PROCESSING CONES:
A COMPUTATIONAL STRUCTURE FOR IMAGE ANALYSIS*

Allen R. Hanson
Edward M. Riseman

COINS Technical Report 81-38

December 1981

Abstract

A layered hierarchical parallel array architecture for image analysis applications, referred to as a processing cone, is described and sample algorithms are presented. A fundamental characteristic of the structure is its hierarchical organization into two-dimensional arrays of decreasing resolution. In this architecture, a protypical function is defined on a local window of data and applied uniformly to all windows in a parallel manner. Three basic modes of processing are supported in the cone: reduction operations (upward processing), horizontal operations (processing at a single level) and projection operations (downward processing). Complex image analysis algorithms are specified as temporal sequences of function applications. The cone structure forms the basis of a sophisticated operating system for an image analysis environment.

This paper has appeared in Structured Computer Vision, S. Tanimoto and A. Klinger (Editors), Academic Press, 1980. It is based on work originally appearing as part of "Preprocessing Cones: A Computational Structure for Scene Analysis," COINS Technical Report 74C-7, 1974.

*This work was supported in part by the Office of Naval Research under Contract No. N00014-75-C-0459, and in part by the National Science Foundation under Grant No. MCS-7918209.

I. Introduction

Over the past several years, much energy has been devoted to the area of scene analysis known as segmentation, or 'low level' vision. The general goal of a low level vision system is the transformation of a large spatial array of pixels (i.e., picture elements) into a more compact description of the image in terms of visually distinct syntactic units and their characteristics. By a variety of means, the visual information in the image must be aggregated, labelled with symbolic names and attributes, and then interfaced to higher level knowledge structures for interpretation of the image [HAN78a]. The syntactic units most often used are boundary segments (connected sets of edges between pixels) and regions (connected sets of pixels); their attributes include properties such as length, size, shape, location, color, and texture.

Given the long range goals of scene analysis systems, one must consider the computational architectures that can facilitate the variety of forms of processing which probably will be required. In almost any application of image analysis, a characteristic which cannot be ignored is the massive amount of visual data which must be processed. For a full-color image of reasonable spatial resolution (512x 512) and color resolution (3 colors, 6 bits/color), close to 5 million bits of information must be processed, often repeatedly. Faced with this computational overload, our group made a commitment to parallel processing at the very beginning of our research effort [HAN74, RIS74]. If such large amounts of sensory data are eventually to be processed by a machine in real time, then the use of large parallel array computers appears to be necessary. It is relevant to note that developments in technology imply that

such devices could be economically feasible in the near future.

Given a choice of developing either serial or parallel algorithms (or both), we have developed parallel algorithms wherever possible. A commitment to the discipline of developing algorithms as local parallel operations pays off in providing a way of thinking about transformations of visual data. Clearly, many algorithms can be implemented in both sequential and parallel versions. However, in much the same way that language appears to affect thought, thinking in terms of parallel computation leads to the development of algorithms which are often not at all obvious in sequential terms. In addition successful demonstration of segmentation algorithms for simple parallel hardware makes future implementation on real machines far more clear.

A second critical consideration is the distinct need to reduce the large amounts of visual information, while at the same time extracting relevant features from local areas of the image. Many interesting features are not a function of individual pixels, but rather a function across the set of points in a local "window" of the image, and the required size of this window will vary. For example the variance of intensity over a square neighborhood centered on a pixel can serve as a texture descriptor of that neighborhood. Given that the size of this neighborhood must vary with the function employed, the environment being imaged, and the resolution of detail (which is dependent upon the distance to the imaged object), the extraction of such features would be facilitated by a hierarchical organization of layers of decreasing image (and processing) resolution.

These design considerations have led us to simulate a general, parallel computational structure, called a "processing cone", for manipulating large arrays of visual data. This parallel array computer is hierarchically organized into layers of decreasing spatial resolution so that information extracted from increasing sizes of receptive fields can be stored and further processed. The structure of the processing cone is described in detail in [HAN74] and applied in [HAN75,NAG77]. It bears a relation to recognition cones of Uhr [UHR72] , the hierarchical data structures of Klinger [KLI76], the pyramids of Tanimoto and Pavlidis, and Levine [TAN75,LEV78], the computational structure of algorithms developed by Rosenfeld et. al [ROS71,HAY74], the planning algorithms of Kelly [KEL71] and Price [PRI77], and knowledge-directed analysis of Ballard, Brown, and Feldman [BAL78]. A survey of some of these uses appears in [TAN78].

