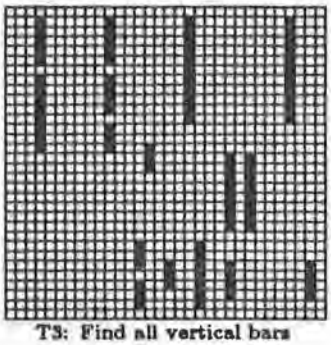
5.1.3 TEST SET 3: The Performance of the Property Routines

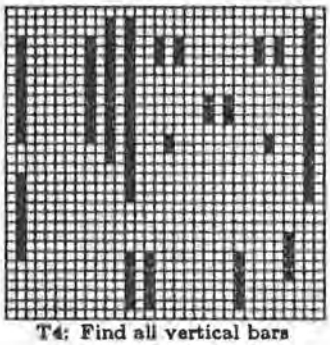
a. Find all vertical bars

Instances Found: T2-8, T3-16, T4-18.

In T4 the two vertical tips at the base of the triangle generate small vertical bars.


T2: Find all vertical bars

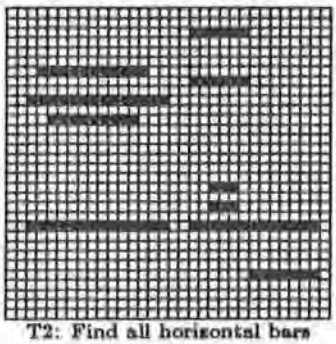

T3: Find all vertical bars

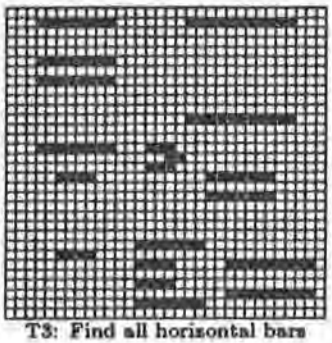

T4: Find all vertical bars

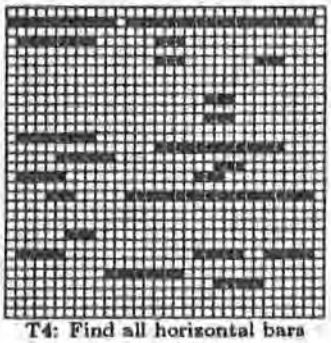
b. Find all horizontal bars

Instances Found: T2-10, T3-17, T4-21.

The little box in T3 generates a single horizontal bar. The same happens for the 'N' figure in T4.

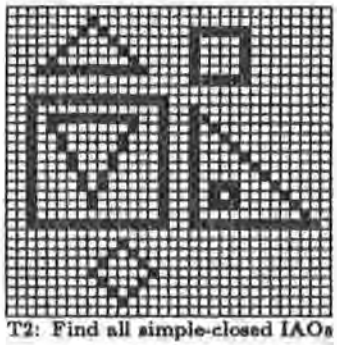

T2: Find all horizontal bars

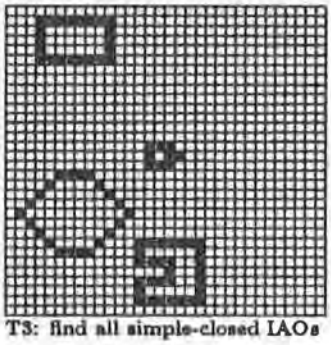

T3: Find all horizontal bars

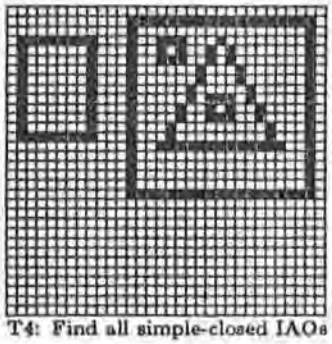

T4: Find all horizontal bars

c. Find all simple-closed IAOs

Instances Found: T2-7, T3-4, T4-5. All are correct.


T2: Find all simple-closed IAOs

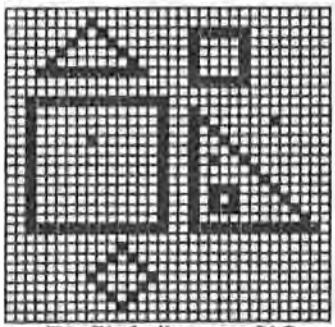

T3: find all simple-closed IAOs

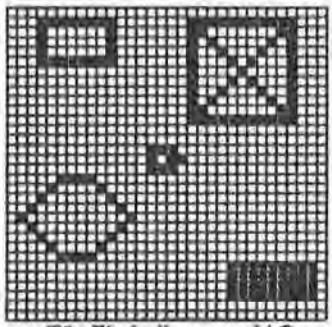

T4: Find all simple-closed IAOs

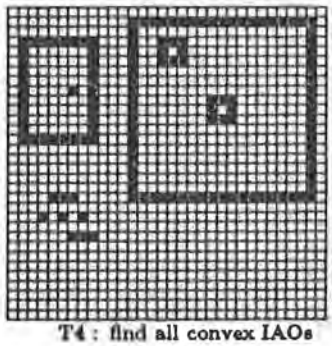
d. Find all convex IAOs

Instances Found: T2-9, T3-5, T4-10.

In both T2 and T4, the triangles which are inside squares are both interpreted to possess concavities on their outer edge. In T2 this is on the bottom vertex. In T4 it is on the two bottom vertices. The problem lies with the narrow locality of the make-convex basic operation.


T2: Find all convex IAOs


T3: Find all convex IAOs


T4 : find all convex IAOs

Discussion on the property routines:

Allowing for the errors in the base maps, the property routines all work.

The time required to find the vertical bars was found to be about half that to find the horizontal bars. This anomaly is due to the task scheduling convention employed where structures are always found before properties. In order to confirm the presence of a horizontal bar attached to the spotlight, we first find any bar there, and then apply the horizontal test. To find the bar means iterating through all the orientation maps, the vertical map being first, the horizontal one being fifth. A superior method would only look at bars in the horizontal map from the start.[3]

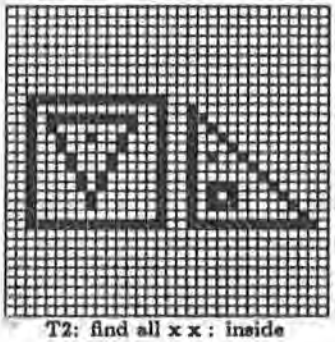
[3]This situation is alleviated somewhat by our indexing only within the horizontal map. The problem is that some of these points are also attached to non-horizontal bars. For instance, take the corners of squares.

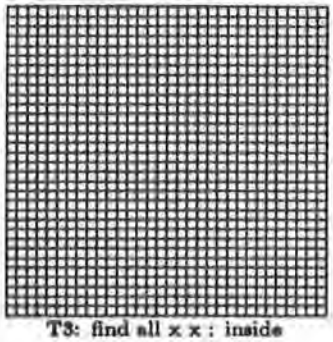
5.1.4 TEST SET 4: The Performance of the Relation Routines

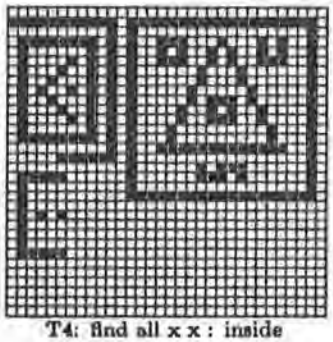
a. Find all IAOs inside IAOs

Instances Found: T2-5, T3-0, T4-13. All are correct.

Note that 'inside' is defined as "being within the scope of". According to this definition, the containing object need not be closed and the contained object must be entirely within the convex hull of the containing object.

