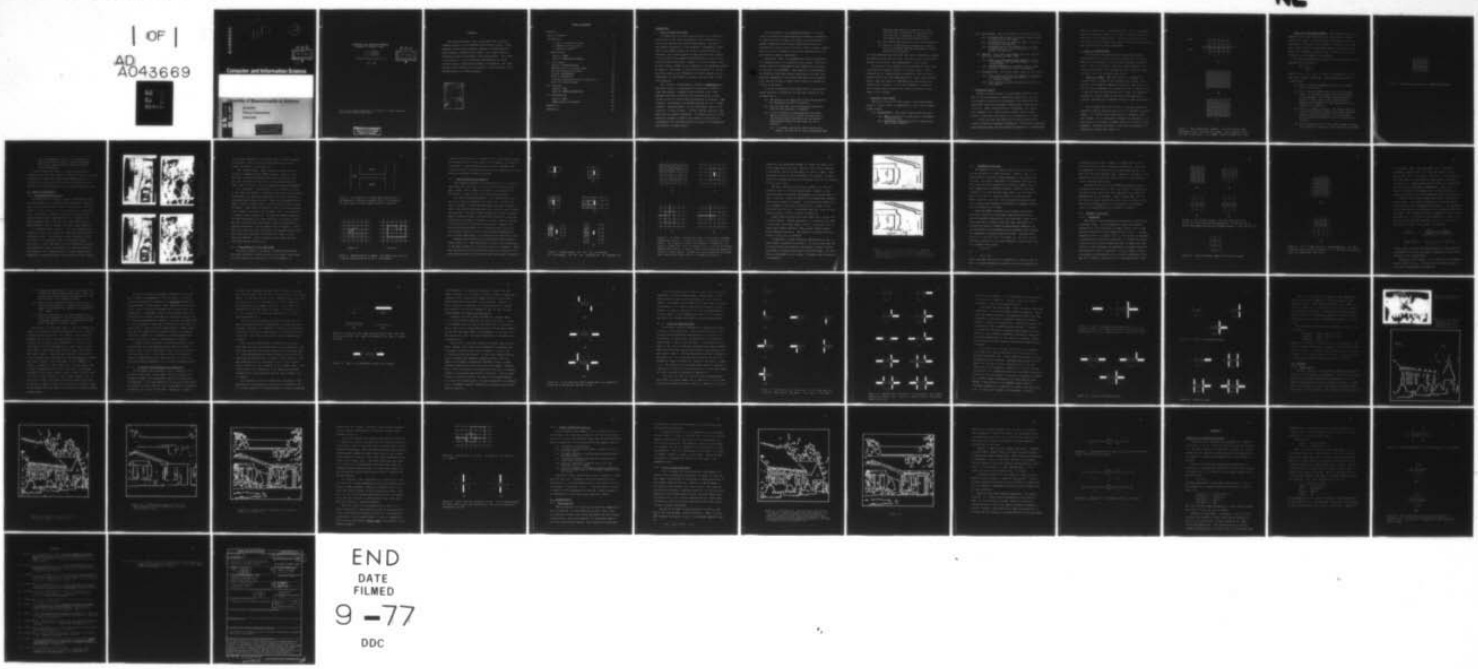
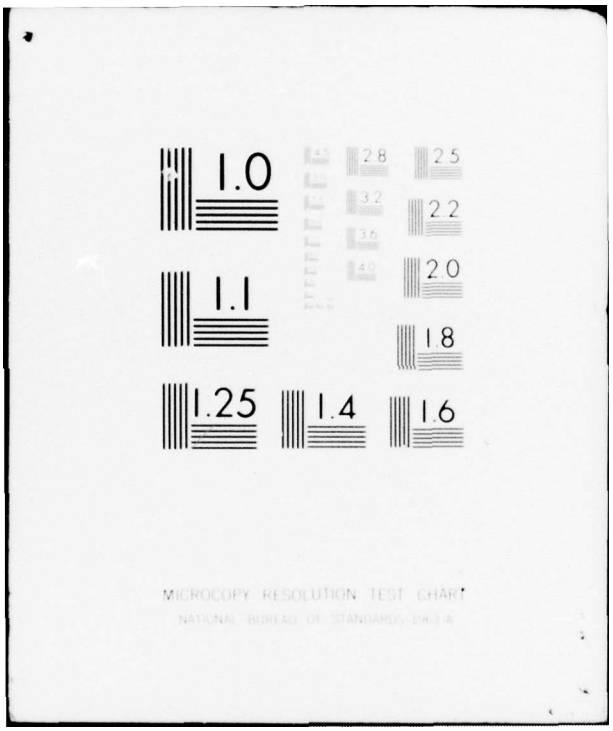
AD-A043 669 MASSACHUSETTS UNIV AMHERST DEPT OF COMPUTER AND INF--ETC F/G 6/4
EXTRACTING AND LABELLING BOUNDARY SEGMENTS IN NATURAL SCENES.(U)
MAY 77 J M PRAGER, A R HANSON, E M RISEMAN N00014-75-C-0459

UNCLASSIFIED COINS-TR-77-7 NL

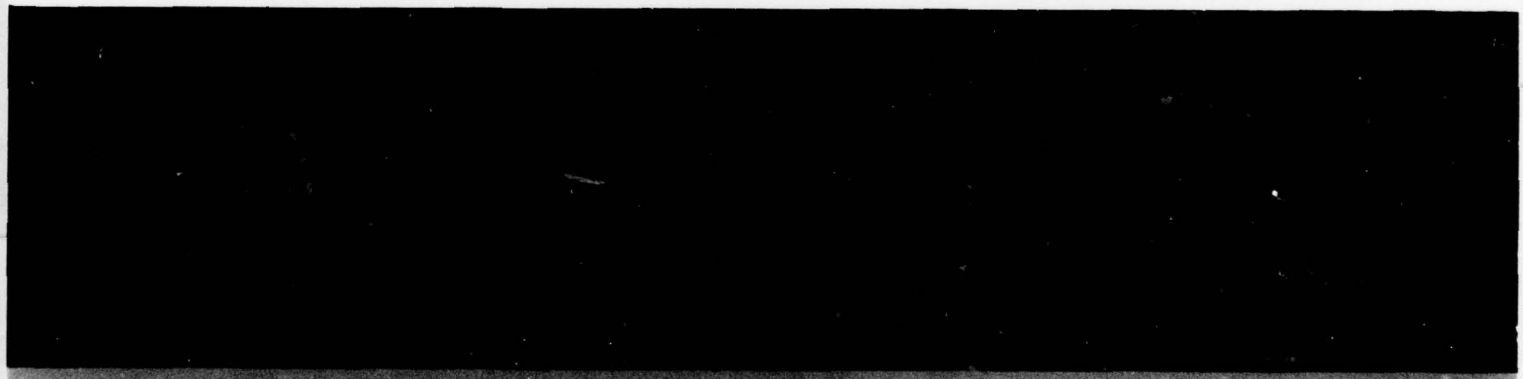
| OF |
AD
A043669



END
DATE
FILMED
9 -77
DDC


1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

Computer and Information Science



University of Massachusetts at Amherst

Computers

Theory of Computation

Cybernetics

EXTRACTING AND LABELLING BOUNDARY SEGMENTS IN NATURAL SCENES

J. M. Prager
A. R. Hanson
E. M. Riseman

COINS Technical Report 77-7

May 1977

This research was supported by the Office of Naval Research under Grant N00014-75-C-0459.

ABSTRACT

This paper describes a set of programs used to perform boundary-analysis in the VISIONS scene-analysis system. These programs lead the data through a sequence of transformations: preprocessing, differentiation using a very simple operator, relaxation using case-analysis, and postprocessing. The output of the system is a set of labelled line-segments for which features such as length and confidence are computed. The lines and associated features will be passed to other portions of the VISIONS system for further analysis.

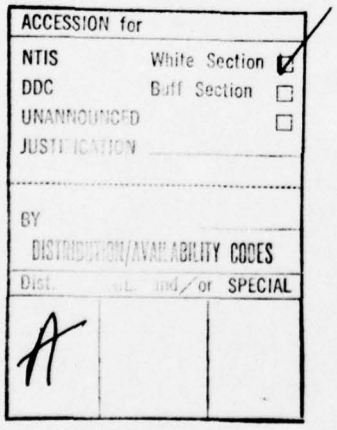


TABLE OF CONTENTS

INTRODUCTION

1. Goals of Low-Level Vision

A primary goal of visual scene analysis is to extract a description of the scene in question, and in particular, to isolate and describe the objects which appear. Because of the enormous complexity of the problem, a comprehensive analysis of any complex scene will involve a large system [1]; otherwise only certain aspects of the problem can be tackled. It is inevitable that semantic knowledge must be used to perform this task effectively. However, it is by no means clear at what stage or stages of the computation this knowledge should be applied. This paper is written in the context of a scene analysis system called "VISIONS" [1,2] which consists of two subsystems, one performing a low-level and the other a high-level analysis.

The low-level system seeks to perform a segmentation of the scene, that is, a description of the scene in terms of lines and regions. Boundary (also referred to as "line") analyses have been concerned with discontinuities in some feature or set of features such as measures of textures, light intensity, and distance (determined through the use of range-finders). Region-analyses, on the other hand, look for continuities in these features. Neither analysis typically uses high-level or semantic knowledge. It would be the job of the high-level system to identify the objects in the scene by interpreting the lines and regions as parts of the boundaries and surfaces of these objects.

For this phase of our system development it is exclusively the high-level system which uses semantic information. It should be noted, though, that there are systems which integrate segmentation with object identification [3,4,5]. Ultimately, we would like to provide a couple of major feedback paths to direct refinement of an initial segmentation by the more abstract semantic hypotheses.

This paper, then, is concerned only with the nonsemantic line analysis part of a low-level vision system. The visual feature that will be employed in all examples in this paper is intensity, or brightness. The scene will be presented to the system as three data arrays consisting of the red, blue, and green components of the scene digitized on a rectangular grid, and the average of these three values will define the feature of intensity.

It is the contention of this paper that a line-analysis system should be constructed in a way that conforms to the following rules:

(1) The nature of the input data to the system should be determined as fully as possible.

(2) The problems that beset line-analyses should be laid out and isolated, if possible.

(3) A set of modules or transformations which when applied in series will convert the input data to the desired output form should be constructed. These modules should be such that, as far as possible,

   (a) it is known exactly how they process their inputs. In this way it may be determined what

preprocessing steps are required to convert the data into an effective (or optimal) form for their use. Properties of the inputs of the modules may also be determined easily.

(b) The modules each perform a single transformation.

(c) There is limited coupling between the modules. This will restrict the size and number of intermediate data structures.

If these guidelines are followed, the flow of information through the system will be easy to trace. This will lead to easier debugging, modification, and comprehension of the system.

We describe in this paper a set of programs that are used to perform the boundary analysis of outdoor scenes. These programs are all computationally inexpensive; they are small and are fairly fast (see Implementation). Due to their modular nature, they can easily be "unplugged" and replaced by more sophisticated versions, or left out altogether, if desired. This is made possible because the state of the data at each stage is well-defined.

2. Overview of the System

There are conceptually four stages to the line-finding process. Each of these is implemented as one or more computational modules.

(1) PREPROCESSING: This stage cleans up the raw data.

(1a) UMMIX corrects for "mixed pixels" introduced in digitization.

(1b) CONDITIONAL AVERAGE smooths out random noise and fine microtexture.

(2) LINE-FINDING: This is the heart of the whole process.

(2a) DIFFERENTIATION finds the apparent edge-strength at each point in the image.

(2b) SUPPRESSION removes "multiple edges" formed on differentiating gradients.

(2c) RELAXATION drives the probability of an edge at each point to 1 or 0 on the basis of local support or inhibition.

(3) GROUPING: This stage joins edges into line segments and finds features of these lines.