The function of our processing cone is the transformation and reduction of the massive amount of image data in a form that facilitates scene interpretation by computer vision systems [HAN78b]. The hierarchy of computational processing provides a structure in which information at higher levels can direct more detailed processing at lower levels of the cone. The cone structure and the range of algorithms definable within the cone are the subject of this paper.

I.1 Overview

The computational structure of the processing cone is designed to facilitate the parallel processing of large arrays of visual data. It is general-purpose in that it may be programmed by defining a prototype computation to be performed on a local window (i.e. subarray) of data. In the cone, this

prototype function will be applied simultaneously - and in parallel - to all local windows across the entire array. The user need only specify the definition of the function, the location of the source(s) of the data within the cone, a description of the size and shape of the local window, and the destination of the result(s) within the cone. The cone's operating system simulates lockstep computation just as if there were parallel arrays of synchronous microprocessors computing on each window; each microprocessor executes a copy of the prototype computation.

An important characteristic of the processing cone structure is its hierarchical organization into layers of decreasing resolution. Figure 1 depicts a cone with an initial image resolution at level 0 of 512x512 pixels; the next layer has one half the resolution on a side (256x256 pixels), etc. Thus, levels 0 through 9 have resolutions of 512x512, 256x256, 128x128, 64x64,...., 2 x 2, 1 x 1, respectively. Since the prototype function associated with a cell at level k is applied to a window of data from level k-1 below it, this type of transformation allows data to be "reduced" up the cone. The decrease in spatial resolution is achieved by requiring adjacent cells at level k to receive data from level k-1 windows which have non-overlapping, but adjacent, 2x2 centers. The effect of this hierarchical reduction in spatial resolution is that cells at successively higher layers store and process information extracted from increasingly larger receptive fields on the image below.

The algorithms which could be implemented within the cone would be severely limited if only a single value could be stored at each cell in a layer.

Therefore, each cell at a given layer is capable of storing a vector of information, not just a single scalar value. For example, the initial data at level 9 might consist of three color components (red, blue, and green). However, additional memory at all levels is usually necessary in order to store both intermediate and final results of processing. This information may be used as the input to further prototype computations. For example, it might be useful to store at a cell some feature of texture and at the same time also have available the average (or maximum or variance) of this feature over a local neighborhood around the cell. In another case, it might be desirable to keep the original color data as well as a region-label image denoting the symbolic region assignment of each pixel. Thus, it is useful to view a level of the cone as a collection, or packet, of planes, where each plane can store either a numeric or a symbolic value.

The design for the cone structure presented here, and which is being simulated in our research on image interpretation, is not intended to be a blueprint for a hardware implementation. Rather, we view it as a tool to be used in the development of parallel processing algorithms for image analysis. Since the forms of processing that are sufficient to achieve these goals are still unknown, maximum flexibility was desired consistent with feasible network connections for a single processor in the network. This has led to a design with connectivity from a given cell to a variable size local neighborhood of cells; the size of the neighborhood is defined under program control up to some fixed maximum value.

II. Modes of Processing and Algorithm Specification

There are three basic modes of processing available within the cone: reduction operations, horizontal (or lateral) operations, and projection operations. These correspond to a flow of information up, laterally, and down the cone, respectively, as shown in Figure 1.

During a reduction process upward through the layers of the cone, a window of data at level k is processed and the resultant value(s) stored at level k+1; the data is reduced since the central portions of each window are non-overlapping.

During horizontal operations, the domain and range of the local function are the same level of the cone, which means that processing is restricted to a single level. The resolution of the data remains constant since each cell at a level will have a window centered over it. This same cell receives the result of the local application of the function, but note that each cell can receive and store a vector of values.

During projection operations, information in upper layers of the cone influence computation at lower layers. This is achieved by extending the definition of the neighborhoood for horizontal processing to include data present in parent cells in the hierarchy above.