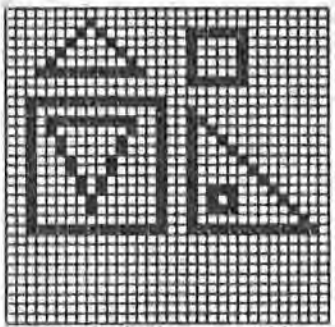

T2: find all x x : inside

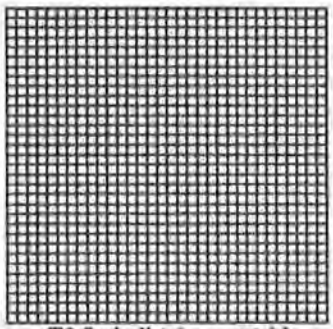

T3: find all x x : inside

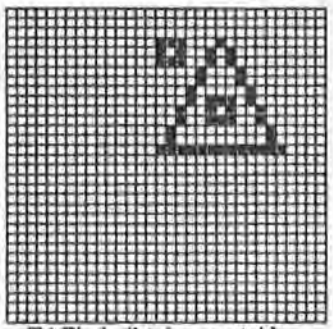

T4: find all x x : inside

b. Find all triangles outside squares

Instances Found: T2-8, T3-0, T4-2. All are correct.

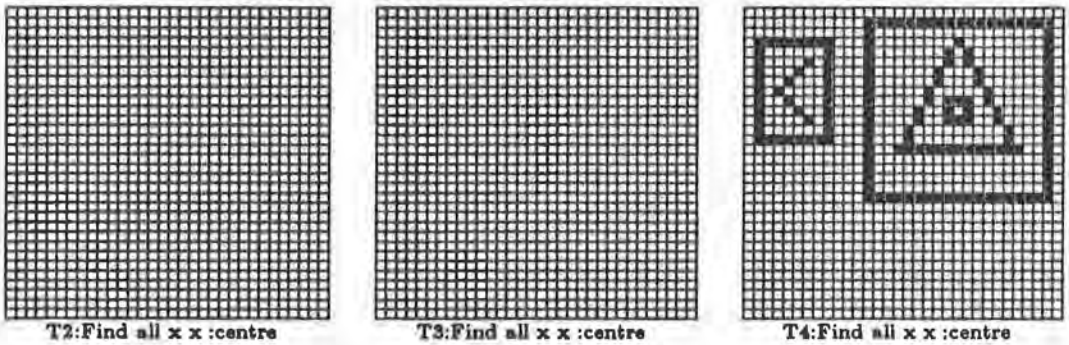

T2:find all tri sqr : outside


T3:find all tri sqr:outside


T4:Find all tri sqr:outside

c. Find all IAOs centred-in IAOs

Instances Found: T2-0, T3-0, T4-4. All are correct.



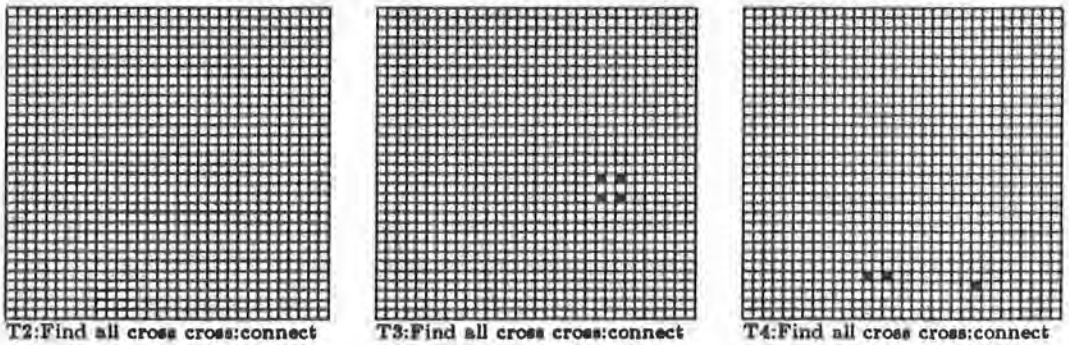
d. Find all bars touching bars

Instances Found: T2-28, T3-48, T4-34.

Allowing for the base maps, all the instances are correct.

e. Find all cross-points connected-to cross-points

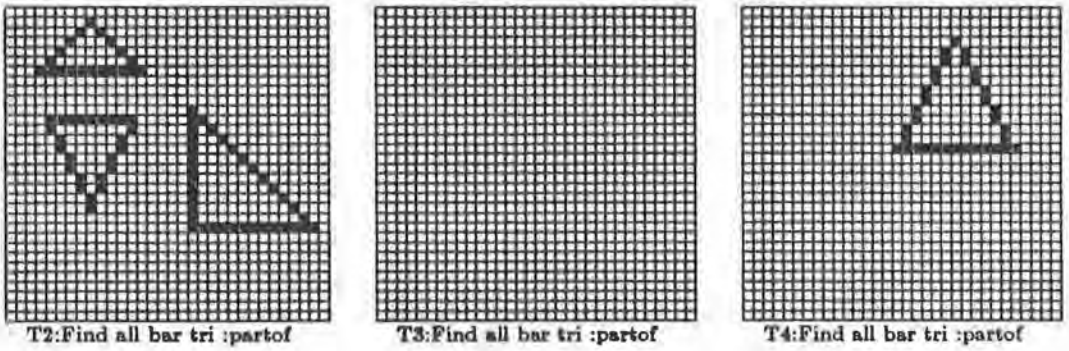
Instances Found: T2-0, T3-6, T4-3. All are correct.



f. Find all bars part-of triangles

Instances Found: T2-9, T3-0, T4-5.

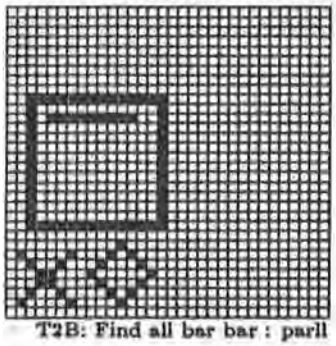
In T4, the two tips at the base of the triangle are recognized as vertical bars.

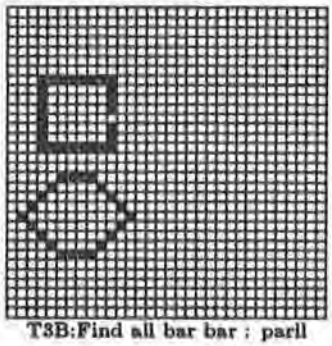


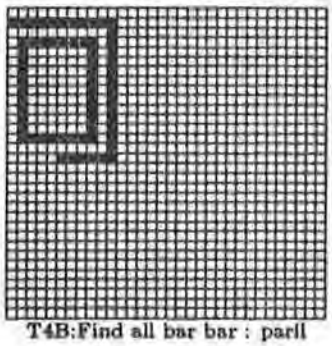
g. Find all bars parallel-to bars

When applied to the entire images of T2, T3, and T4, this query caused a stack-overflow system crash. There are simply too many sets of parallel bars in each image. The following tests were done with only a portion of each image.

Instances Found: T2-10, T3-11, T4-9. All are correct.


T2B: Find all bar bar : parll


T3B:Find all bar bar : parll


T4B:Find all bar bar : parll

Discussion on the relation routines:

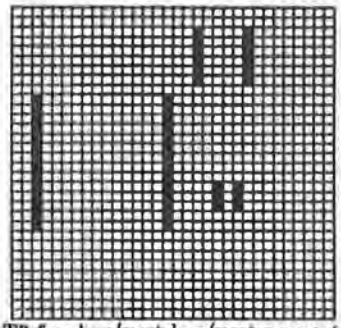
Allowing for what the structure and property routines recognize, the relation routines all work.

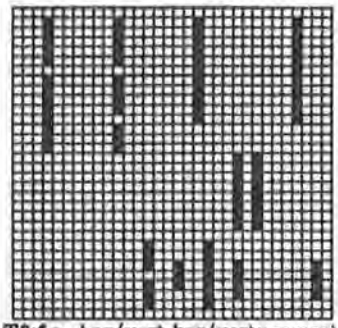
5.1.5 TEST SET 5: Performance Aspects of the Control Logic

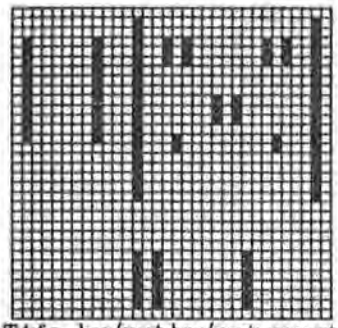
a. Find all vertical bars connected-to vertical bars

Instances Found: T2-3, T3-13, T4-9.

In T4, the two vertical tips of at the base of the triangle were recognized as vertical bars.