(3a) BIND joins contiguous edges together to form line-segments, and each line segment is given a unique label.

(3b) FEATURE EXTRACTION of features such as length, contrast, location for each line-segment.

(4) POSTPROCESSING: This final stage cleans up the results.

(4a) TRIM removes selected line segments on the basis of features extracted previously (e.g. short, low contrast lines).

(4b) Many other more global organizing processes are now possible.

3. Tuning the System

Tuning a complex system is a notoriously difficult task. Even if the algorithms are regarded as fixed and it is only the parameters which are to be adjusted, the interactions between them, especially nonlinear interactions, cause many difficulties. These difficulties are considerable in a system with feedback. In a serial system, though, they are much easier to deal with since there is no "circularity of effect."

It will be shown that there are not many parameters involved in the above sequence of processes. It has been found that those which do exist can very easily be adjusted to a satisfactory

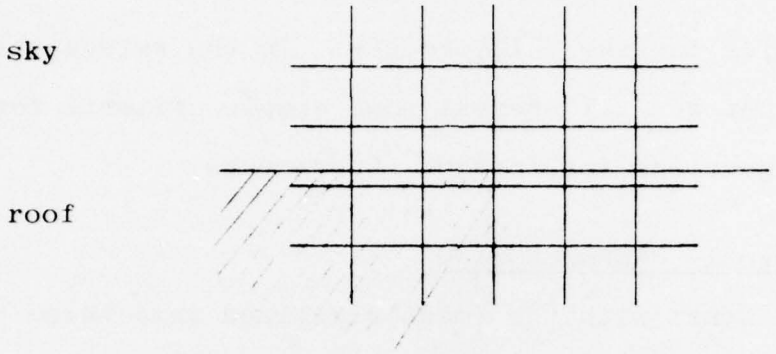
value, so tuning is not a great difficulty. One of the newer and more interesting tasks in this area is the development of heuristics for the TRIM process. In the relevant section of this paper we will present some simple criteria for line-removal and suggestions for further experiments.

I. Stage 1--PREPROCESSING

We start with the image digitized into three arrays containing the red, green, and blue components of the scene. These are now averaged to form the black and white intensity image. Prior to differentiation, this image undergoes two preprocessing stages which provide the black and white image upon which the line-finding process works.

(Step 1a) UNMIX: The first process is designed to eliminate what is known as the "mixed-pixel" problem. This problem occurs whenever images are digitized, and is due to the fact that boundaries in the image will not in general fall in register with the digitization grid. Thus, the intensity recorded at a pixel might overlap two regions adjoining a boundary, representing a weighted average of them (see Figure 1).

The procedure must test to see if a two-step intensity gradient occurs at the same place in all of the three colored images. If it does, then a mixed pixel is assumed to have been formed. It is consequently "unmixed" by assigning to it the values of its nearest neighbor along the direction of the gradient. This has the effect of shifting the boundary by a fraction of a pixel (see Figure 1c).



1a

45	45	45	45	45	45
45	45	45	45	45	45
35	35	35	35	35	35
20	20	20	20	20	20
20	20	20	20	20	20

1b

45	45	45	45	45	45
45	45	45	45	45	45
45	45	45	45	45	45
20	20	20	20	20	20
20	20	20	20	20	20

1c

Figure 1: The "mixed-pixel" problem. 1(a) digitization grid superimposed upon a portion of an image. 1(b) intensity values recorded in this grid. 1(c) "UNMIX" correction applied to 1(b).

(Step 1b) CONDITIONAL AVERAGE: The second process is an adaptation of a smoothing process [6] which helps eliminate noise in the image. In this process, the intensity value at each point is replaced by the average of itself and its neighbors, except that if the difference between the value of the point and a neighbor is greater than a certain value T that neighbor is not included in the average.

For the neighborhood $\{N_i\}$ of the point N_0 in Figure 2, its updated value is given by:

$$N_0' = \frac{1}{n} \sum_{N_i \in S} N_i$$

where $S = \{N_i : |N_i - N_0| < T\}$ and n is the cardinality of S. Note that S always contains N_0. This procedure has the following effects:

(1) Within a (nearly) homogeneous region, it smooths out the small noise.

(2) Near a region boundary whose contrast is greater than T it includes no points across the boundary in the average. This allows a smoothing of the points on either side of the boundary without blurring the boundary as a nondiscriminatory averaging process would do.

(3) Within an intensity gradient, the process averages a point with roughly as many other points that have smaller intensity as greater. This will smooth noise within the gradient but will not destroy the gradient.

(4) In a textured region, if the texture elements differ only slightly in intensity, they they will be smoothed

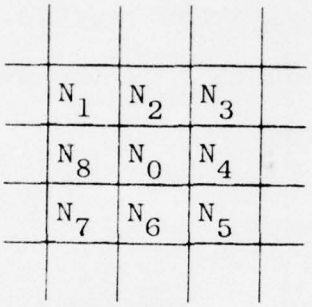


Figure 2: Neighborhood considered in CONDITIONAL AVERAGE.

into a homogeneous region. If the texture elements differ by more than T, then no averaging will occur, except perhaps within the texture elements themselves.

Figure 3 shows the results of using the differential operator (to be described later) on images that have selectively undergone the UNMIX and/or CONDITIONAL AVERAGE passes. It is seen that an application of both gives the cleanest results without losing any important lines, or gaining extraneous ones.

II. Stage 2--LINE FINDING

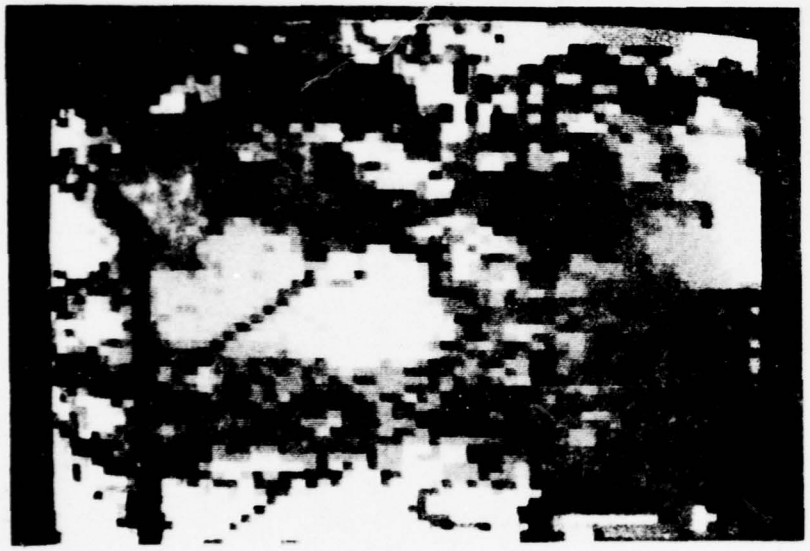
II.1. DIFFERENTIATION (Step 2a)

Differentiation is the most drastic transformation that the data undergoes, so it needs careful attention. Ideally, edges should be placed only between regions that differ with respect to some feature (in our case intensity), and nowhere else. In practice, problems occur due to texture within a region, blurred edges, and gradients, etc. However, our simple preprocessing of the data will reduce the impact of these problems. Let us consider the three cases separately.

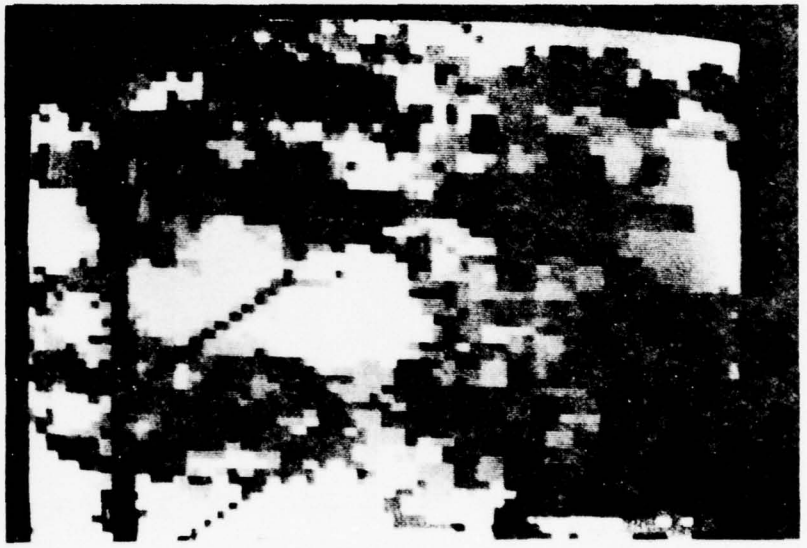
(1) Texture within a region. Fine low contrast microtexture will have been largely eliminated by the conditional averaging process. Very distinct texture elements of high contrast will be prominent, and so will produce edges. At this point it is not the task of the differentiator to determine whether the edges are boundaries of texture elements or the boundaries of a textured region. Texture edges may be removed by a subsequent process which eliminates short and/or

3a

3b



3c



3d

Figure 3. Results of preprocessing house scene and tree trunk foliage scene. Figures 3a and 3c show the original intensity data. Figures 3b and 3d show the corresponding data after undergoing the UNMIX and CONDITIONAL AVERAGE steps. Unfortunately, the quality of the reproductions does not show the full extent of the improvements.

low contrast boundaries, or one that detects texture patterns if necessary. This will be performed when more reliable global information is available [7].

(2) Blurred edges. Many of these will have been corrected (or reduced) by the "UNMIX" process. Some of those that were introduced through noise or some other means and were not corrected will give rise to two adjacent parallel edges; one of the pair will be eliminated through SUPPRESSION.

(3) Gradients. This problem is a more general version of (2), where the change in intensity occurs over several pixels. One procedure for detecting gradients is to use a hierarchy of increasing-sized masks [3,7,8,9]. While this procedure can be shown to work in simple cases, it is difficult to make it work in general. Many masks of different sizes at varying distances from a boundary can detect that boundary, and it is difficult to organize them consistently. In addition genuine gradients are indistinguishable in a digitized image from three or more parallel one-pixel-wide regions with intensity monotonically varying across them. For an example refer to Figure 4. Since there is no way that this distinction can be made without using very high-level knowledge, the system will treat all such cases as gradients. We will accept the fact that such one-pixel-wide regions will be lost.

II.2. Representation of the Edge Image

The input consists of an array of numbers representing the light intensity of each position in the image. Since each of these pixels is in some region, it is reasonable to

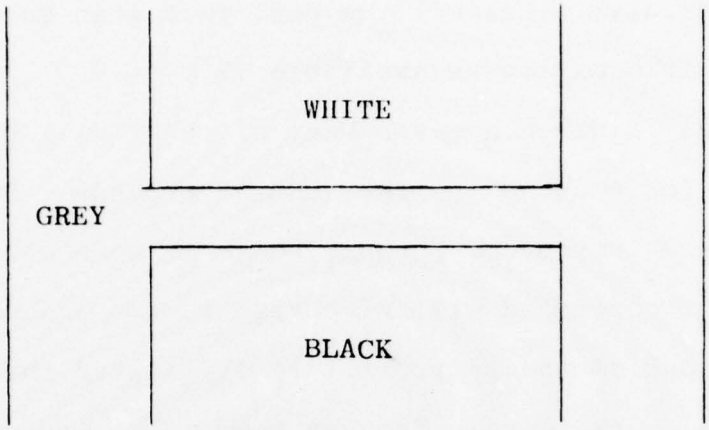


Figure 4: An example of a 1-pixel-wide region that is intermediate in intensity between that of its neighbors. The grey strip is a real region, and not a gradient.