In the next sections we will examine each of these modes of processing in somewhat more detail; they are graphically illustrated in Figure 2. The formal

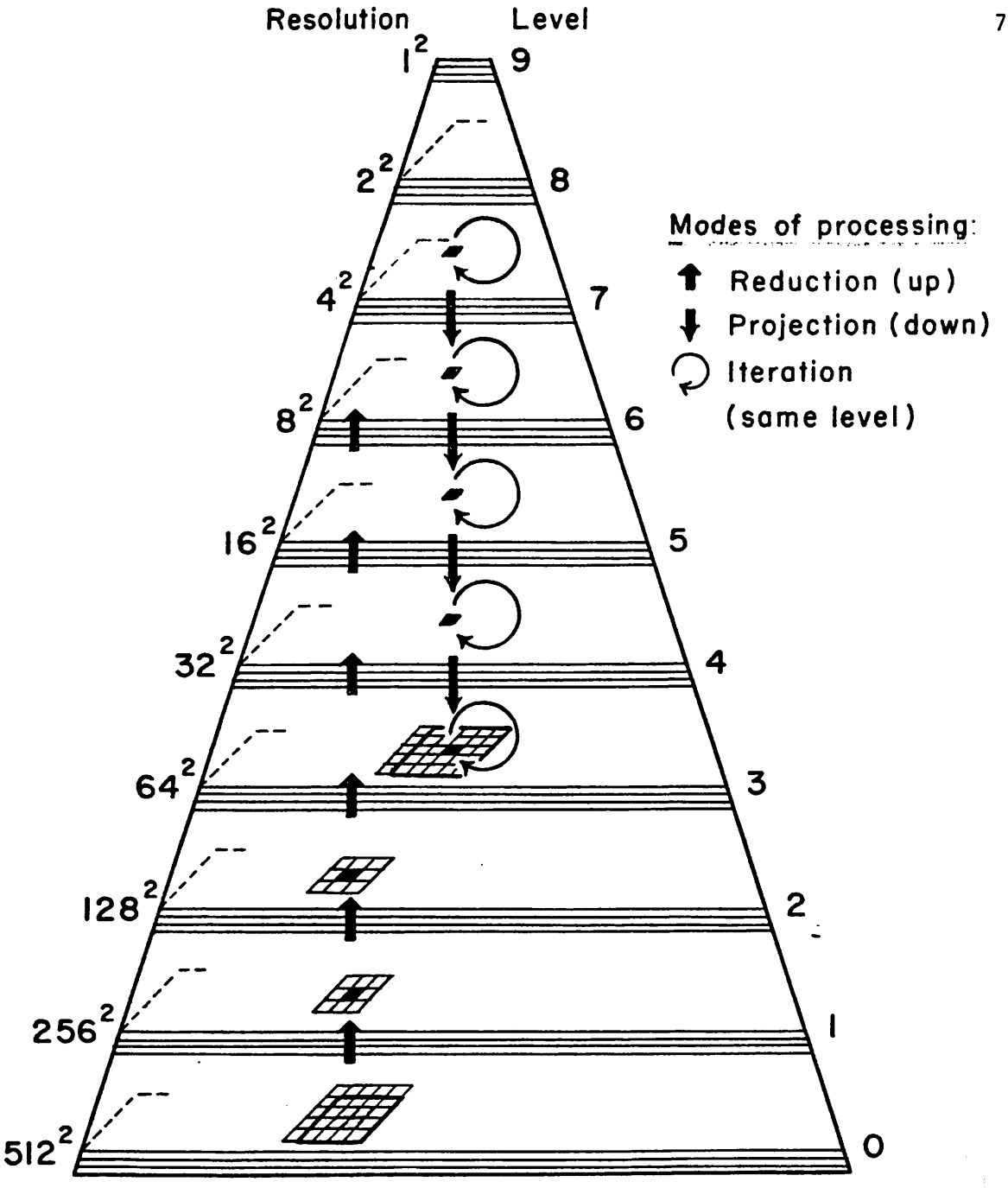


Figure 1. The processing cone. This parallel array computer is hierarchically organized into layers of decreasing spatial resolution. Information within the cone is transformed by means of functions operating on local windows of data. The results of the function are stored in one or more "planes" of data at specified levels. Cone algorithms are specified as sequences of these parallel functions applied in one of three processing modes: reduction (processing up the cone), projection (processing down the cone), and iteration (processing at the same level).