T2:f.a. bar/vert bar/vert:connect

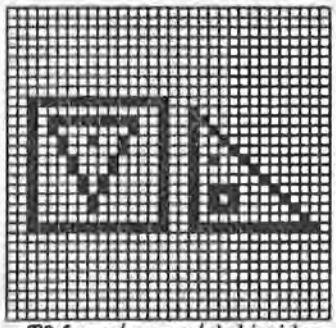

T3:f.a. bar/vert bar/vert:connect

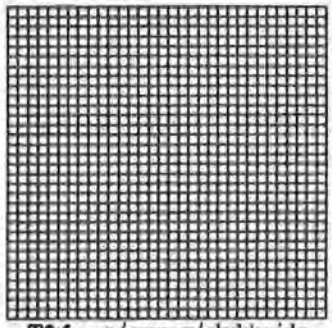

T4:f.a. bar/vert bar/vert:connect

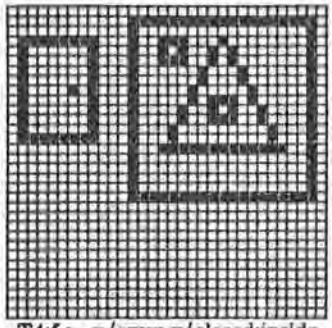
b. Find all convex IAOs inside closed IAOs

Instances Found: T2-4, T3-0, T4-4.

The triangles in T2 and T4 were not considered convex because of the perceived concavities near some of their vertices.

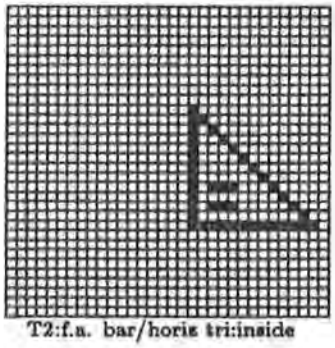

T2:f.a. x/cnvx x/clsd:inside

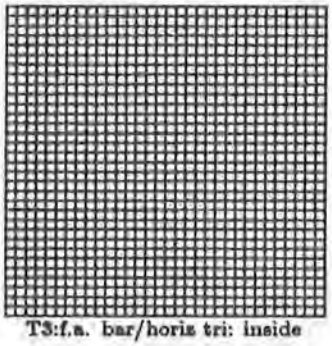

T3:f.a. x/cnvx x/clsd:inside

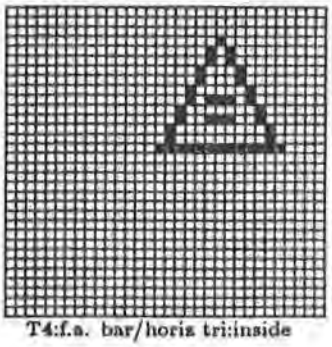

T4:f.a. x/cnvx x/closed:inside

c. Find all horizontal bars inside triangles

Instances Found: T2-2, T3-0, T4-2. All are correct.


T2:f.a. bar/horiz tri:inside

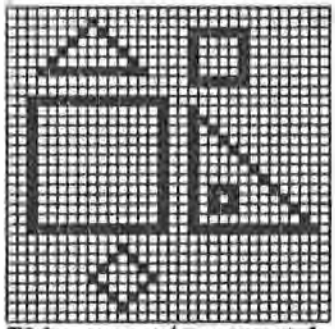

T3:f.a. bar/horiz tri: inside

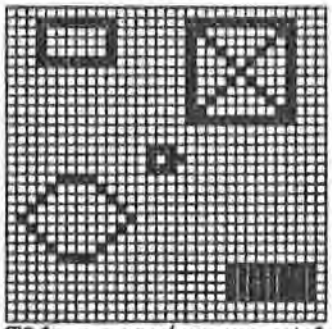

T4:f.a. bar/horiz tri:inside

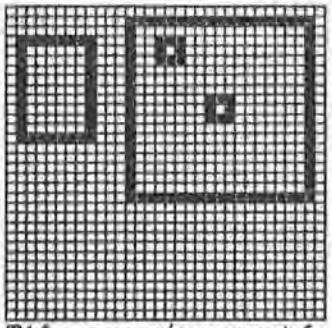
d. Find all corners part-of convex IAOs

Instances Found: T2-22, T3-20, T4-16.

Apart from the previously mentioned problem with the non-convex triangles, and the peculiarities of the the base map computations of the corners, the instances found are correct.

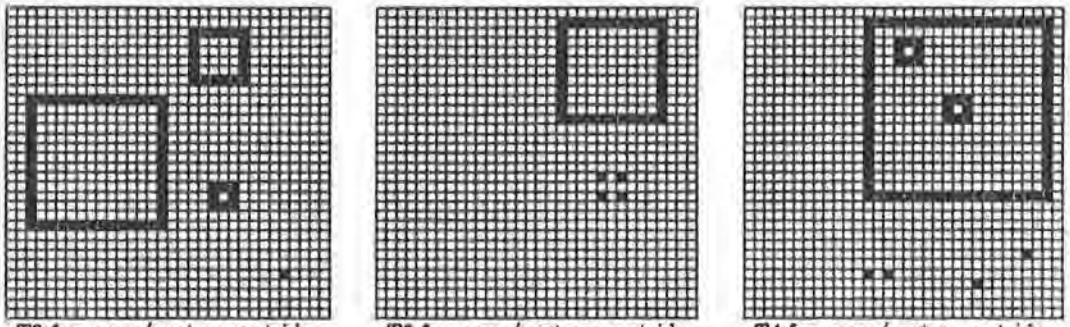

T2:f.a. corner x/convex:partof


T3:f.a. corner x/convex: partof


T4:f.a. corner x/convex:partof

e. Find all vertical crosses outside squares

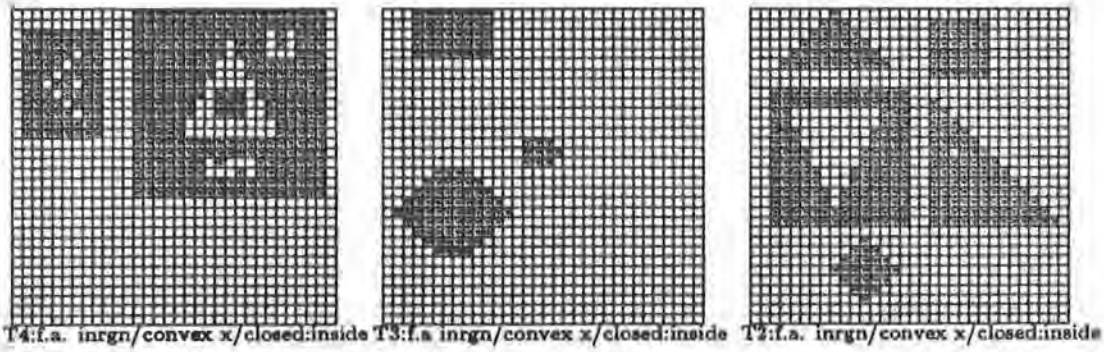
Instances Found: T2-3, T3-4, T4-12. All are correct.



f. Find all convex inner-regions inside closed IAOs

Instances Found: T2-7, T3-3, T4-7.

In T4 the interior of the triangle is not considered convex. Again, this is due to the narrow locality of the *mk-convex* operation.



Discussion on the performance of the control logic:

The control logic successfully manages a mixture of structures, properties, and relations.

5.2 General Evaluation

We found that the visual routines were strained somewhat at the task of computing base maps. Although visual routines were never intended to compute base maps, the maps they produced were adequate for our purposes.

The routines for confirming the presence of geometric objects work well enough. The *bar* routine's failings were caused by the failings in the base map computations.

The property and relation routines appear to work correctly. They were only limited by the structures they were given. The base map computation for convexity vertices could be improved by expanding the region of locality of the *make-convex* operation. This would solve the problem of the two triangles (one in T2 and the other in T4) not being recognized as convex.

The control logic successfully manages the search. However, it is inefficient in certain respects. For example, when searching for vertical bars, we should not use the vertical map solely to *guide* the search for bars in the image; we should search for the vertical bars *in* the vertical map as though it were an image. Also, when watching the system search, it is apparent that much time is spent reevaluating the properties and structures of regions we have already examined. A major revision to the control logic would have it create a high-level semantic map of what objects are in the image. For example, it could label all the points in a square as "square points". Another possibility would have it create a map of object locations. The search could then proceed in this much simpler map rather than in the image.