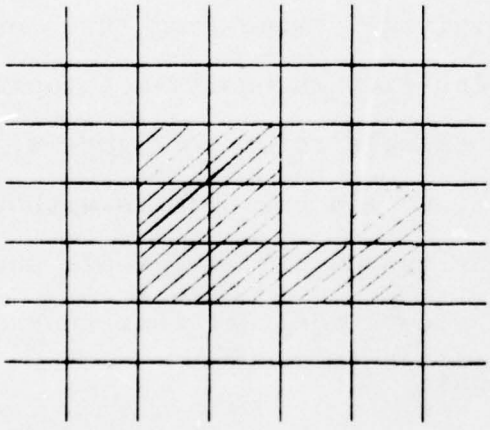


Figure 5a

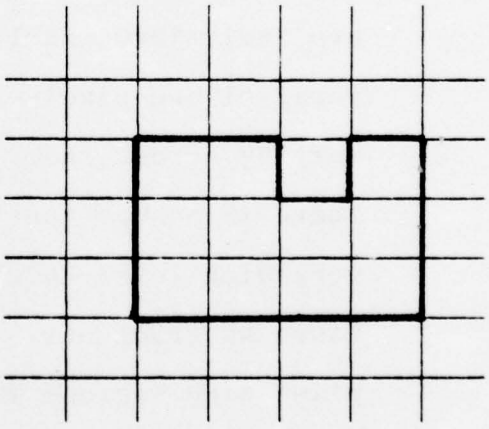


Figure 5b

Figure 5: Representation of edges. The shaded area in 5a is a region, and the outline in 5b is its boundary.

constrain the boundaries of regions to fall only between pixels. This representation of the image on a rectangular grid and the constraints of edges between pixels imposes a boundary that consists entirely of horizontal and vertical edges [10,11]. This greatly facilitates further processing.

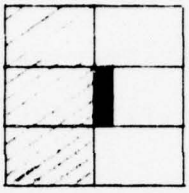
II.3. The Differentiation Operator

The standard technique for differentiation is to convolve edge masks with the image. It can be generalized to apply a set of masks, and to compute the output as some function of the results of these masks, often the maximum response.

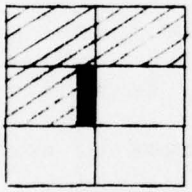
For sharply defined boundaries, the simplest mask possible is all that is necessary (see Figure 6a). We will call this a 1×2 mask. In this and subsequent diagrams of masks, a heavy line indicates the edge position to which a mask's output is associated. On long straight boundaries a better response might be achieved using a 3×2 mask (see Figure 6b), since the information from three 1×2 masks in a line is used to average out the presence of occasional noise points. However, the relaxation processes that follow should be able to fill in such an edge, and we believe it will render the benefits of the 3×2 mask unnecessary; its limitations are described below.

Diagonals and corners can be detected by using diagonal masks (Figure 6c). This mask might be used in the computation of the edge strength of a vertical section of a diagonal. Note that application of this mask alone would give a positive response when applied to a horizontal edge (see Figure 7a and 7b).

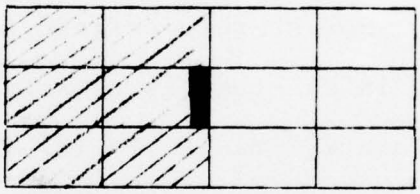
6a



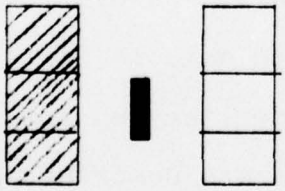
6b



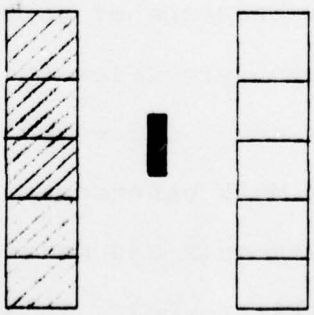
6c



6d

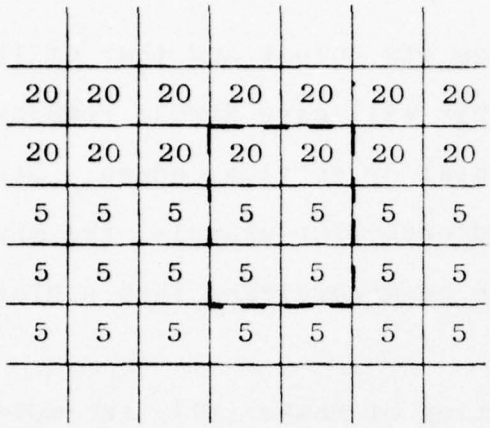


6e

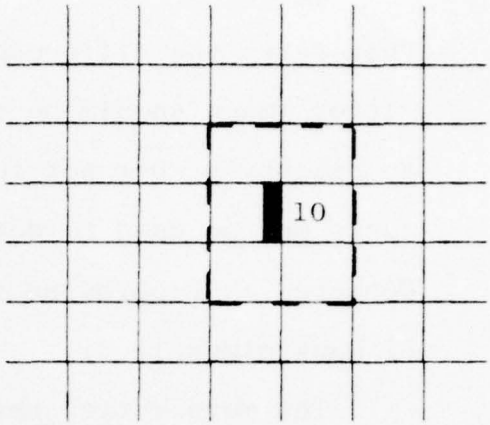


6f

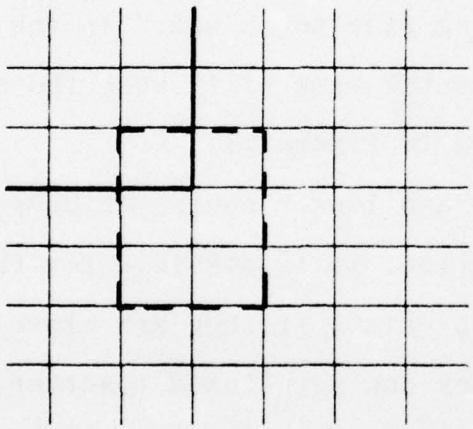
Figure 6: Typical masks. 6a: 1×2, 6b: 3×2 straight, 6c: 3×2 diagonal, 6d: 3×4, 6e: expanded 3×2, 6f: expanded 5×2.



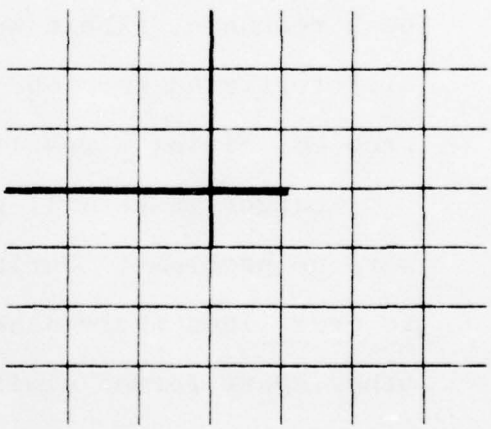
7a



7b



7c



7d

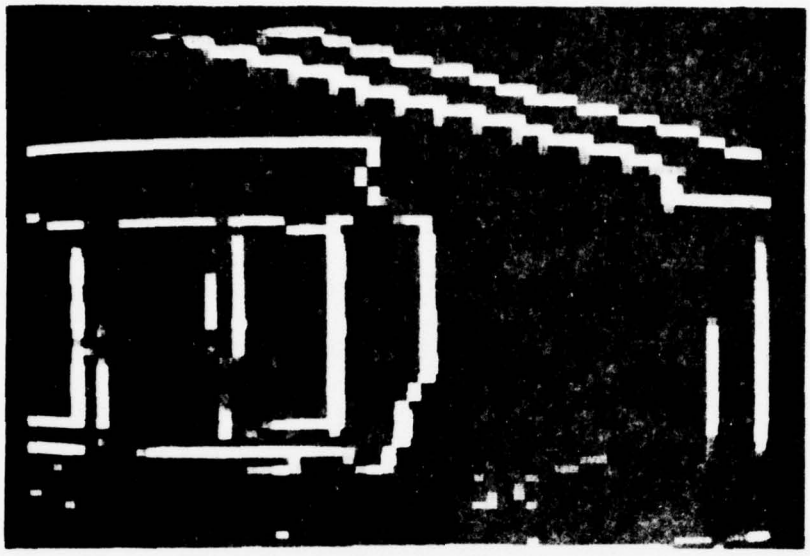
Figure 7: 7a shows a 3×2 diagonal mask on a horizontal boundary between two regions. 7b shows the output of this mask (if used above). The response is seen to be (20+20+5 - (5+5+5))/3 = 10, which is significant, since it happens to be 2/3 of the difference between the regions. 7c shows a 3×2 straight mask overlapping the corner of a dark region. 7d shows the spur produced by this mask and a smiliar horizontal mask in a neighboring position.

Therefore, the difference between its output and that of its mirror image should be used. This will give strong response to diagonals, but not to horizontal or vertical edges. Other masks may be used to detect gradients; for example, the masks depicted in Figures 6d to 6f can be generalized into a hierarchy of mask sizes [7,8].

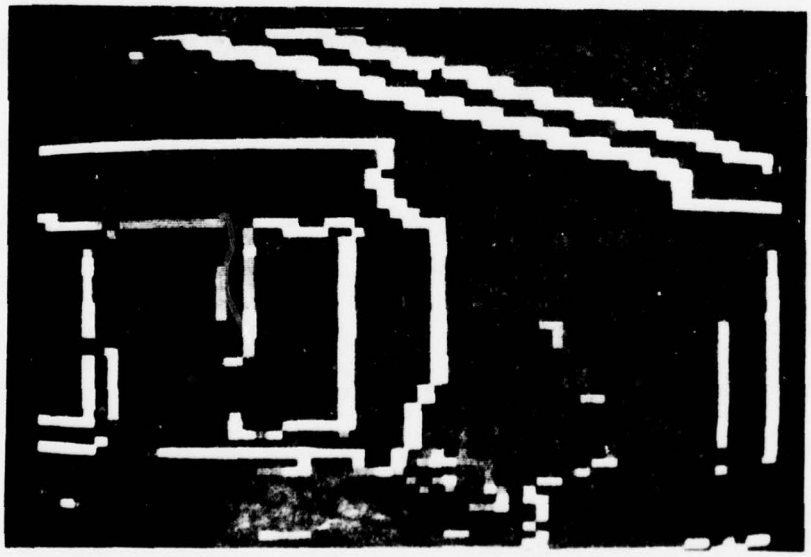
The more varied the collection of masks [11], the more guarantee there is of detecting the edge. However, using large masks has unfortunate consequences in positions where no edge is desired. Figure 7c shows a 3×2 straight mask superimposed upon a corner of a region. In this position there will be a response, albeit weak, giving rise to a "spur" in the differentiated version. A horizontal mask will cause the same problem, giving a result as shown in Figure 7d.

Larger masks will give more and longer spurs, which cause serious problems. During relaxation, it is possible for them to grow lines where none ought to exist; if they are close to other spurs formed similarly, they can get linked together during grouping processes. The results of the whole process begins to get quite ill-defined.

A comparison of these different combinations of some of these masks is presented in Figure 8. It can be seen that the 1×2 gives fairly good results; the absence of spurs is quite noticeable in contrast with some of the other masks. For this reason, we decided to use a simple 1×2 mask as our differentiation operator.



8a



8b

Figure 8. Effects of using large masks. Figure 8a shows a portion of the house scene of Figure 3b differentiated using a 3x2 mask. The spurs that are so prominent here are not produced by a 1x2 mask, as shown in Figure 8b.

II.4. SUPPRESSION (Step 2b)

The weakness of a 1×2 mask is that it will be prone to missing boundaries of wider gradients. However, most of the boundaries in the several scenes examined in this paper were detected. Of course, the problem of gradients still must be dealt with since the system will be blind to edges, such as wide shadows on a cylindrical surface. Ideally, the total strength of the wide gradient edge ought to be collected [3], which is the goal of employing masks of increasing size. Rather than deal with some of the problems discussed in the last section, here instead we seek means to suppress multiple parallel indications of edges.