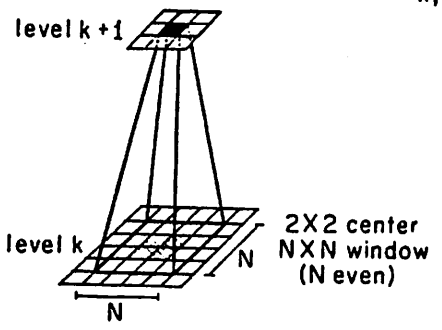
notation specifying these modes is denoted in Figure 2, but the discussion of this notation is delayed until Section II.5.

II.1 Reduction: Upward Processing within the Cone.

During reduction, each function has as its input the data from a local window of cells at level k and outputs one or more values into a single cell at level k+1. Each window is of size nxn, where n is greater than or equal to 4 and is even; each such window is placed over a unique 2x2 set of cells at level k. Since the centers are non-overlapping, each cell at level k+1 is associated with a particular 2x2 subarray at level k, and this provides the decrease in spatial resolution at each higher level. In Figure 2a we have shown the case for n=4 so that each neighborhood overlaps by one cell the neighborhood adjacent to it in the north, south, east, and west directions. The overlap of adjacent n x n neighborhoods avoids such difficulties as, for example, a spatial differentiation operator missing an edge because it lies on the border of adjacent 2x2 centers. The source plane(s) at level k and the destination plane(s) at level k+1 are specified under program control as the input and output of the local function. The window size may also be set under program control.

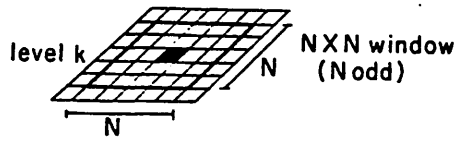
Averaging represents a simple example of the reduction process. It is trivial to average an image up the cone using a single function AVERAGE-UP to obtain a sequence of reduced resolution images. The value found at the 1 x 1 level represents the average intensity of the entire image. One merely has to write a single function AVE which averages the four points in the 2 x 2 center;

(a) Reduction processing: $f^{\dagger}_{k,k+1}(I_1,\ldots,I_D;O_1,\ldots O_R)$



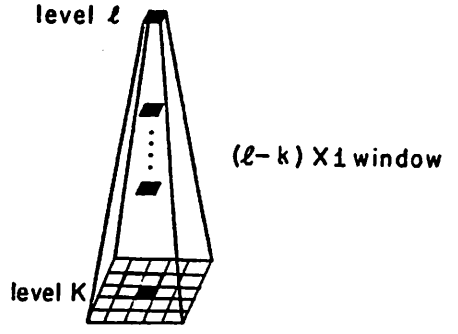
Every cell at level k+1 is associated with a unique 2X2 window of cells at level k.

(b) Horizontal processing: $f^{\dagger}_{k,k}(I_1,\ldots,I_D;O_1,\ldots O_R)$



Every cell at level k is treated as the central cell of an NXN window of cells at level k.

(c) Projection processing: $f^{\dagger}_{(k+1,k+2,\ldots,\ell-1,\ell;k)}(I_1,\ldots,I_{\ell-k};O_1,\ldots O_R)$



Every cell at level k is associated with a window of ancestral cells from level k+1 through the top of the cone, one from each level. The particular cell is determined by the sequence of reduction windows.

Figure 2. Processing modes in the cone. (a) During reduction processing, the local function f is applied in parallel to all even-sized windows of data at level k (the input data is in planes $I_1,\ldots,I_D$). Results are stored in the output planes $O_1,\ldots,O_R$ at level k+1. (b) During horizontal (or iterative) processing, the input data for f is derived from odd-sized windows and the results are stored at the same level in the cone. (c) During projection, the input data for f is obtained from levels higher in the cone. Results are stored in the output planes at level k.

AVERAGE-UP is composed from AVE by applying AVE at level k-1 at time k (for k=1,2,...,9).