Chapter 6

Conclusion

We set out to see what methods within the visual routine paradigm are suited to the task of efficiently finding the geometric properties and relationships of figures in an image. To this end we reviewed the work that had been done on visual routines, preattentive vision, and the spotlight of attention, the latter two being key components of visual routine thinking. We then embarked upon our own study of the problem in hopes of constraining the form these methods should take. We chose a visual domain rich in geometric properties and relations and free of other distracting problems. This was the world of simple 2-D geometric shapes, constructed from lines of uniform width and intensity. We then approached the problem of designing efficient visual routines from the perspective of language design. We asked ourselves what would make such a visual routine language descriptively and procedurally adequate.

We saw that a visual routine language was descriptively complete for a visual domain if it could compute every property and relation imaginable in that domain. However, we discovered that establishing such completeness would be difficult. We then saw reasons why descriptive adequacy might not be all that important a property of a visual routine language, especially in comparison to procedural adequacy.

Turning then to procedural adequacy, we divided this issue into concerns over *resources* and concerns over *convenience*. We decided that our visual rou-

tine language would be resource-efficient if it had some guarantee of containing the best algorithms possible for computing visual properties and relations in the 2-D geometric shape world. We decided that establishing such a guarantee would require a thorough study of the task of computing visual properties and relations. So we embarked on this difficult course and ended up going part way. We first looked at image representation and decided that a square grid tessellation on which only one-pixel-width lines appear was suitable. We then looked at what basic features could be efficiently computed directly from the image. We found that simple local properties could make ready use of parallelism and thus could be efficiently computed. Such properties include: the orientation of any single line at a point; whether or not a line is present at the point; whether the point is an end point of some line; whether two, three, or more lines cross at a point; the type of pattern that can occur when lines cross; and the curvature at a point. We found that to compute some of the orientation-independent local properties, like curvature or crossing pattern, it was space-efficient to introduce further stages of processing rather than to match for one of possibly thousands of individually oriented patterns. We then moved on to consider non-local properties that could be computed directly from the image. We saw how, by using a Hough transform type of technique, we could again introduce several stages of processing and thereby efficiently compute the presence of some size and position invariant properties.

We continued on to look at second-order properties and relations which could be computed once the initial properties had been computed and positioned in a topographic map. Beyond the obvious reapplication of the previous local computations to these new maps, we determined that the efficient computation of complex geometric properties and relations would have to emerge out of a combination of parallel and serial methods. But just which parallel and serial methods was not obvious. Therefore, it was argued, we should put a temporary end to a priori analysis and go gather some experience – go try assorted methods and see which ones work.

We then turned from discussing resource issues to convenience issues. A

number of capabilities were outlined which we would hope to find in a visual routine language. The language which was subsequently developed had all these features.

An argument was made for why we would be wise to embed the visual routine system in a system which interrogated images about what they contained rather than tried to recognize what they contained. In an interrogation system, since we were given the objects, properties, and relations which we sought, we had only to locate their defining characteristics in the image. This problem was much more constrained than the problem of examining all the image features and indexing them to all the candidate properties, objects, and relations in order to see which ones were present in the image.

We then began a study of the visual routine system which was developed as part of this thesis. After reviewing the overall design and the key data structures, we looked at the basic operations. We saw that several of these operations were used frequently in the writing of the visual routines. In particular these were: moving maps, spreading activation, intersecting maps, combining maps, clearing maps, removing portions of a map, and exiting a routine when a map is empty or non-empty. An argument was made that this finding signified the genuine importance of these particular basic operations. We then compared our set of basic operations with that given in Ullman (1984) and concluded that Ullman's was incomplete in that it failed to include the most elementary operations.

Next we turned to the visual routines which were written with these operations. We gave all twenty-two property, relation, and structure visual routines in detail.

Next we saw what would make an ideal query language, and we were shown the actual, much simpler, query language which had been implemented.

We were then introduced to eight key control issues which were encountered during development. These issues were:

1. deciding on a search strategy, both initially and as search progresses

2. setting the diameter of the attention spotlight

3. choosing the feature map to index within

4. treating object searches differently from relation searches

5. scheduling tasks when several tasks are required to confirm an instance of an object or of a relation over objects.

6. deciding what parts of an index map can be removed after we have searched for an instance at the index point

7. managing memory to avoid duplicate examinations of the same area of the image

8. deciding what to do when a particular search pass fails

Each of these issues was explained in detail. It was conjectured that they represent universal issues which would arise for anyone trying to apply visual routines to the problem of searching and matching objects and relations in an image.

Finally, we were shown the system in action. We saw that the visual routines worked well, and that the search control logic worked well too. We finished by evaluating the system's overall performance. We found that the visual routines were strained somewhat at the task of computing base maps. At computing object descriptions, properties, and relations they worked well. At computing object descriptions, they probably managed as well as they did only because geometric objects were definable in terms of geometric properties and relations. We also saw aspects of the control logic which could be improved. In particular, it would be advantageous to be able to search base maps or intermediate maps as though they were images. Also, it would be effort saving to create representations locating the objects in the image and to conduct our search in these representations and not in the image.

In summary: We set out to find efficient methods to compute visual properties and relations while working in the visual routine paradigm. After choosing

a suitable experimental domain we deduced some constraints on the input to our visual routine language. When deduction seemed no longer appropriate, we turned to experimentation and developed a number of actual routines and a strategy for controlling their application; these together made an effective working system. We gave a complete account of the heart of this system: the basic operations, the visual routines, and their controlling logic. By developing the system we discovered a number of control issues which we believed were intrinsic to the visual search problem. The discovery of the basic operations, the visual routines, and the control issues constituted the principle contribution of this thesis to vision science.

Directions for future development

The work in this thesis presents opportunities for a number of natural extensions. One of top priority is the working-out of a consistent and powerful visual query language. With such a language we could develop more complete control strategies for managing groups of objects, groups of properties, patterns within groups, properties of properties, and so on. The addition of negation to our language would create interesting control problems. But even with the simple language we employed, there is much progress that can be made. We could implement methods for handling different scales of objects and also rotated objects (e.g., square versus diamond). The visual routines could be made more robust. They could be made to operate on noisier images, images with objects of variable intensity, partial objects, and occluding objects. They could be made to return degrees of match rather than the binary yes/no that they currently return. We could add new search strategies, such as raster search. We could add search heuristics like the proximity and similarity heuristics Koch and Ullman (1984) mention. In the domain of search control, we could add means to remember interpretations of features and thus avoid their reinterpretation. There are also numerous visual routines to be written for handling curvature, density, number, size, symmetry, relative position, and so on. The beauty of the visual routine paradigm is that it gives structure to all these pursuits.

Bibliography

[1] Ballard, Dana H., and Brown, Christopher M., *Computer Vision*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1982.

[2] Hearn, Donald, and Baker, Pauline, *Computer Graphics*, Englewood Cliffs, NJ: Prentice Hall, Inc., 1986.

[3] Hillis, W. Daniel, *The Connection Machine*, Cambridge, MA: The MIT Press, 1985.

[4] Horn, Berthold Klaus Paul, *Robot Vision*, Cambridge, MA: The MIT Press, 1986.

[5] Hurlbert, Anya, and Poggio, Tomaso, "Spotlight on Attention", MIT AI memo #817, April, 1985.

[6] —— , "Visual Attention in Brains and Computers", *Nature*, Vol. 321, #12, June 1986, pp.651-652.

[7] Jolicoeur, Pierre; Ullman, Shimon; and Mackay, Marilynn; "Curve tracing: A possible basic operation in the perception of spatial relations", in *Memory and Cognition*, Vol. 14, No. 2, 1986, pp. 129-40.

[8] Julesz, Bela, and Bergen, J.R., "Textons, The Fundamental Elements in Preattentive Vision and Perception of Textures", in *The Bell System Technical Journal*, Vol. 62, No. 6, July-August 1983, pp. 1619-45.

[9] Julesz, Bela, "Toward an Axiomatic Theory of Preattentive Vision", in *Dynamic Aspects of Neocortical Function*, ed. by G.E. Edelman, W.E. Gall, and W.M. Cowan, 1984, pp.585-612.