While the UNMIX procedure will eliminate some narrow gradients, others will inevitably remain and give rise to parallel multiple indications of the same edge. These can be removed by what is known as multiple edge suppression [3]. Consider the image in Figure 9a representing brightness and its derivative in 9b representing the strength of the gradient. The suppression technique works as follows: Consider three pixels in a vertical line as in Figure 10. Let p,q,r be the horizontal gradient at the lower boundary of these three cells. If it happens that either

$$|q| \leq |r|$$

or

$$|q| < |p|$$

with q the same sign as r or p respectively, then q is set to zero. A similar operation is applied to vertical gradients in

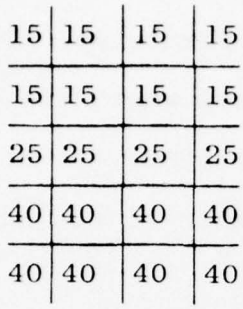
a horizontal row of cells. Hence, in Figure 9b the row of 10's will be set to zero, resulting in Figure 9c. This type of suppression is restricted to the cases where the pair of edges have the same gradient sign. Therefore, in Figure 11 the suppression process does not remove either of the boundaries of the one-pixel-wide region.

An improved version of the SUPPRESS procedure requires an increase in the values of the local maxima of gradients by the sum of those values that were suppressed in a direction perpendicular to the gradient. Thus in Figure 9b, the 10's will be set to zero, and the 15's will be set to 25 as in 9d. This more accurately reflects the strength of the boundary since it is between regions of intensity 15 and 40.

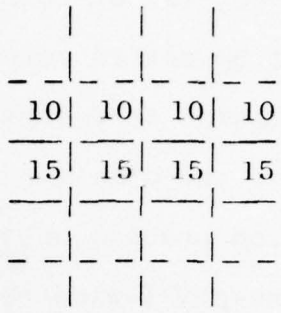
II.5. RELAXATION (Step 2c)

II.5.1. Background

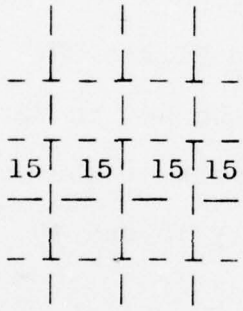
The output of the differentiation process is usually far from being clean. If the strengths of edges are viewed as probabilities of the existence of edges, usually few of them would be considered to have probabilities of 0 or 1. An edge probability that is neither 0 nor 1 is effectively an ambiguous interpretation of the entity concerned. A relaxation process allows the local context around each edge to update the probability so that ultimately the ambiguity is reduced and interpretations are locally consistent. In this scheme, a label λ is assigned to each position with an initial probability $P(\lambda)$. The set of labels Λ would be a set of edge-descriptors, such



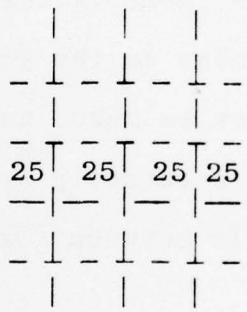
9a



9b



9c



9d

Figure 9: 9a intensity values; 9b after differentiation; 9c the top line in 9b has been suppressed; 9d the top line in 9b has been suppressed and its strength added in to the bottom line.

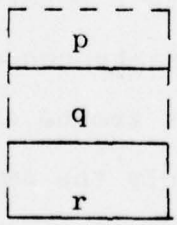
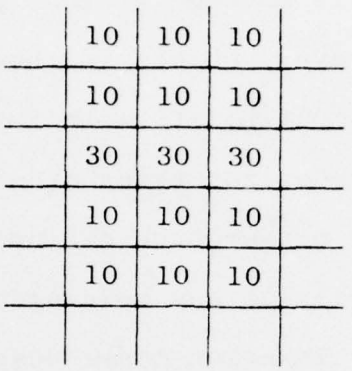
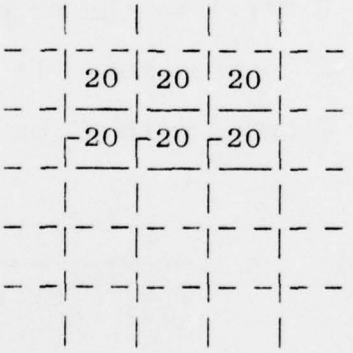


Figure 10: Three horizontal edges in a vertical column.



11a



11b

Figure 11: 11a. A light strip on a dark background. 11b. The resulting edges and their strengths. Since they are of differing signs, no suppression takes place.

as "horizontal edge," "vertical edge," etc., and usually including a special label, the "null edge" which is an assertion that there is no edge at that point. Each probability for every label on every object is then updated in parallel according to its compatibility with the labels on neighboring objects (in some predefined neighborhood). Under quite restricted conditions convergence can be guaranteed [12], although not necessarily to any meaningful global interpretation.

Consider the relaxation model described in [13] and summarized above. This kind of relaxation, for reasons noted below, will be called "homogeneous relaxation." An implementation is described in [14] and will not be treated in detail in this paper. The relaxation scheme is summarized by the following equations. Let $P^{(k)}(\lambda)$ be the probability of label λ on object a_i after the k^{th} iteration. If $r_{ij}(\lambda,\lambda')$ is the compatibility of label λ on a_i with λ' on a_jk then

$$P_i^{(k+1)}(\lambda) = \frac{P_i^{(k)}(\lambda)\,[1 + q^{(k)}(\lambda)]}{\sum_{\lambda'} P_i^{(k)}(\lambda')\,[1 + q^{(k)}(\lambda')]}$$

$$\text{where } q^{(k)}(\lambda) = \sum d_{ij}\,[\sum_{\lambda'} r_{ij}(\lambda,\lambda')\,P_j^{(k)}(\lambda')].$$

The d_{ij} serve to select the neighborhood of P_i and weight the contributions of the P_j according to distance (or some other property of the neighborhood).

The heart of the scheme is in the setting of the compatibility coefficients r_{ij}. The aim is to set these weights so that the following goals are achieved:

(1) Neighboring edges which can form a consistent line continuation should support each other. Edges lost through local noise should be brought back in-line through support from neighboring edges.

(2) Edges locally introduced through noise should be eliminated through lack of support from neighboring edges (or from inhibition through the neighboring "null" label).

(3) Multiple indications of the same boundary due to either the inaccuracies of the digitization process, or through gradients, should be eliminated through mutual inhibition of parallel edges.

The beauty of the process, then, is that it seemingly accounts for all these conditions through the use of a single formula applied iteratively. The trouble is that there are drawbacks to this kind of processing due to just that fact. Consider the case where the neighborhood of a point is the 3×3 window of points centered upon it, and a point can have three labels: horizontal edge, vertical edge, and no edge (the "null" label). There are many weights to be specified if one considers the relationship of the surrounding horizontal and vertical edges to the given edge (in fact, $\frac{8\times3\times3}{4} = 18$, subject to symmetry). More labels (e.g. diagonal edges) and a larger neighborhood give rise to many more relationships. Although relative angle between a pair of edges could be used to provide a simple measure of their mutual support [14], it will run into many cases when the relative spatial positions of the pair of edges is taken into account (i.e. relative angle is not enough unless the edges are restricted to having a common joining point).

Not only are there many weights needing to be set, but due to heavy interdependence of effects there is no direct correlation between the setting of an individual weight and the performance of the system. Thus, tuning can be very difficult, since it requires optimization of many variables simultaneously. Furthermore, there is no guarantee that it is possible to set weights such that all the desired effects can be achieved simulatneously. White it is fairly straightforward to set the weights so that some of the more obvious cases are taken care of, there is rarely enough leeway to adjust them so that the more awkward cases, such as #3 above, are managed correctly. *Indeed*, it is difficult to determine where the system is failing, or how it is achieving its results.

It appears that one source of these difficulties arises from the fact that the updating process employs a single formula that is used to take care of the various very different cases that arise. In the next section, an alternative scheme is proposed which will deal with each of the aforementioned problems separately, in a clearly structured manner.

II.5.2. A Different Representation for Relaxation

In the scheme just described, a set of labels are competing for each point in the image. Thus for a point on a diagonal boundary, both horizontal and vertical labels will be competing. In our representation, we can allow both labels to coexist at a pixel since we are placing edges at interpixel boundaries, not on top of the pixel. Therefore, at each

vertical pixel boundary the only labels we need to consider are "vertical edge" and "no edge," and similarly for horizontal edges. In this way, the set of two probabilities at each edge location $\{P_i(\lambda) | \lambda \in \Lambda\}$ can reduce to a single parameter P_i. The probability of an edge at position i is P_i, while the probability of the null label "no edge." at position i is $1 - P_i$. Relaxation is very much simplified as a result of this representation.

We will use the notation of Figure 12 to describe the edge-configurations under consideration. An open rectangle represents the edge to be updated. A dotted line represents an edge-position with no edge present, a thick solid line an actual edge, *and a* thin solid line an edge of undetermined strength.

The algorithm employed may be summarized as follows: Every edge position may be defined by its two end-points, and every end-point has three other edge-positions incident upon it. Each edge-position will have a value associated with it, indicating the probability of an edge at that position. Each edge end-point will be classified as one of four "vertex-types" according to the strengths of the incident edges. The vertex types of the end-points of the edge-position under examination will then determine how the edge-strength is to be updated.

The homogeneous form of summing the contributions independently has the advantage of being easy to formulate, but it is often difficult to comprehend what is actually happening

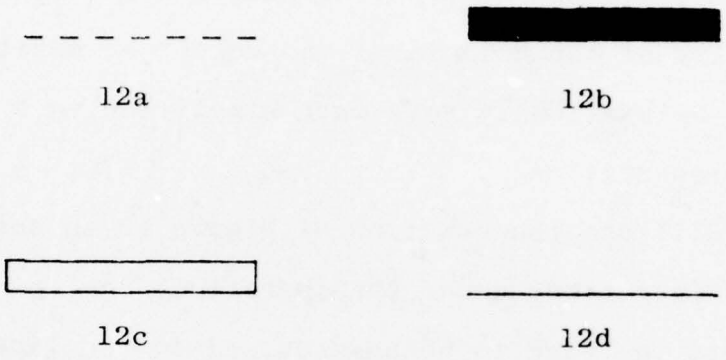


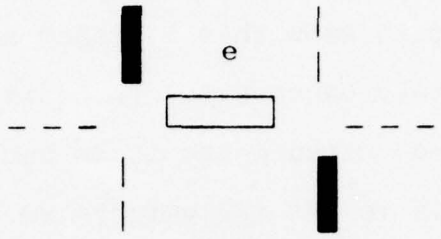
Figure 12: Notation. 12a. edge position with no edge; 12b. edge position with edge; 12c. edge to be updated; 12d. edge of unknown strength.

Figure 13: Edge to be updated has strong local support.

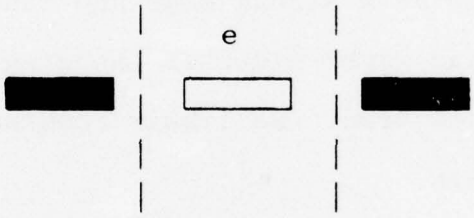
in the process. In clear-cut situations, such as the one shown in Figure 13, it is easy to figure out what is happening internally and to know that a proper setting of the weights should definitely cause Pr(e) to be increased. However, in more "cluttered" situations, it is unclear what will happen. One approach is to set the weights so that all the "obvious" cases produce the clearly desired results.

Consider the three cases in Figure 14. Suppose edges a, b, g, and h where indicated are strong, and edge e is weak. It is probably the case that in 14a and 14b it is desirable to have e come up as a strong edge and thus link the edge segments on the left and right sides of the diagrams; yet in case 14c, the opposite is true. No linear combination of weightings can accomplish this.