II.2 Horizontal (Lateral) Processing: at a Fixed Level in the Cone

During a horizontal processing operation, as illustrated in Figure 2b, each local function receives data at level k from an n x n window (of odd size, n=1,3,5,....) and places the resultant value into the central cell of that neighborhood. Note that the overlap of adjacent neighborhoods is significant when n is greater than 1. Since every cell at level k has its own unique window placed over it, there is no decrease in the resolution of the data. Again, neighborhood size and the memory planes for source and destination values are all under program control.

Many interesting algorithms require iterative applications of a function. Typical examples are region and edge algorithms based on relaxation techniques [HAN78a]. In these algorithms, information in a local neighborhood is used to update a set of likelihoods or confidences of hypotheses at the central cell in the neighborhood. During each iteration the prototype local function is applied simultaneously to every cell at the specified level in a true parallel manner. The number of iterations of a local function can be fixed prior to computation, or it can be determined dynamically as execution proceeds. In the latter case, any of the local copies of the prototype function, as it executes on its local window, can set a global flag which specifies that either another iteration is to be performed or that processing is to be terminated. This implies that the continuation of horizontal processing can be under the control of a global

decision mechanism based on the existence (or absence) of a local property across the image. For example, implementation of a region growing algorithm requires that horizontal processing be performed until no new pixels have been added to the region being grown. In this case, whenever a pixel is assimilated into the region, the particular local process can specify that iteration is to continue by setting the global flag to TRUE. The flag is examined after each iteration; if it is FALSE, then no change has taken place (the region is completely grown) and iteration is terminated.

II.3 Projection: Processing Information Downward in the Cone

In the previous two sections, we have shown how information can flow up and laterally within the cone. These operations have the effect of allowing a local computation to participate in the formation of more global values at levels higher in the cone. However, it is also quite useful to allow global information to influence local computation, (for example in planning mechanisms [KEL71,PRI77] which allow greatly increased efficiency in computation). This is achieved by a projection process which makes available global information from cells at higher levels in the cone to cells at lower levels. A simple example of the utility of this type of processing is found in thresholding operations. In this application, a threshold value is computed at some high level in the cone, perhaps the 1x1 level (level 9). This information is projected down to level k (k<9) where it could be used to set flags on or off, or to set below-threshold values to zero. The threshold might have been obtained by computing the mean and variance of the values at level 0 via reduction operations. A more detailed example using local neighborhood thresholds will be presented in Section III.

Let us examine the projection process a bit more carefully. Each cell at any level of the cone is a member of a unique 2 x 2 center of a neighborhood defined by the reduction operator; that is, it has a parent cell which this neighborhood would map into by reduction. For a given cell at some level, successive reductions denote a set of ancestral cells, with exactly one cell on each higher level (Figure 2c). If values throughout a cone have already been computed, then information can be passed down the cone in parallel by making available all the ancestral information of a given cell during a projection pass.

It is of interest to note that the projection neighborhoods could be extended to allow a window (rather than a single cell) into the information at the level above. Thus, a cell at level k would see an nxn neighborhood around its ancestral cell at level k+1. It is not possible to extend this definition to the level above k+1 without causing an exponential increase in the number of connections to the given cell at level k. This was exactly the reason for not allowing the reduction operator at level k to have direct access to the cells in its receptive field at levels below k-1. In the architecture described here, information is required to pass indirectly to cells at level k via cells at the intermediate levels residing between. In summary, therefore, we maintain in the projection neighborhood only the single ancestor from each level, but consider it a reasonable possibility to expand the ancestor, at the next higher level only, into an nxn window.

II.4 The Full Local Neighborhood of a Cell

In the previous sections we have presented the pure elementary modes of processing in the cone. Sequences of local operations can be used to form

complete algorithms or used to define transformations of the data. In many cases, the domain of the function to be computed at time t in the sequence cannot be constrained to the neighborhood of a single processing mode. The function may simultaneously require information in neighborhoods from more than just one of the reduction, horizontal, and projection modes. This can be achieved indirectly by copying the information into planes at level k by application of pure reduction and projection processes. The function can then be executed via a pure horizontal operation. However, such processing can be made more efficient by allowing a function at some level to access a window composed of the union of the three types of windows. Simultaneous access to all three windows allows hybrid modes of computation - computation that could not be classified as any of the elementary modes - where the domain of the function is composed of data from more than one level.