[10] ——, "Preattentive Human Vision, A Link between Neurophysiology and Psychophysics", in *Handbook of Physiology: Volume V, Higher Functions of the Brain*, ed. by Fred Plum et al., American Physiological Society, forthcoming in 1987.

[11] Kaneff, S. (ed.), *Picture Language Machines*, London: Academic Press, 1970.– Narasimhan, R., "Picture Languages", pp. 1-30.

[12] Koch, Christof, and Ullman, Shimon, "Selecting One Among the Many: A Simple Network Implementing Shifts in Selective Visual Attention", M.I.T. A.I. Memo 770, 1984.

[13] Little, James J., "Parallel Algorithms for Computer Vision", M.I.T. A.I. Memo 928, November, 1986.

[14] Lowe, David G., *Perceptual Organization and Visual Recognition*, Boston: Kluwer Academic Publishers, 1985.

[15] Mackworth, Alan K., "Adequacy Criteria for Visual Knowledge Representation", University of British Columbia technical report 87-4, also to appear in *Computational Processes in Human Vision*, edited by Zenon Pylyshyn, Norwood N.J.: Ablex Publishers, 1988 (in press), 23 pp.

[16] Mahoney, James, and Ullman, Shimon, "Image Chunking: Defining Spatial Building Blocks for Scene Analysis", to appear in *Computational Processes in Human Vision*, edited by Zenon Pylyshyn, Norwood N.J.: Ablex Publishers, 1988 (in press), 41 pp.

[17] Marr, David, *Vision*, San Francisco, CA: W.H. Freeman & Co., 1982.

[18] Nakayama, K, and Silverman, G.H., "Serial and Parallel Processing of Visual Feature Conjunctions", in *Nature*, Vol. 320, 1986, pp.264-265.

[19] Nake, F., and Rosenfled, A. (eds.), *Graphic Languages*, Amsterdam: North-Holland Pub. Co., 1972.

[20] Preston Jr., Kendall, "Languages for Parallel Processing of Images", in *Real-Time/Parallel Computing: Image Analysis*, New York: Plenum Press, 1981, pp. 145-58.

[21] Pylyshyn, Zenon, "The Role of Location Indexes in Spatial Perception: A Sketch of the FINST Spatial-index Model", University of Western Ontario Cognitive Science memorandum COGMEM 23, July, 1987, 31 pp.

[22] ——, "Here and There in the Visual Field", to appear in *Computational Processes in Human Vision*, edited by Zenon Pylyshyn, Norwood N.J.: Ablex Publishers, 1988 (in press), 25 pp.

[23] Stanton, R.B., "The Interpretation of Graphics and Graphic Languages", in F. Nake and A. Rosenfled (eds.), *Graphic Languages*, Amsterdam: North-Holland Pub. Co., 1972, pp. 144-162.

[24] Tanimoto, Stephen, and Klinger, A. eds., *Structured Computer Vision*, New York: Academic Press, 1980.

[25] Treisman, Anne, and Gelade, Garry, "A Feature-Integration Theory of Attention", in *Cognitive Psychology*, Vol. 12, 1980, pp.97-136.

[26] Treisman, Anne, and Schmidt, Hilary, "Illusory Conjunctions in the Perception of Objects", in *Cognitive Psychology*, Vol. 14, 1982, pp. 107-141.

[27] Treisman, Anne, "Preattentive Processing in Vision", in *Computer Vision, Graphics, and Image Processing*, Vol. 31, 1985, pp.156-177.

[28] ——, "Properties, Parts, and Objects", in *Handbook of Perception and Performance*, Vol. 2, ed. by K. Boff, L. Kaufman, and J. Thomas, John Wiley & Sons, Inc., 1986.

[29] ——, "Features and Objects in Visual Processing", in *Scientific American*, November 1986, pp.114B-125.

[30] Ullman, Shimon, "Visual Routines", in *Cognition*, Vol. 18, 1984, pp. 97-159.

[31] ——, "Artificial Intelligence and the Brain: Computational Studies of the Visual System", in *Annual Review of Neurosciences*, Vol. 9, 1986, pp. 1-26.

[32] Walters, Deborah, "Selection and Use of Image Features for Segmentation of Boundary Images", in *Proceedings of Graphics Interface '86, Vision Interface '86: 26-20 May, 1986: Vancouver British Columbia*, Toronto: Canadian Information Processing Society, 1986, pp. 318-324.

Appendix A

Source Programs

The structure, property, and relation visual routines were given in Chapter 4. Here we present miscellaneous other visual routines, Pascal subroutines for selected basic operations, and the Pascal program that computes the 3x3 convolution masks.

A.1 The Routines that Build the Base Maps

BASEMAPS.VR

```
{--------------------------
{Directory of Register Maps
{--------------------------
{ 0     = input image
{ 1-8   = orientation maps 1= vertical, 2=22.5 deg NE, ...
{ 9     = composite of all terminators in each orientation map
{ 10    = crossings : intersections of two or more oriented edges
{                     which aren't corners
{ 11    = corners :   intersections of two or more oriented edges
{                     which are also terminators
{ 12    = concavity vertices
{ 13-17 = Reserved for future base maps
{ 18,19 = Internal System Utility
{ 20-29 = Work registers free for use by the visual routine programmer
{ 30-40 = FIND/MATCH local work registers
{ 41-49 = Unused
{ 50-62 = Index maps: markable copies of base registers to be searched
{--------------------------
 call orient   {place orientation maps in regs 1-8
 call endpts   {place composite (all orients) terminator map in reg 9
 call intersec {place composite (all orients) crossing rgns in reg 10;
               {place composite (all orients) corners map in reg 11
 call invertex {place all inner pts of concave vertices in reg 12
               {invertex makes use of crossing regions
```

ORIENT.VR

```
{*** BUILD ORIENTATION MAPS in regs 1-8
do z 1 8
  begin
                                {- find oriented narrow bars
    conv3 line 0 z (z-1)        {convolve with oriented narrow-bar detector
    cbop 0 [ z z                {remove any points not in image}
    kop -20 + z z               {clip weak points (< 20) : part 1
    deunit 2 z z                {singletons are not bars
    kop  20 + z z               {clip weak points (< 20) : part 2
  end
mov  1 11 8     {save 1-8 in 11-18 for future use
call compete    {have each compete with his two neighbors for dominance
do z 1 8
  begin
    deunit 2 z z                {remove units introduced by competition
    {-- "conv3 line" above followed by clipping typically removes the
    {-- endpoints of lines.  Here we grow them back again.
    conv3 grow.a z 20 (z-1)     {  grow the "right" side of the map
    cbop 0 [ 20 20
    conv3 grow.b z 21 (z-1)     {  grow the "left" side of the map
    cbop 0 [ 21 21
    bop z + 20 z
    bop z + 21 z                {  add these points back in
  end
call compete                    { repeat competion
{--  now, if any are left out we had better replace them : a hack
set_all 20 0
do z 1 8
  bop 20 + z 20                 {take union of all orientations
kop 1 < 0 21                    {set all image points to 1
bop 21 - 20 20                  {20 now has the points to be re-included
do z 1 8
  begin
    mov.z 22
    sp_act_lin 2 22 (z+10)      {grow out in original precompetition edge
    cbop 22 ] 20 22             {recover points to be re-included
    bop z + 22 z                {add them back in
    cbop 0 ] z z                {reset to image intensity
  end
```

ENDPTS.VR

```
set_all 9 0                         {clear terminator map}
do z 1 8
  begin
    conv3 end.pt z (z+20) (z-1)     {convolve with end.pt finder
    cbop z ] (z+20) (z+20)          {constrain to oriented bar
    bop 9 + (z+20) 9                {add to terminator map
  end
```

INTERSEC.VR

```
set_all 10 0                    {init crossings map
set_all 11 0                    {init corners   map
do x 1 8
   if_nz x   call interse2 x    {put in 10 all intersectn orient regions
                                {and in 11 all intersectng orient points
mov 11 20
mov 11 21
{-------------  corners
```

```
cbop 9 [ 11 11                  {corners are crossing terminators
cbop 11 [ 0 11                  {reset values to initial image values
mov 11 25
sp_act_1_a 2 25 20              {find intersec regns attached to corners
{-------------  crossings
sp_act_1_a 2 20 10          {recover normal cornrs & crossng regions
bop 10 - 20 10              {erase these, leavng captured trouble regns
bop 10 + 21 10              {add back normal crossing points
cbop 10 [ 0 22              {reset intersects to initial image values
bop 22 - 25 10              {crossings are intersecting non-corners
```