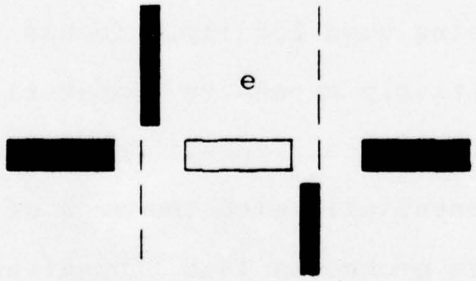
This is not to say that iterative processes should not be used. Indeed, they allow local information to propagate and, in essence, provide a wider local "window" or context than is actually being used locally. To use large windows directly can be prohibitively expensive computationally, since the number of different configurations possible within a window increases exponentially with the area of the window. Furthermore, iterative processes lend themselves very well to machines capable of processing large arrays in parallel. Such an architecture is ideally suited to the kind of processing required in a low-level vision system. So while relaxation is considered to be a worthwhile process, additional structure in the processing is required.



14a



14b



14c

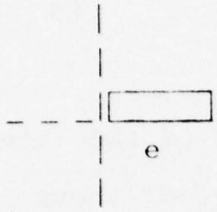
Figure 14: It is clear that edge e should have its probability increased in 14a and 14b, but not in 14c.

It may be argued that in order to take care of the interdependency of the surrounding edges, combinations of these edges should be taken as units entered into the relaxation scheme. For example, the maximum of the three edges adjoining each endpoint of e could be used in the relaxation formula. However, in order to take care of all the interdependencies, a large number of these combinations needs to be evaluated, and there still remains the problem of setting the weights.

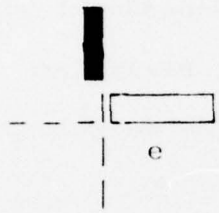
II.5.3. Cases for Updating Edges

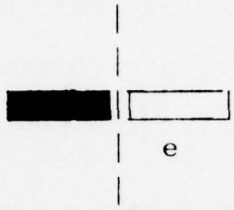
A different iterative procedure for updating the probabilities is described below. The following notation is used to depict the neighborhood characteristics (or state) of e; the symbols i-j denote that configuration i is at one end of central edge e, and j is at the other. A configuration of n edges to one side of e will be considered equivalent no matter what their positions in the three possible edge positions to that side of edge e. The four types are depicted in Figure 15. Obviously i-j $\equiv$ j-i, so we need only consider the cases 0-0 through 3-3 shown in Figure 16. These are the cases i-j where i $\leq$ j. The determination of which vertex types are present is computed as a function of the probabilities of the three edges to either side and is discussed in Appendix A.

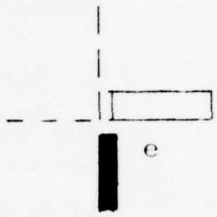
In states 0-0, 0-2, 0-3, one can quite confidently say that there is no good support for e, and in 1-1, 1-2, 1-3 one can quite confidently say that there is. However, if e is in



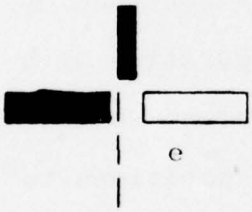
15a

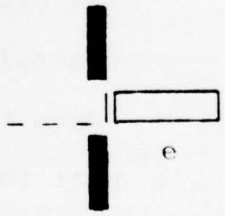




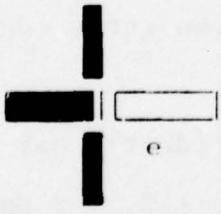


15b





15c



15d

Figure 15: Classification of "vertex-type" of left-hand end-point of edge e. 15a. Type 0; 15b. Type 1; 15c. Type 2; 15d. Type 3.

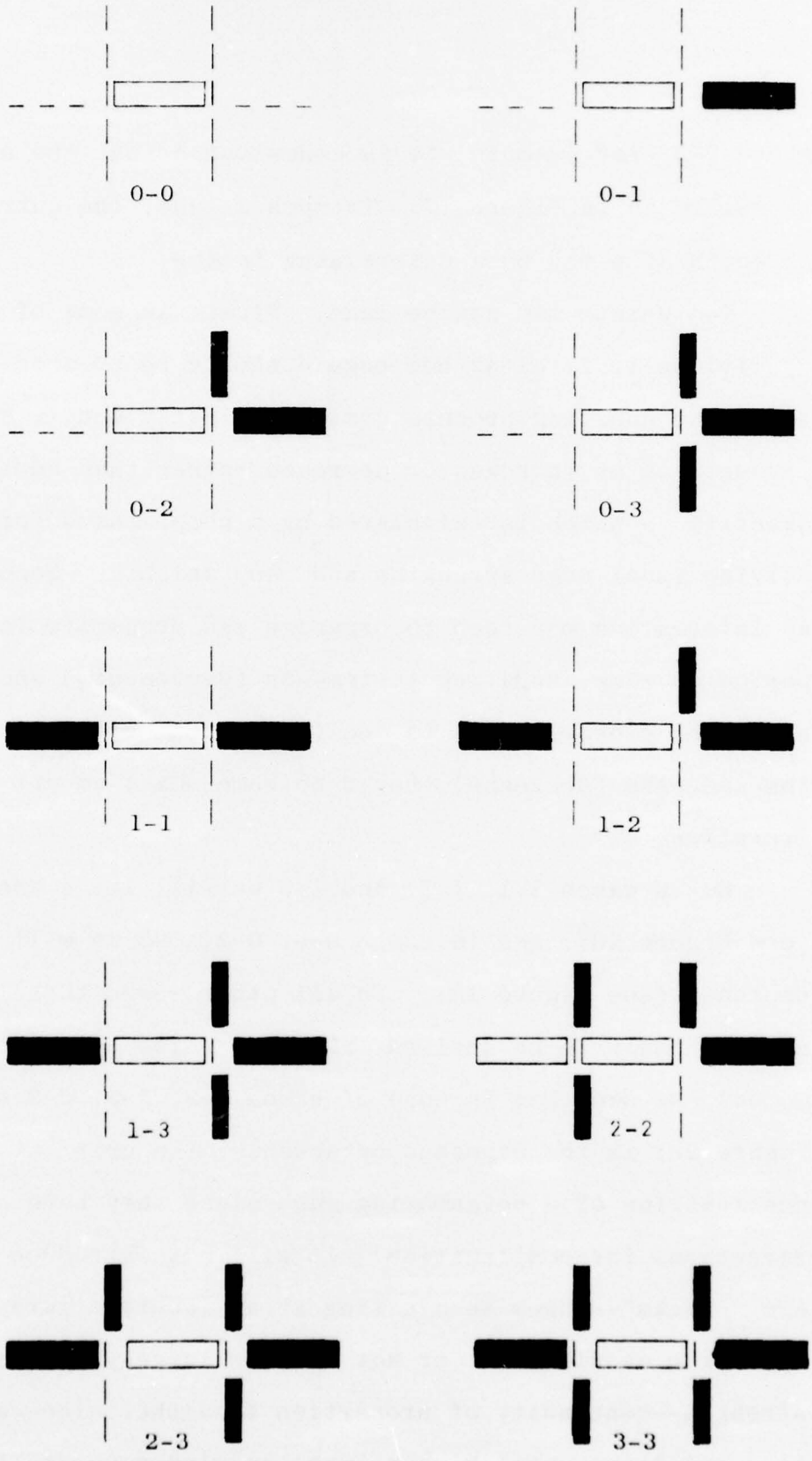


Figure 16: Representative combination of vertex types. This figure depicts all possible cases, subject to symmetry and the equivalences noted in the text.

state 0-3, for example, it is conceivable that the situation is really as in Figure 17. In such a case, the current strength of e may be a determining factor.

Two points may now be made. First, in some of the above conditions it is clear how edge e should be updated. Therefore, the updating process could explicitly modify the edge strength as an increase or decrease rather than adding a quantity e which is calculated by a complicated formula involving local edge strengths and many weights. Secondly, as information may need to organize and propagate for some period of time, updating increments (decrements) should not drive the probabilities to one (zero) too quickly. Rather, the increase (decrease) should be some small amount on each iteration.

So in cases 1-1, 1-2, and 1-3 we will let e increase (see Figure 18); and in cases 0-0, 0-2, 0-3 we will let e decrease (see Figure 19). In all other cases there is really not much help to be derived. Leaving aside case 0-1 for the moment, we see that in none of cases 2-2, 2-3, 3-3 (see Figure 20) is the presence or absence of e critical for the continuation of a neighboring edge since they have alternative directions for continuation. It will not introduce or eliminate "cracks"--edges terminating at an indeterminate point. Whether e should exist or not depends largely upon its strength--continuity of properties to either side--and little else, at least until higher level knowledge is applied.

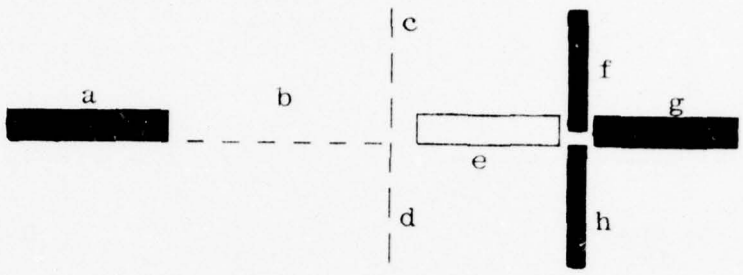


Figure 17: Edge e is classified as being type 0-3. If its strength is high, it is likely that edge a will join up with it. The desirability of this effect is not so clear if e is weak.

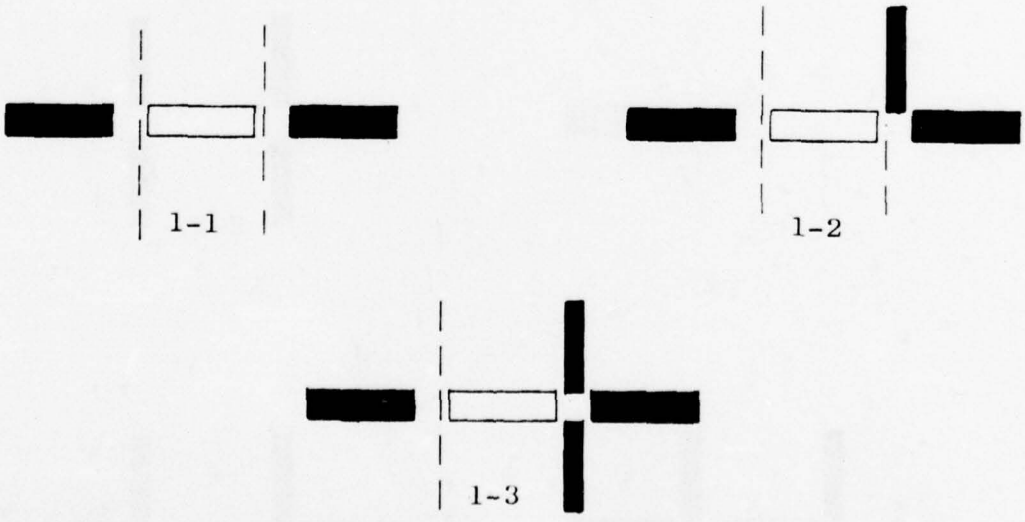


Figure 18: Cases for incrementing edge.