The full neighborhood of a cell at level k is shown in Figure 3. If the reduction neighborhood is 4 x 4 and the horizontal neighborhood is 5 x 5, then a cell at level k will have simultaneously available the storage planes of 16 cells at level 0 (the reduction neighborhood), 25 cells at level 1 (the horizontal neighborhood) and the unique set of 8 ancestral cells from levels 2 through the apex of the cone.

II.5 Functional Notation and Algorithm Specification

In this section the notational conventions used to specify parallel algorithms will be presented. In some cases sequences of operations can be simply expressed diagrammatically in such a way that the domain and range of the

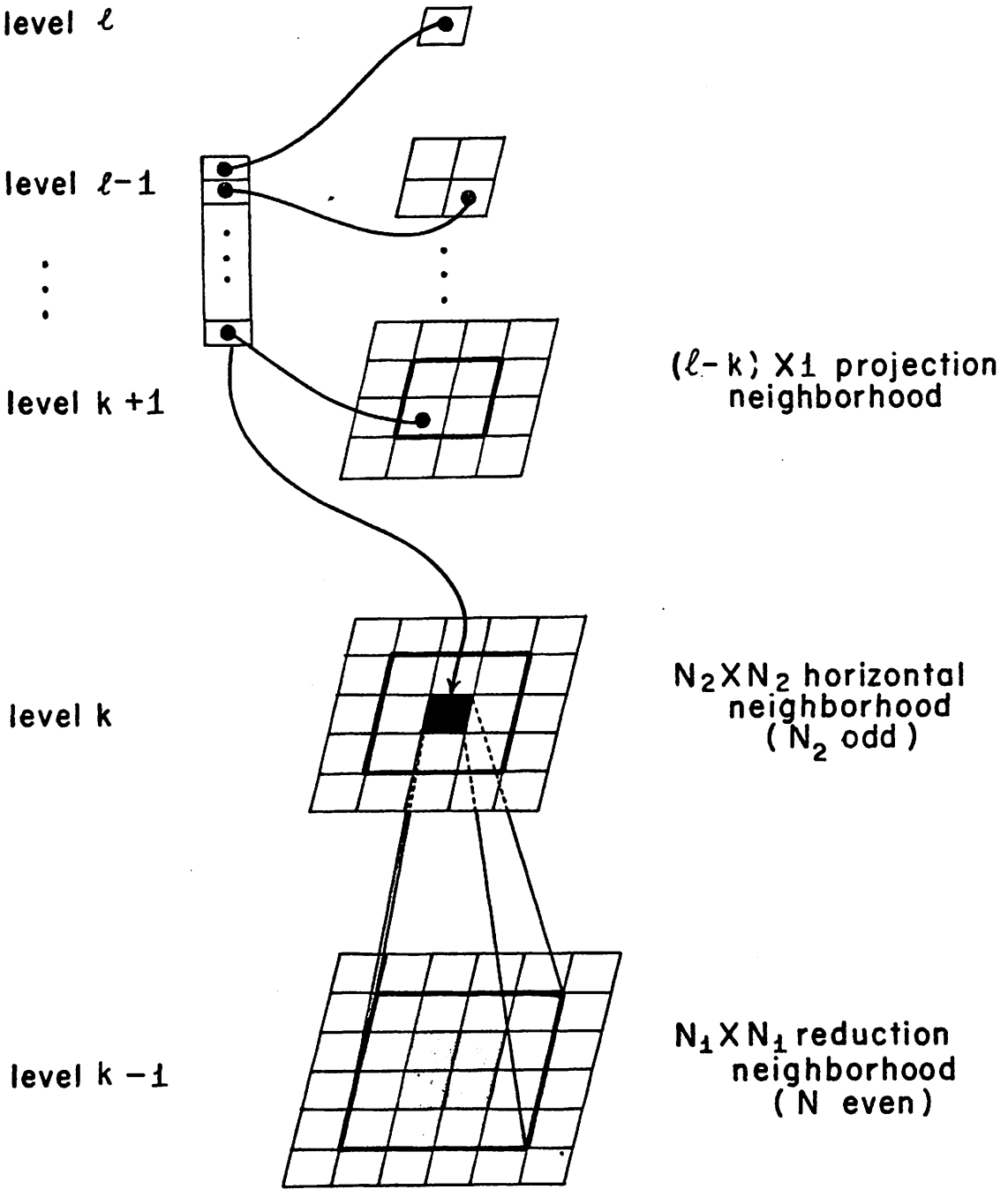


Figure 3. The full local neighborhood of a cell in the cone. Given a particular cell at level k, the value placed there can be computed from the data which is simultaneously available from the neighborhoods defined for the three processing modes. Thus, the domain of the local function is the union of the three types of windows associated with the three pure processing modes defined in Figure 4.2.

function, and the temporal relationships between functions, are obvious from the context. In general, however, this simple notation will not be sufficient. Some algorithms will require precise specification of both the levels and specific planes for the domain (i.e., the input values) and range (i.e., the output values) of each primitive function, as well as the relative order in which the functions are to be applied. The following notation will be used to specify one elementary operation of reduction, horizontal computation, projection, or a combination of horizontal and projective processing:

$$f^{t}_{ij}\ (I_1,\ldots I_D; O_1,\ldots O_R)$$

where the symbols are represented as follows:

f — the symbolic name of the function (and associated program code),

i, j — level i is the domain and level j is the range of f; information is processed from level i to level j,

$I_1,\ldots,I_D$ — the domain of f is the D memory planes $I_1,\ldots,I_D$ at level i,