INTERSEC2.VR

```
{%1 is outside calling loop parm (1-8)
mov %1 20
spread 2 20 21
do y (%1+1) 8
  begin
    {-- find intersect regions
    spread 2 y 22
    bop y + 20 25     {take union of both orientations
    cbop 21 ] 22 23   {intersect the two smoothed orientations
    cbop 23 ] 25 25   {intersect with the original two orientations
    bop 10 + 25 10    {add anything in common to crossings map
    {-- now find intersect points
    cbop 20 ] y 25
    bop 11 + 25 11    {add anything in common to temp map
  end
```

COMPETE.VR

```
{*** Have each orientation map compete with his two immediate neighbors
compete_3 8 1 2 21
compete_3 1 2 3 22
compete_3 2 3 4 23
compete_3 3 4 5 24
compete_3 4 5 6 25
compete_3 5 6 7 26
compete_3 6 7 8 27
compete_3 7 8 1 28
mov 21 1 8
```

INVERTEX.VR

```
bop 10 + 11 20      {take all crossings and corners
sp_act_lin 2 20 0   {sp act for two steps in image
sp_act_lin 2 20 0
mk_convex 20 21     {make this convex
bop 21 - 20 12      {remove the foreground material ; done
```

A.2 The Relation Preprocessing Routines

Each such routine uses the following parameter passing conventions:

```
{%1 = img reg
{%2 = pt  reg
{%3 = local img reg
{%4 = d unused (assume 5 for now)

INSIDE1.VR
----------
{return in local img reg the convex hull portion
{of the image containing pt reg
mov %1 20
mov %2 21
until_nc 20
  mk_convex 20 20
until_nc 21 50
  sp_act_lin 2 21 20 {using 8-conn, sp act to recover this rgn
cbop 21 [ %1 %3

CONNECT1.VR, PARTOF1.VR, and TOUCH1.VR
--------------------------------------
{return in local img reg all that is connected to pt-reg
mov %2 %3
sp_act_1_a 2 %3 %1
```

A.3 Miscellaneous Visual Routines

The following routine removes the isolated portion of the structure in register STR from the map in register MAP which is a subset of the image register IMG.

```
mov STR 20
mov STR 21
mov IMG 22
mov MAP 23
sp_act_lin 2 20 22   {20 is widened at points touching anything else.
bop 20 - STR 20      {remove the struct
sp_act_lin 2 20 21   {reactivate points in struct that touch something
                     {20 now has what we want to keep
set_all 24 1
bop 24 - 20 25       {create inverse of what we want to keep
cbop 25 [ 21 25      {find portion of inverse which is touching STR
                     {this is what we can discard
kop 63 > 25 25       {set these to max intensity
bop MAP - 25 MAP     {and remove them from MAP
```

A.4 Pascal Subroutines for Selected Basic Operations

The following represents but a small portion of the complete system which is currently about 4000 lines in size. Included are declarations of the more important data structures used by the basic operation subroutines.