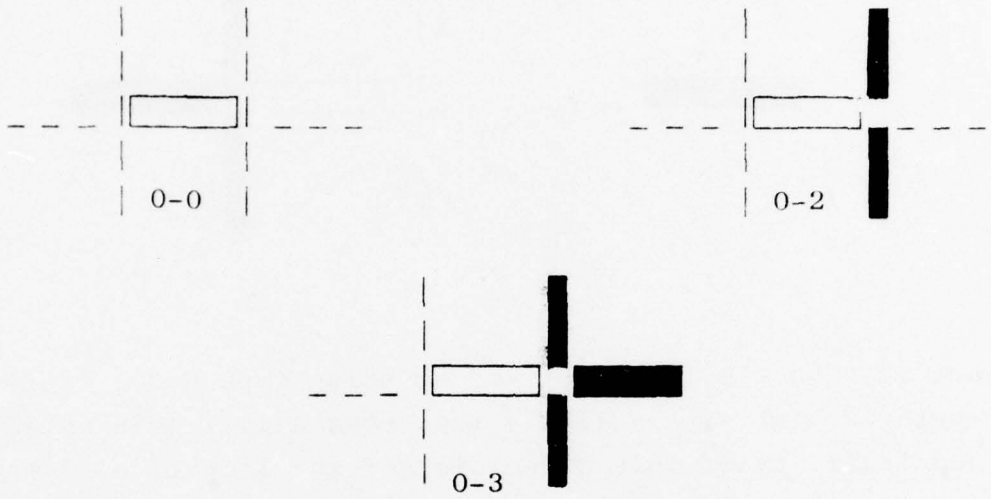


Figure 19: Cases for decrementing edge.

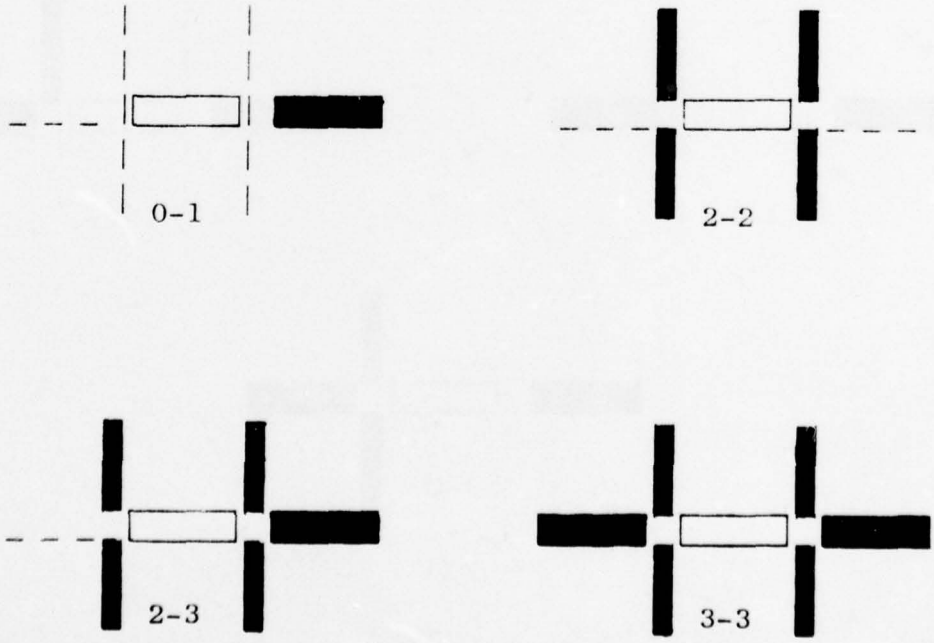


Figure 20: Uncertain cases.

Case 0-1 is really the only problem. The neighborhood on one side strongly supports e, the other suggests that e should be absent. As no sensible decision can be made, no action is taken here, or in cases 2-2, 2-3, and 3.3. This is a very important decision. It implies that in the updating process, the 0-1 case remaining constant will prevent lines from growing into noise or from being eaten away at its end-point.

The operation of the systems in updating an edge is then as follows:

Increment: $e \leftarrow \text{Min}(1, e + k)$
Decrement: $e \leftarrow \text{Max}(0, e - k)$
Uncertain: $e \leftarrow e$

where K is a constant. A large K gives fast convergence, but does not permit information to propagate very far before edges survive or decay. For small K the opposite is true. A value of about .15 to .20 was found to be suitable. Typical results of using this relaxation process are given in Figure 21.

III. GROUPING

III.1. BIND (Step 3a)

The next stage is to decide which neighboring edges will link up to form extended line segments. It is clear that those points in the current representation which have 1, 3, or 4 edges entering them are natural termination points, i.e. end-points or vertices, for these line-segments (see Figure 22). This tends to break boundaries into boundary segments which lie

Figure 21a. Intensity data after preprocessing.

Figure 21b. Differentiated version of 21a. Edge strengths have been thresholded at .25 for display purposes only.

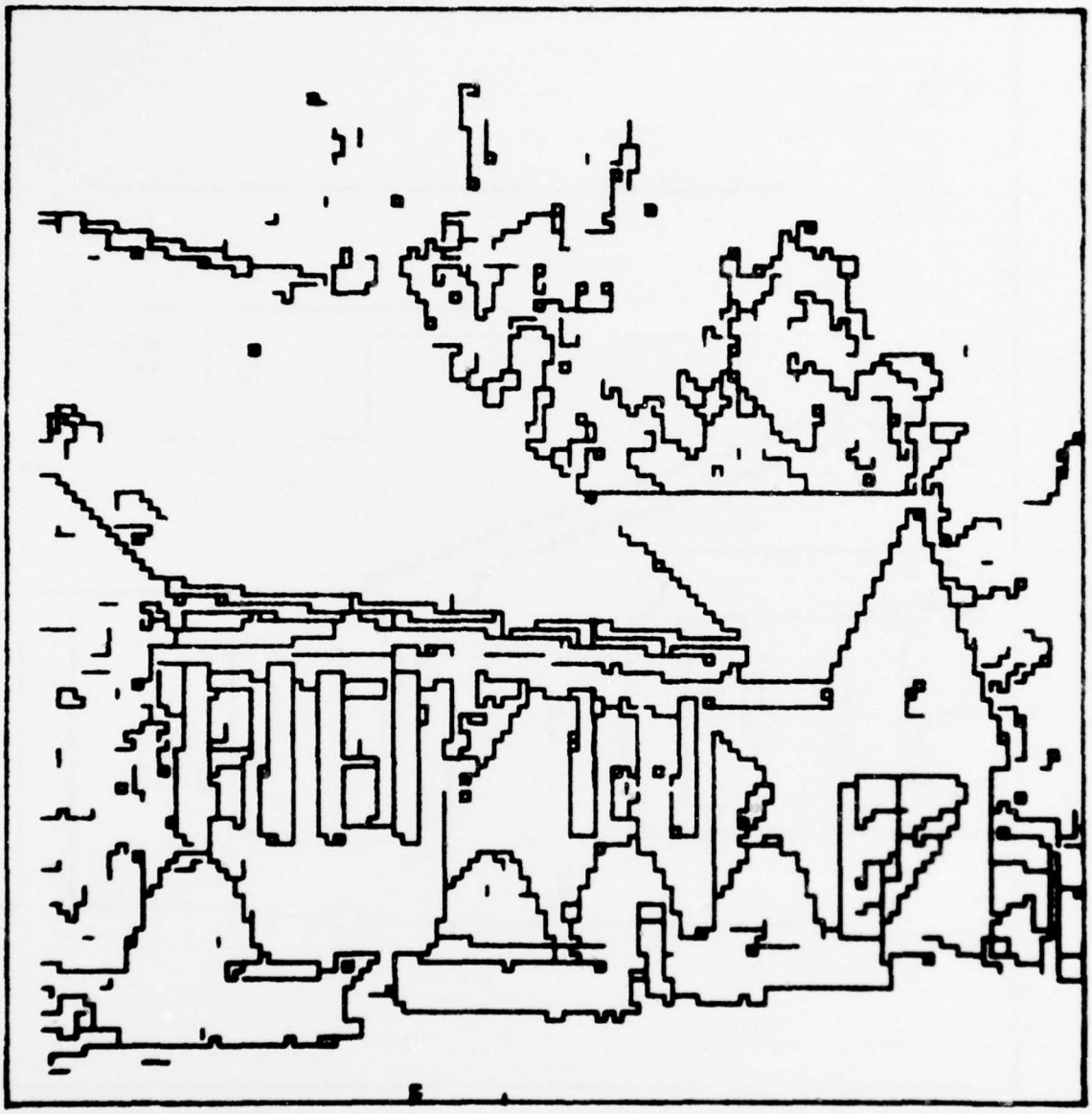


Figure 21c. Results after 5 iterations of relaxation applied to Figure 21b.

Figure 21d. Differentiated version of Figure 3b. Edge strengths have been thresholded at .25 for display purposes only.

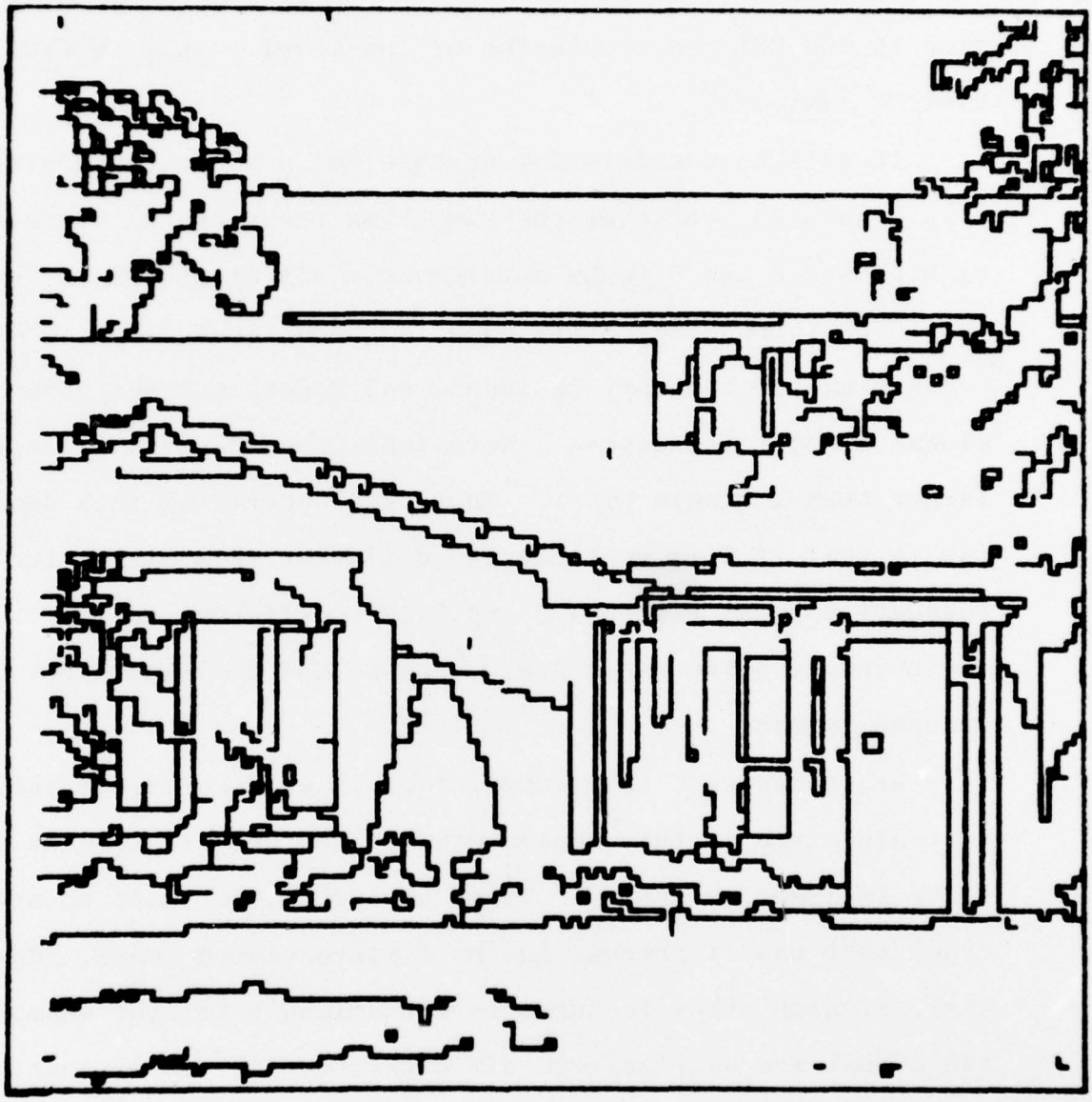


Figure 21e. Results after 5 iterations of relaxation applied to Figure 21d.

between only two regions. This was a major design consideration in the RSE representation of low-level output in the VISIONS system [1].