The map registers have been partitioned into pyramids. This was an early design decision intended to treat each register as an "image pyramid" (Tanimoto & Klinger ,1980). The *d* parameter which appears in the parameter list of most subroutines selects the level of the pyramid at which the basic operation is applied. Level 0 is 1x1, level 1 is 2x2, level 2 is 4x4, and so on up to level 5 which is 32x32. As development progressed other issues grew in prominence and the pyramid idea fell by the wayside. In the current system all the operations operate at level 5. The idea of applying visual routines to image pyramids remains a worthwhile one.

```
{-------------------------------------------------}
Const
  max_id          = 31;  {max image dimension}
  max_id_log2     = 5;   {log2(max_id +1)}
  max_regs        = 63;  {# of map regs, minus one (0))}
Type
  ireg_array      = array[0..47,0..31] of integer;
  reg_array       = array[0..47,0..31] of byte;
  reg_pointer     = ^reg_array;
  reg_stack_ptr   = ^reg_stack_rec;
  reg_stack_rec   = record
                      reg         : reg_array;
                      reg_stk_ptr : reg_stack_ptr;
                    end;
  reg_dim_pos     = array[0..max_id_log2] of integer;
  mask_3x3_array  = array[0..3,0..7,-1..1,-1..1] of integer;
  connectvty_arr  = array[1..2,-1..1,-1..1] of boolean;
Const
  {-- partitioning of registers into subregisters --}
  rdx_st : reg_dim_pos = (32,32,32,32,32,0 );
  rdx_fn : reg_dim_pos = (32,33,35,39,47,31);
  rdy_st : reg_dim_pos = (30,28,24,16, 0,0 );
  rdy_fn : reg_dim_pos = (30,29,27,23,15,31);
  rd_lng : reg_dim_pos = ( 1, 2, 4, 8,16,32);
  connected : connectvty_arr = (((false, true,false), {4-connectivity}
                                 ( true,false, true),
                                 (false, true,false)),
                                ((true, true, true), {8-connectivity}
                                 ( true,false, true),
                                 ( true, true, true)));

{-------------------------------------------------}
{-------  Centroid  ------------------------------}
{-------------------------------------------------}
Procedure Centroid(r1,res,d:integer);
{return in res a point whose location is the centroid of the
 figure in r1, and whose value is the average value of the figure.}
  Var
    i,j,val,n     : integer;
    sum,sumx,sumy : real;
  Begin
    with EXP do
```

```
    begin
      n     := 0;
      sum   := 0.0;
      sumx  := 0.0;
      sumy  := 0.0;
      for i := rdx_st[d] to rdx_fn[d] do
      for j := rdy_st[d] to rdy_fn[d] do
        if reg[r1]^[i,j] > 0 then
           begin
             n := n + 1;
             val := reg[r1]^[i,j];
             sum  := sum  + val;
             sumx := sumx + i * val;
             sumy := sumy + j * val;
           end;
      FillChar(reg[res]^,1536,black);
      reg[res]^[Round(sumx/sum), Round(sumy/sum)] := Round(sum/n);
    end;
  End;
{-------------------------------------------------}
{-------  Compete_3  -----------------------------}
{-------------------------------------------------}
Procedure Compete_3(r1,r2,r3,res,d:integer);
{for each pt, if r2 >= r1 and r3 then retain r2 in res, ow zero res.}
  Var
    i,j : integer;
  Begin
    with EXP do
    begin
      FillChar(temp_reg,1536,black);
      for i := rdx_st[d] to rdx_fn[d] do
      for j := rdy_st[d] to rdy_fn[d] do
         if (reg[r1]^[i,j] <= reg[r2]^[i,j]) and
            (reg[r3]^[i,j] <= reg[r2]^[i,j])  then
            temp_reg[i,j] := reg[r2]^[i,j];
      reg[res]^ := temp_reg;
    end;
  End;
{-------------------------------------------------}
{-------  Cond_Bin_Op_Regs  ----------------------}
{-------------------------------------------------}
Procedure Cond_Bin_Op_Regs(op: char; r1,r2,res,d:integer);
  Var
    i,j,ii : integer;
  Begin
    with EXP do
    begin
      FillChar(temp_reg,1536,black);
      for i := rdx_st[d] to rdx_fn[d] do
      for j := rdy_st[d] to rdy_fn[d] do
      if reg[r1]^[i,j]  > 0 then
      if reg[r2]^[i,j]  > 0 then
         case op of
          '+': begin
                 ii := reg[r1]^[i,j] + reg[r2]^[i,j];
                 if ii > white then  temp_reg[i,j] := white
                               else  temp_reg[i,j] := ii;
               end;
          '-': begin
                 ii := reg[r1]^[i,j] - reg[r2]^[i,j];
                 if ii < black then  temp_reg[i,j] := black
                               else  temp_reg[i,j] := ii;
```

```
            end;
          '*': begin
                 ii := reg[r1]^[i,j] * reg[r2]^[i,j];
                 if ii > white then  temp_reg[i,j] := white
                               else  temp_reg[i,j] := ii;
               end;
          '/': ii := Round(reg[r1]^[i,j] / reg[r2]^[i,j]);
          '<': if reg[r1]^[i,j] < reg[r2]^[i,j]  then
          {min}      temp_reg[i,j] := reg[r1]^[i,j]
               else  temp_reg[i,j] := reg[r2]^[i,j];
          '>': if reg[r1]^[i,j] > reg[r2]^[i,j]  then
          {max}      temp_reg[i,j] := reg[r1]^[i,j]
               else  temp_reg[i,j] := reg[r2]^[i,j];
          ']': temp_reg[i,j] := reg[r1]^[i,j];
          '[': temp_reg[i,j] := reg[r2]^[i,j];
        end;
      reg[res]^ := temp_reg;
    end;
  End;
{-------------------------------------------------}
{-------  Convolve_3x3_32  -----------------------}
{-------------------------------------------------}
Procedure Convolve_3x3_32(mask_name: string_10; r1,res,a : integer);
  Var
    i,j,m,n,ix,iy,ii,i4,j4,iii,mask_id : integer;
    ireg  : ireg_array;
  Begin
    if mask_name = 'line'   then mask_id := 0 else
    if mask_name = 'grow.a' then mask_id := 1 else
    if mask_name = 'grow.b' then mask_id := 2 else
    if mask_name = 'end.pt' then mask_id := 3 else
                                 mask_id := -1;
    if mask_id >= 0 then
       begin
         if a < 0 then  a := a + 8;
         FillChar(ireg,3072,black);
         with EXP do
         begin
           for i := 0 to 31 do
           for j := 0 to 31 do
             if reg[r1]^[i,j] > 0 then
                for m := -1 to 1 do
                  begin
                    ix := i - m;
                    if (ix >=0) and (ix < 32) then
                    for n := -1 to 1 do
                      begin
                        iy := j - n;
                        if (iy >=0) and (iy < 32) then
                           ireg[ix,iy] := ireg[ix,iy] + reg[r1]^[i,j] *
                                          mask_3x3[mask_id,a,m,n];
                      end;
                  end;
           FillChar(reg[res]^,1536,0);
           for i := 0 to 31 do
           for j := 0 to 31 do
             begin
               {iii := abs(ireg[i,j]);}
               if ireg[i,j] > 0 then
                  begin
                    iii := ireg[i,j] shr 8;
                    if iii > white then  reg[res]^[i,j] := white
```

```
                                        else  reg[res]^[i,j] := iii;
                    end;
                end;
            end;
        end;
    End;

{-------------------------------------------------}
{-------  K_Op_Reg  ------------------------------}
{-------------------------------------------------}
Procedure K_Op_Reg(op: char; k,r1,res,d:integer);
  Var
    i,j,ii : integer;
  Begin
    with EXP do
    begin
      FillChar(temp_reg,1536,black);
      for i := rdx_st[d] to rdx_fn[d] do
      for j := rdy_st[d] to rdy_fn[d] do
        if (reg[r1]^[i,j] > 0) then
          case op of
           '+':  begin
                   ii := k + reg[r1]^[i,j];
                   if ii > white then temp_reg[i,j] := white else
                   if ii < black then temp_reg[i,j] := black else
                                      temp_reg[i,j] := ii;
                 end;
           '-':  begin
                   ii := k - reg[r1]^[i,j];
                   if ii > white then temp_reg[i,j] := white else
                   if ii < black then temp_reg[i,j] := black else
                                      temp_reg[i,j] := ii;
                 end;
           '*':  begin
                   ii := k * reg[r1]^[i,j];
                   if ii > white then temp_reg[i,j] := white else
                   if ii < black then temp_reg[i,j] := black else
                                      temp_reg[i,j] := ii;
                 end;
           '/':  begin {--- NB. Reverse Division ---}
                   ii := Round(reg[r1]^[i,j] / k);
                   if ii > white then temp_reg[i,j] := white else
                   if ii < black then temp_reg[i,j] := black else
                                      temp_reg[i,j] := ii;
                 end;
           '<':  if k < reg[r1]^[i,j]  then
           {min}       temp_reg[i,j] := k
                 else temp_reg[i,j] := reg[r1]^[i,j];
           '>':  if k > reg[r1]^[i,j]  then
           {max}       temp_reg[i,j] := k
                 else temp_reg[i,j] := reg[r1]^[i,j];
          end;
        reg[res]^ := temp_reg;
    end;
  End;
{-------------------------------------------------}
{-------  Make Convex Step -----------------------}
{-------------------------------------------------}
Procedure Make_Convex_Step(r1,r_res,d:integer);
{Note, the last points found should signal the biggest gaps}
```

```
{NB, One could gain speed by performing this in place, with no
 diffence in the result, if one performs until no change in r1.
 However, since I plan to use the single step function,
 I leave it as is.
 ----------------------}
  Var i,j : integer;
  t : reg_array;
  Label Found, NotFound;
  Begin
    t                 := EXP.reg[r1]^;
    EXP.reg[r_res]^ := EXP.reg[r1]^;
    for i := rdx_st[d]+1 to rdx_fn[d]-1 do
    for j := rdy_st[d]+1 to rdy_fn[d]-1 do
      if t[i,j] = 0 then
        begin
          if t[i+1,j] > 0 then
            if t[i,j-1] > 0 then
              begin if (t[i-1,j-1] > 0) or (t[i-1,j] > 0) then
                        goto Found; end
            else
            if t[i,j+1] > 0 then
              begin if (t[i-1,j+1] > 0) or (t[i-1,j] > 0) then
                        goto Found; end;
          if t[i-1,j] > 0 then
            if t[i,j-1] > 0 then
              begin if (t[i+1,j-1] > 0) or (t[i+1,j] > 0) then
                        goto Found; end
            else
            if t[i,j+1] > 0 then
              begin if (t[i+1,j+1] > 0) or (t[i+1,j] > 0) then
                        goto Found; end;
          if t[i,j+1] > 0 then
            if t[i-1,j] > 0 then
              begin if (t[i-1,j-1] > 0) or (t[i,j-1] > 0) then
                        goto Found; end
            else
            if t[i+1,j] > 0 then
              begin if (t[i+1,j-1] > 0) or (t[i,j-1] > 0) then
                        goto Found; end;
          if t[i,j-1] > 0 then
            if t[i-1,j] > 0 then
              begin if (t[i-1,j+1] > 0) or (t[i,j+1] > 0) then
                        goto Found; end
            else
            if t[i+1,j] > 0 then
              begin if (t[i+1,j+1] > 0) or (t[i,j+1] > 0) then
                        goto Found; end;
          goto NotFound;
          Found: EXP.reg[r_res]^[i,j] := 1;
          NotFound:
          end;
  End;

{-------------------------------------------------}
{-------  Odd Man Out  ---------------------------}
{-------------------------------------------------}
Procedure Odd_Man_Out(r1,r_res,d:integer);
{Return, in an entier reg to itself, a point with highest value}
  Var i,j,max,imax,jmax : integer;
  Begin
    max  := 0;
```

```
    imax := rdx_st[d];
    jmax := rdy_st[d];
    for i := rdx_st[d] to rdx_fn[d] do
    for j := rdy_st[d] to rdy_fn[d] do
      if EXP.reg[r1]^[i,j] > max then
         begin
           max := EXP.reg[r1]^[i,j];
           imax:= i;
           jmax:= j;
         end;
    FillChar(temp_reg,1536,black);
    temp_reg[imax,jmax] := EXP.reg[r1]^[imax,jmax];
    EXP.reg[r_res]^ := temp_reg;
  End;
{-------------------------------------------------}
{-------  Sp_Activate_Line  ----------------------}
{-------------------------------------------------}
Procedure Sp_Act_Lin(con_typ,r_sp,r_rel,d:integer);
{For one time step all 4/8 neighbors of any activated point are tested.
 If any is active in r_rel, then it is also activated in r_sp}
  Var
    i,j,m,n,ii,jj : integer;
  Begin
    with EXP do
    begin
      temp_reg := reg[r_sp]^;
      for i := rdx_st[d] to rdx_fn[d] do
      for j := rdy_st[d] to rdy_fn[d] do
        if reg[r_sp]^[i,j] > 0 then
           for m:= -1 to 1 do
             begin
               ii := m + i;
               if (ii >= rdx_st[d]) and (ii <= rdx_fn[d]) then
               for n:= -1 to 1 do
                 if connected[con_typ,m,n] then
                    begin
                      jj := n + j;
                      if (jj >= rdy_st[d]) and (jj <= rdy_fn[d]) then
                         if  reg[r_rel]^[ii,jj] > 0  then
                             temp_reg[ii,jj] :=  reg[r_rel]^[ii,jj];
                    end;
             end;
      reg[r_sp]^ := temp_reg;
    end;
  End;

{-------------------------------------------------}
{-------  Sp_Activate_Line_All -------------------}
{-------------------------------------------------}
Procedure Sp_Act_Lin_All(con_typ,r_sp,r_rel,d:integer);
{Same as above, but runs until no change in r_sp}
{This is substantially faster than interpreting UNTIL_NC}
  Var
    i,j,m,n,ii,jj : integer;
  Begin
    with EXP do
      repeat
        temp_reg := EXP.reg[r_sp]^;
        for i := rdx_st[d] to rdx_fn[d] do
        for j := rdy_st[d] to rdy_fn[d] do
          if reg[r_sp]^[i,j] > 0 then
             for m:= -1 to 1 do
```

```
                begin
                  ii := m + i;
                  if (ii >= rdx_st[d]) and (ii <= rdx_fn[d]) then
                  for n:= -1 to 1 do
                    if connected[con_typ,m,n] then
                      begin
                        jj := n + j;
                        if (jj >= rdy_st[d]) and (jj <= rdy_fn[d]) then
                          if reg[r_rel]^[ii,jj] > 0  then
                             reg[r_sp]^[ii,jj] :=  reg[r_rel]^[ii,jj];
                             {note how r_sp is updated on the fly
                              within the nested array passes: trick!}
                      end;
                end;
    until Identical_Regs(temp_reg,EXP.reg[r_sp]^,d);
  End;
{-------------------------------------------------}
{-------  Unitize  -------------------------------}
{-------------------------------------------------}
Procedure Unitize(con_typ,r1,r_res,d:integer);
{very inefficiently implemented, but adequate.}
  Var i,j,m,n,tal,min_next_tal,mnt_i,mnt_j,ib,ie,jb,je  : integer;
      none_elim : boolean;
  Label A1;
  Begin
    temp_reg := EXP.reg[r1]^;
    {elim any boundary points}
    for i := rdx_st[d] to rdx_fn[d] do
      begin
        temp_reg[i,rdy_st[d]] := 0;
        temp_reg[i,rdy_fn[d]] := 0;
      end;
    for j := rdy_st[d] to rdy_fn[d] do
      begin
        temp_reg[rdx_st[d],j] := 0;
        temp_reg[rdx_fn[d],j] := 0;
      end;
    ib := rdx_st[d]+1;
    ie := rdx_fn[d]-1;
    jb := rdy_st[d]+1;
    je := rdy_fn[d]-1;
    A1: {eliminate all points with only 1 neighbor}
    Repeat
      none_elim := true;
      for i := ib to ie do
      for j := jb to je do
        if temp_reg[i,j] > 0 then
           begin
             tal := 0;
             for m:= -1 to 1 do
             for n:= -1 to 1 do
               if connected[con_typ,m,n] then
                  if temp_reg[i+m,j+n] > 0 then tal := tal + 1;
             if tal = 1 then {delete}
                begin
                  temp_reg[i,j] := 0;
                  none_elim := false;
                end;
           end;
    Until none_elim;
    {- eliminate one point with min next higher # of neighbors -}
    i := ib;
```

```
    none_elim := true;
    min_next_tal := 10;
    while (i<=ie) and none_elim do
      begin
        j := jb;
        while (j<=je) and none_elim do
          begin
          if temp_reg[i,j] > 0 then
            begin
              tal := 0;
              for m:= -1 to 1 do
              for n:= -1 to 1 do
                if connected[con_typ,m,n] then
                    if temp_reg[i+m,j+n] > 0 then tal := tal + 1;
              if tal = 2 then
                 begin
                   temp_reg[i,j] := 0;
                   none_elim := false;
                 end
               else
                 begin
                   if tal > 2 then
                      if tal < min_next_tal then
                         begin
                           min_next_tal := tal;
                           mnt_i := i;
                           mnt_j := j;
                         end;
                 end;
            end;
            j := j + 1;
          end;
        i := i + 1;
      end;
    if not none_elim then
       goto A1
    else
       if min_next_tal < 10 then
          begin {go and elim a higher n_touch}
            temp_reg[mnt_i,mnt_j] := 0;
            goto A1;
          end;
    EXP.reg[r_res]^ := temp_reg;
  End;
```