It will be possible for an edge to be absent at position e in Figure 23, while at the same time the features associated with points C and D to be sufficiently different that a vertex should be introduced. This case can occur when the information organizing the boundary between C and D does not get enough global support to survive. Note that C and D could be regions larger than a single pixel. The logic underlying this analysis is that if C is sufficiently different from D then the boundary between region A-B and C is a different entity from the boundary between A-B and D, and so the two should not be grouped together.

As it happens, this computation is of theoretical interest only since in this implementation the only feature we are using is light intensity. Thus, no information will be accrued that wasn't present in the differentiated image. However, if some other features were examined here, for example, those that are difficult to differentiate like color (hue), then this would be a valuable technique.

The first stage of the binding process, then, is to mark as vertices all those positions with the characteristics of the configurations in Figures 12 and 13. Following this computation, it is straightforward to track all segments between vertices and assign a unique label (line-number) to each boundary segment.

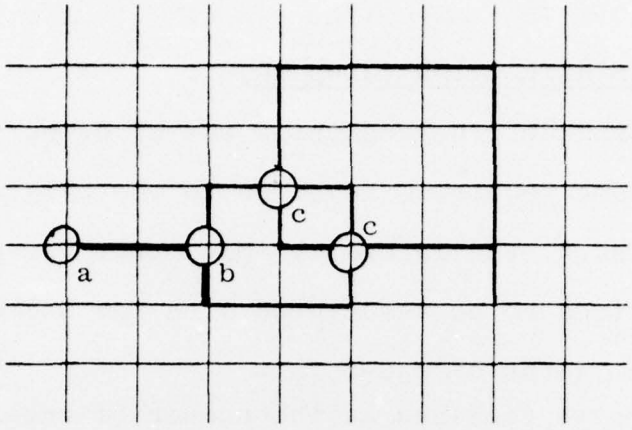
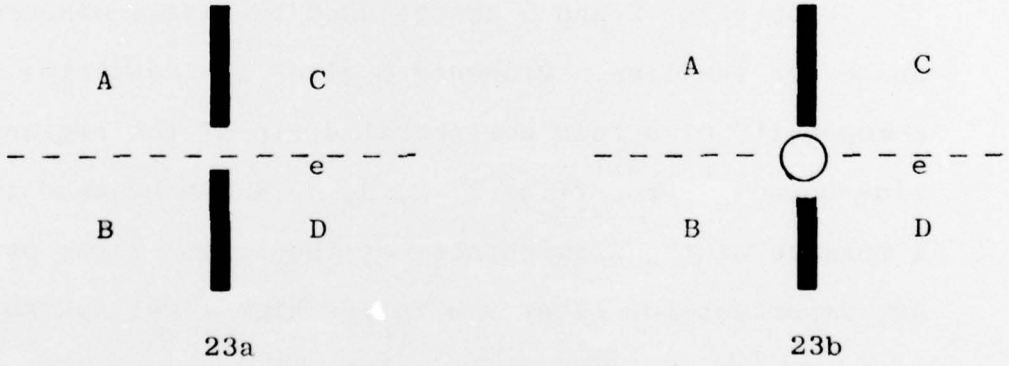


Figure 22: Three kinds of vertices. (a) Order-1; (b) Order-3; (c) Order-4.



23a 23b

Figure 23: A,B,C, and D are intensity values. If C is significantly different from D along some feature axis, then a vertex should be introduced as in 23b.

III.2. FEATURE EXTRACTION (Step 3b)

For each unique line-segment a set of properties can be established, some requiring recourse to the original intensity image, or at least, the intensity image that was differentiated. Typical properties to be associated with the segment label are:

(1) coordinates of end-points;
(2) N-length (defined as the number of edges that comprise the line);
(3) E-length (defined as the Euclidean distance between the end-points);
(4) frequency with which the edges that comprise the line change direction;
(5) mean and variance of contrast across the line, computed along its length;
(6) mean and variance of difference between neighboring points on either side of the boundary computed along its length.

Properties 2 and 5 can be used to give a measure of confidence for the line. Property 6 gives an indication of the homogeneity of a thin peripheral strip of the regions that the line bounds. Properties 1, 2, 3, 4 can be used to compute a measure of the straightness of the line. These properties are important for later use in the high-level system.

IV. POSTPROCESSING

IV.1. TRIM (Step 4a)

These techniques will clean up the low-level segmentation prior to passing it to the high-level system. While there are limitless criteria that can be developed from the features listed earlier, one of the simplest was tested and found to give very satisfactory results. This operation is described

below and is followed by a discussion of other criteria that may typically be applied.

A terminating point of a line-segment will be considered to have order 1 if there is only one edge (of the four possible edges) incident upon it; this type of line termination will be called an end-point. The criterion we use is quite simply to eliminate all edges of length 3 or less that have at least one end-point. This will then remove all the small ($\leq$ 3 units of length) "cracks" or "spurs" in the image. Figure 24 shows the result of applying this process to the output of the RELAXATION stage.

IV.2. Other Clean-Up Techniques

It may so happen that long continuous line-segments are broken by the introduction of vertices, as shown in Figure 23. This might occur because of variations in region properties on one or both sides of the line. However, if the breakage is due only to local effects (e.g. noise points), the situation can easily be remedied. Consider Figure 25. If the statistics gathered in the binding phase indicate that the regions above and below line L_1 have similar characteristics to those above and below L_2, the vertex may be removed and the lines merged to form a single line-segment.

Let $I^k_{n\mu}$ be the mean of some property k of line L_n, and $I^k_{n\sigma}$ be the corresponding standard deviation. Then, a reasonable measure for merger of a pair of contiguous segments might be

$$|I_{1\mu} - I_{2\mu}| \leq k(I_{1\sigma} + I_{2\sigma})$$

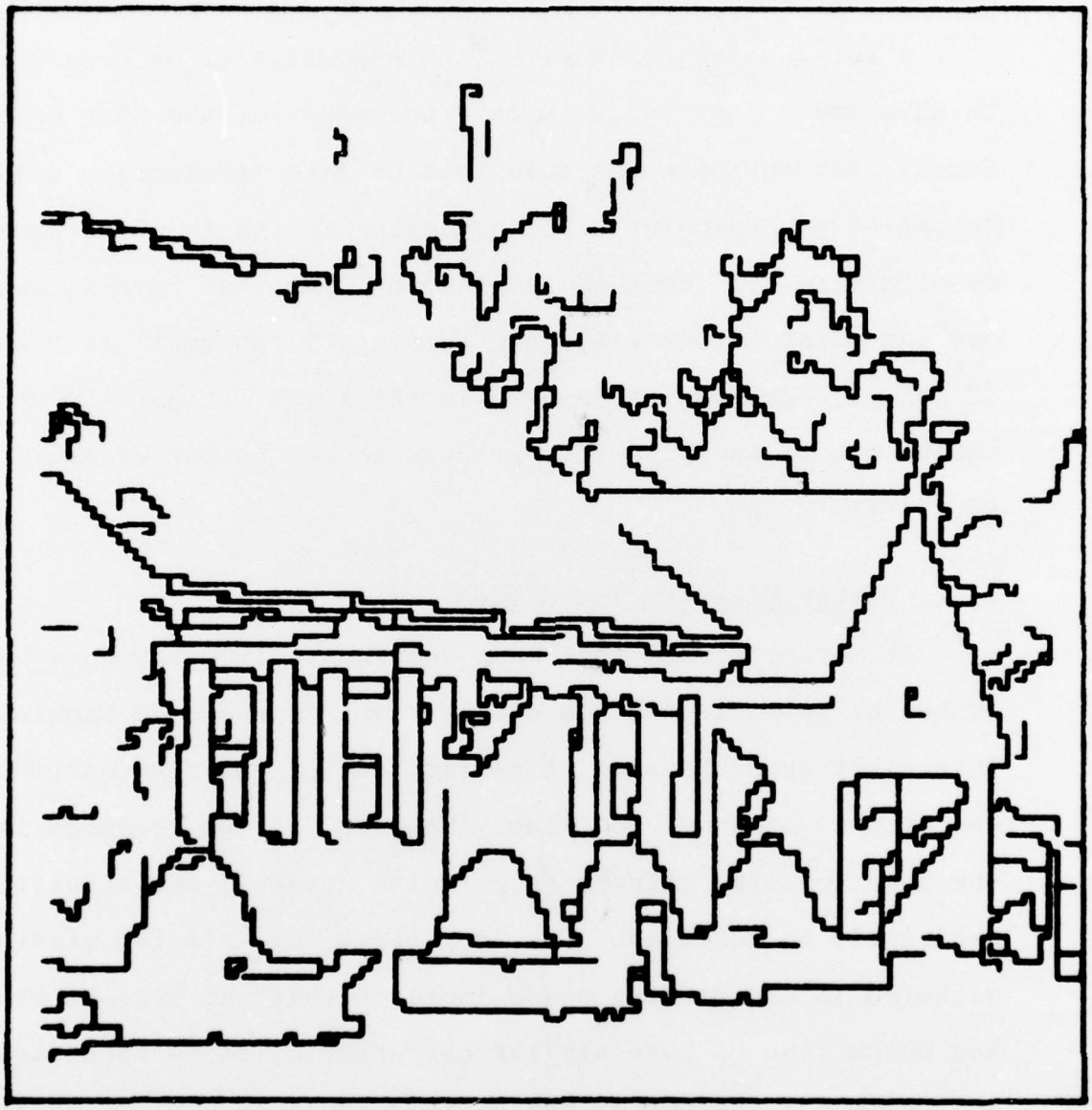


Figure 24. Postprocessing. The short edges and most of the smallest (1-pixel) regions have been removed. This represents only the first set of clean-up techniques which are currently under further development. Figures 24a and 24b show this post-processing applied to the data in Figures 21c and 21e respectively.

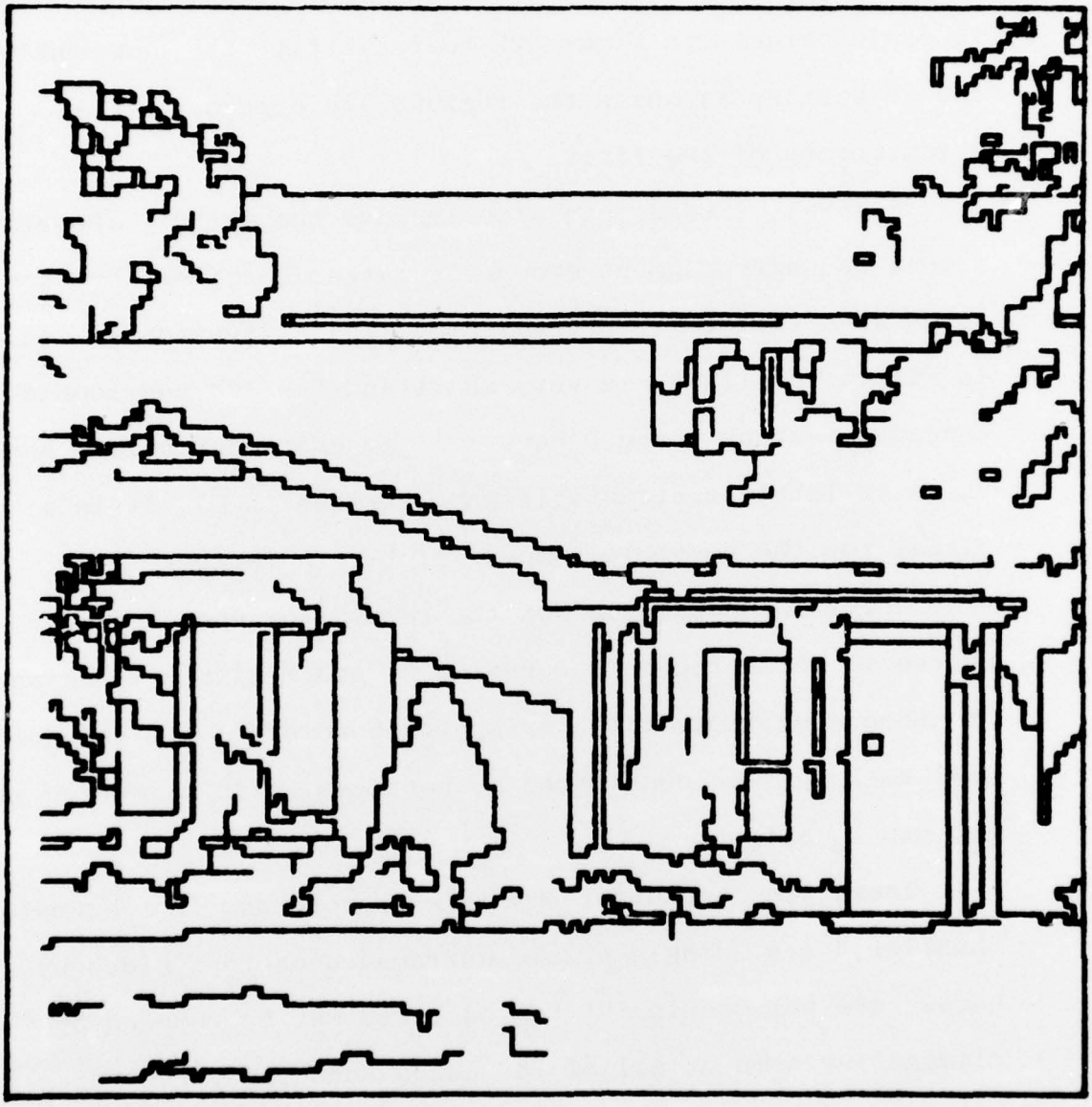


Figure 24b.

where k is a constant between, say, 1/2 and 1. Of course, the references to "above" and "below" are conveniences used in conjunction with Figure 25 to facilitate the description. The directions in which the regions lie depend upon the orientations of the lines.

Removal of edges can also improve the output. Sometimes a weak boundary segment grows (or rather survives) because it is between two other strong lines. This situation is depicted in Figure 26. If L_3 is very short and has low confidence (because regions C and D have smiliar characteristics, and so there is little contrast, if any, between them), it is a signal for the possible removal of L_3. A further

A further condition for the removal of edge L_3 is the degree to which region A ≡ region B, and region E ≡ region F. If these conditions call for L_3 to be removed, its end-points will no longer be considered as vertices, with L_1 merged with L_2, and L_4 with L_5.

There are many other possible approaches. For example, "bubbles," i.e. single pixels surrounded on four sides by edges, are not meaningful regions, and may be removed by eliminating some or all of the edges concerned, according to context. However, since there are so many such heurstics to be tried, we will postpone a thorough analysis for another paper.

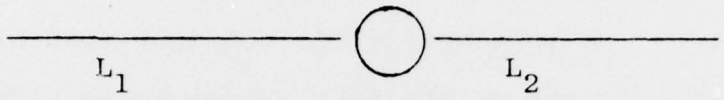


Figure 25: If the properties of lines L_1 and L_2 are sufficiently alike, the vertex can be removed.

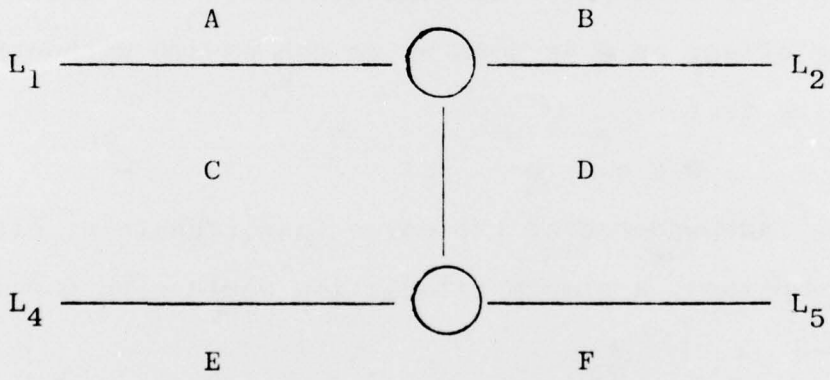


Figure 26: Configuration for possible clean up. See text.

APPENDIX A

Computation of Neighborhood Pattern

We would like to classify the configuration of edges to each side of e as one of the four vertex types given in Figure 15. Consider first the left end-point in Figure 27. We will assume that the numerical values associated with edges are in the range 0-1, representing probabilities of the presence of an edge.

Since we are treating perpendicular continuation as equivalent to straight-line continuation, a and c have exactly the same effect on e as does b, we can assume without loss of generality that

$$a \geq b \geq c.$$

Assuming independence of the edges (unfortunately, often a bad assumption), a simple calculation would give for vertex types 0-3:

$$\begin{aligned} \Pr(\text{type } 0) &= (1-a)(1-b)(1-c) \\ \Pr(\text{type } 1) &= a(1-b)(1-c) \\ \Pr(\text{type } 2) &= ab(1-c) \\ \Pr(\text{type } 3) &= abc. \end{aligned}$$

The case with the highest probability is then chosen as being the "state" of the left side of edge e.

However, in cases where, for example, b and c are very low and a is considerably larger than them but perhaps not close to 1, we would like a strong indication of a type 1 vertex (see Figure 28a). The remedy would be as follows: Instead of subtracting a, b, and c from 1 to form the no-edge

probabilities, we can subtract from m, where m = max(a,b,c) = a in this case. m thus represents at a very local level the probability of a high-confidence edge.

Thus we have:

$$Pr(0) \simeq (m-a)(m-b)(m-c)$$
$$Pr(1) \simeq a(m-b)(m-c) \ldots \text{etc.}$$

There is one difficulty with this formulation. If a is much larger than b or c but this time is very close to zero itself (see Figure 28b), then Pr(1) will be larger than Pr(0) when type 0 should actually be selected. This can be easily fixed by anchoring m to some minimum value, (say·1). We need a lower bound for m because there is always a chance that a stronger edge should be present. This will guarantee type 0 to be the most probable edge when all incident edges have very low strengths. Thus, the final definition of m is

$$m = \max(a,b,c,.1) \text{ and}$$
$$Pr(0) \simeq (m-a)(m-b)(m-c)$$
$$Pr(1) \simeq a(m-b)(m-c)$$
$$Pr(2) \simeq ab(m-c)$$
$$Pr(3) \simeq abc.$$

As it happens, we do not need to normalize these probabilities so that they sum to 1. We only need to know the relative sizes of Pr(i) since we will select type i, where $Pr(i) = \max_j[Pr(j)]$.

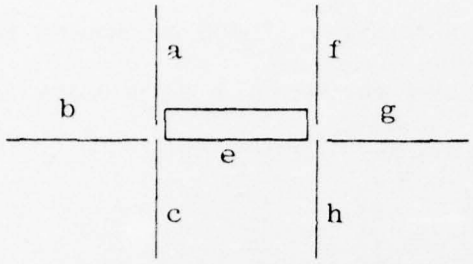
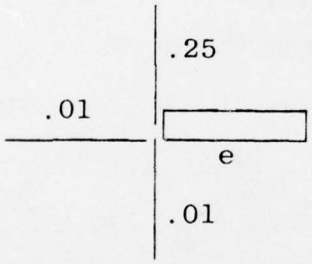
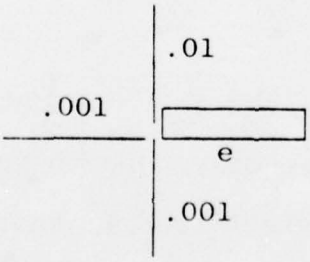


Figure 27: The neighborhood of e, the edge which is to be updated.



28a



28b

Figure 28: Two configurations depicting low-probability neighbors of e; vertex-type 1 is indicated in 28a, and vertex-type 0 in 28b.

REFERENCES

[1] Hanson, A., and Riseman, E. 1976. A Progress Report on Visions: Representation and Control in the Construction of Visual Models. COINS Technical Report 76-9, Department of Computer and Information Science, University of Massachusetts, Amherst.

[2] Williams, T. D., and Lowrance, J. D. 1977. Model Building in the VISIONS High-Level System. COINS Technical Report 77-1, Department of Computer and Information Science, University of Massachusetts, Amherst.

[3] Riseman, E. M., and Arbib, M. A. 1977. Computational Techniques in the Visual Segmentation of Static Scenes. COINS Technical Report 77-4, Department of Computer and Information Science, University of Massachusetts, Amherst.

[4] Tenenbaum, J. M., and Barrow, H. G. 1976. Experiments in Interpretation-Guided Segmentation. Joint Conference on Pattern Recognition and Artificial Intelligence. Hyannis, MA, June.

[5] Feldman, J. A., and Yakimovsky, Y. 1974. Decision Theory and Artificial Intelligence: I. A Semantics-Based Region Analyses. Artificial Intelligence 5(4):

[6] Rosenfeld, A. 1977. Technical Report, Computer Science Center, University of Maryland.

[7] Hanson, A., and Riseman, E. 1974. Preprocessing Cones: A Computational Structure for Scene Analysis. COINS Technical Report 74C-7, Department of Computer and Information Science, University of Massachusetts, Amherst.

[8] Marr, D. 1975. Early Processing of Visual Information. AI Memo 340, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge.

[9] Rosenfeld, A., and Thurston, M. 1971. Edge and Curve Detection for Visual Scene Analysis. IEEE Trans. on Computers C-20: 562-569.

[10] Brice, C. R., and Fenema, C. L. 1970. Scene Analysis Using Regions. Artificial Intelligence 1: 205-226.

[11] Yakimovsky, Y. 1976. Boundary and Object Detection in Real World Images. Journal of the ACM 23(4): 599-618.

[12] Zucker, S. W., Krishnamurty, E. V., and Hoar, R. L. 1976. Relaxation Processes for Scene Labelling: Convergence, Speed and Stability. Technical Report 477, Computer Science Center, University of Maryland.

[13] Rosenfeld, A., Hummel, R. A., and Zucker, S. W. 1976. Scene Labeling by Relaxation Operations. IEEE Trans. on Systems, Man and Cybernetics SMC-6: 420-433.

[14] Zucker, S. W., Hummel, R. A., and Rosenfeld, A. 1977. An Application of Relaxation Labeling to Line and Curve Enhancement. IEEE Trans. on Computers C-26: 394-403.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: COINS-TR-77-7	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER: Technical Rept.	
4. TITLE (and Subtitle): Extracting and Labelling Boundary Segments in Natural Scenes.	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): J.M. Prager, A.R. Hanson, E.M. Riseman	8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0459
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Arlington, VA 22217	12. REPORT DATE: May 1977
	13. NUMBER OF PAGES: 50
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: 57p.	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Distribution of this document is unlimited.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Scene analysis, image processing, boundary extraction, relaxation, artificial intelligence

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This paper describes a set of programs used to perform boundary-analysis in the VISIONS scene-analysis system. These programs lead the data through a sequence of transformations: preprocessing, differentiation using a very simple operator, relaxation using case-analysis, and postprocessing. The output of the system is a set of labelled line-segments for which features such as length and confidence are computed. The lines and associated features will be passed to other portions of the VISIONS system for further analysis.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

407701