A.5 The Pascal Program that Builds the 3x3 Masks

```
Program MASK_Builder;
Const
  pi_by_8       = 0.392699;
Type
  mask_3x3_array = array[0..3,0..7,-1..1,-1..1] of integer;
Var
  a,i,j           : integer;
  mask_3x3        : mask_3x3_array;
  sigma,sigsq     : array[1..25] of real; {.2 increments of sigma 1=>.2}
  x,y,t,
  temp,temp2,
  temp3,temp4,
  theta,result    : real;
```

```
  orient,dist   : array[-1..1,-1..1] of real;
{-------------------------------------------------}
{           P  R  O  C  E  D  U  R  E  S          }
{-------------------------------------------------}
{-------  arctan_MR  ----------------------------}
Function  arctan_MR(y,x:real): real;
  Const
      pi_by_2 = 1.5707963;
      pi      = 3.1415927;
  Var a: real;
  Begin
    if x = 0.0 then
      begin
        if y > 0 then  a :=  pi_by_2
                 else  a := -pi_by_2;
      end
    else a := arctan(y/x);
    if x < 0.0 then arctan_MR := a + pi
               else arctan_MR := a;
  End;
{-------  Gauss_1D  -----------------------------}
Function  Gauss_1D(x,sigma_sqrd:real): real;
  Const
    two_pi = 6.2831854;
  Begin
    Gauss_1D := exp(sqr(x)/(-2 * sigma_sqrd)) /
                sqrt(two_pi * sigma_sqrd);
  End;
{-------  Gauss_2D  -----------------------------}
Function  Gauss_2D(x,y,sigma_sqrd:real): real;
{NB. unlike the 1D case, the constant factor has been removed}
  Begin
    Gauss_2D := exp((sqr(x) + sqr(y))/(-2 * sigma_sqrd));
  End;
{-------------------------------------------------}
{                  M    A    I    N               }
{-------------------------------------------------}
BEGIN
  Clrscr;
  for i := -1 to 1 do
  for j := -1 to 1 do
   begin
     orient[i,j] := arctan_MR(j,i);
     dist[i,j]   := sqrt(i*i + j*j);
   end;
   for i := 1 to 25 do
     begin
       sigma[i] := i/5;
       sigsq[i] := sqr(sigma[i]);
     end;
  {--------------------------------}
  {--    Construct 3x3 Masks      --}
  {--------------------------------}
  for a :=  0 to 7 do  {for each orientation 1..8}
    begin
      t := pi_by_8 * a;
      for i := -1 to 1 do
      for j := -1 to 1 do
        begin
          theta := orient[i,j] - t;
          x := cos(theta) * dist[i,j];
          y := sin(theta) * dist[i,j];
          {---  0: LINE : oriented DOG * 2-D gaussian}
```

```
            temp   := Gauss_1D(x,sigsq[3]) - Gauss_1D(x,sigsq[5]);
            temp2  := 400  * Gauss_2D(x,y,sigsq[7]);
            result := temp * temp2;
            mask_3x3[0,a,i,j] := Round(result);
            {---  1: GROWa: 1_D Gaussian * directional step fn -->}
            temp3  := 150 * Gauss_1D(x,sigsq[1]);
            result := temp3 * (1.4 * y - 1);
            if result < -300 then result := -300;
            mask_3x3[1,a,i,j] := Round(result);
            {---  2: GROWb: 1_D Gaussian * directional step fn <--}
            result := temp3 * (-1.4 * y - 1);
            if result < -300 then result := -300;
            mask_3x3[2,a,i,j] := Round(result);
            {---  3: ENDPT: simple Gaussian with constant removed;
                            has + spike and - surround}
            result := 100 * (Gauss_2D(x,y,sigsq[1]) - 0.4);
            mask_3x3[3,a,i,j] := Round(result);
          end;
    end;
End.
```