$O_1,\ldots,O_R$ — the range of f is the R memory planes $O_1,\ldots,O_R$ at level j,

t — the time stamp, used to specify the relative time of activation of f in a sequence of function applications.

Where it is necessary, this notation will be extended slightly in order to handle the cases where the memory planes of the domain and range do not all reside at the same level:

$$f^{t}_{(i1,\ldots,iD;j1,\ldots,jR)}(I1,\ldots,ID;O1,\ldots OR)$$

In this full notation, the memory plane of the input argument Ik is at level ik and the destination memory plane for the output argument Ok is at level jk. Thus, there is explicit specification of the level of every input argument and output argument of function f. The notation has become somewhat clumsy, but it usually not necessary to employ the full notation. Wherever possible, a cone diagram with arrows indicating the flow of processing will be used to sketch the execution of the algorithm. For simplicity we will omit many of these parameters when the intended computation is clear from the context.

Using this functional notation, let us present an example algorithm which is a composition of functions defined over the levels of the cone and time t. The algorithm, MEASURE-EDGENESS, is to compute the average edge strength from level 0 to level k as shown in Figure 4. The function will also be required to compute the average edge strength squared since it will be used again in Section III. Thus, MEASURE-EDGENESS requires two subsidiary functions:

1. DIFF - a horizontal function which performs a spatial differentiation of the original data using a 1x2 mask applied horizontally and vertically, and then takes the max of these two values; and

MEASURE - EDGENESS:

$DIFF^{1}_{0,0}(I;E)$,

$FORM\text{-}ESQ^{2}_{0,0}(E;ESQ)$,

$AVE^{3}_{0,1}(E,ESQ;\ AVE\text{-}E,\ AVE\text{-}ESQ)$,

$AVE^{4}_{1,2}(AVE\text{-}E,\ AVE\text{-}ESQ;\ AVE\text{-}E,\ AVE\text{-}ESQ)$,

$\vdots$

$AVE^{k+2}_{k-1,k}\ (AVE\text{-}E,\ AVE\text{-}ESQ;\ AVE\text{-}E,\ AVE\text{-}ESQ)$

Graphic Notation:

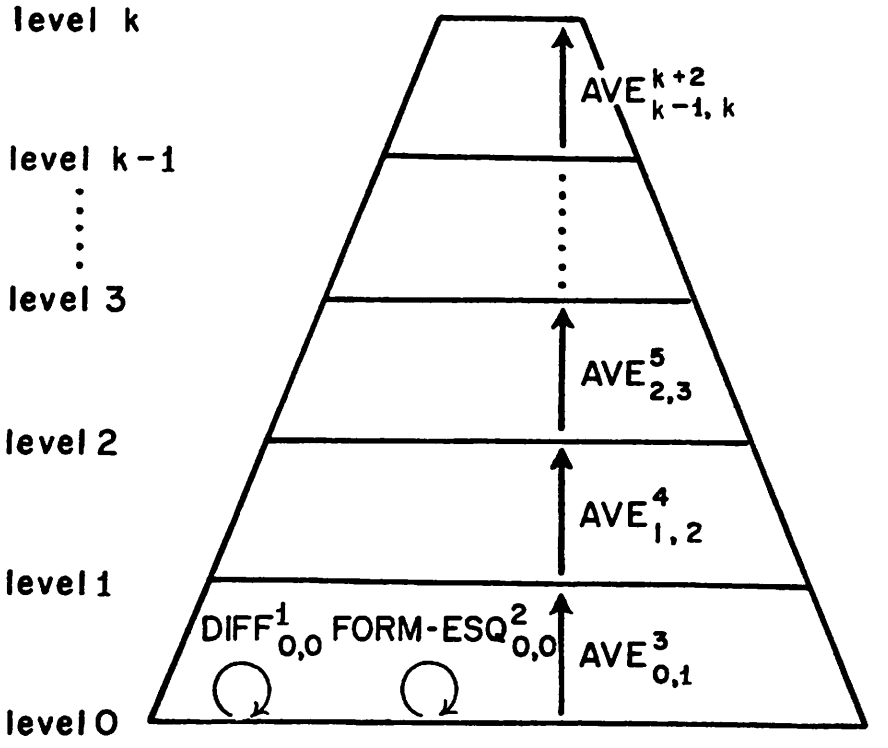


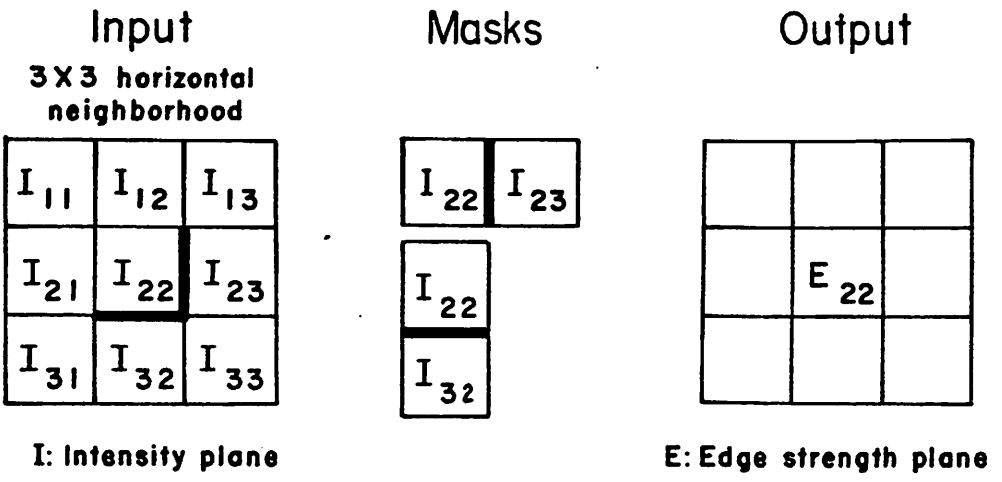
Figure 4. MEASURE-EDGENESS cone algorithm. This algorithm computes the average edge strength of an image from level 0 to level k; it also computes average edge strength squared. It is defined by the sequential application of parallel local functions, each of which is one of the three basic modes of processing.

2. AVE - a reduction function which averages data from one level to the next level above in the cone.

First, let us examine the spatial differentiation operator illustrated in Figure 5a. DIFF (I;E) is applied as a horizontal processing operator to the image intensity plane I at level 0, and the result is stored in plane E at level 0. Spatial differentiation is often achieved by convolving a simple mask with the image [DUD73] and here we use the smallest meaningful mask, of size 1x2 [HAN78a]. The result stored in plane E is a numeric value representing the maximum of the local contrast change across pixels to the right and below. The additional function FORM-E-SQ (E;ESQ) will just store in a plane called ESQ the square of the values in plane E.

The local functions are most conveniently specified in a PASCAL-like notation, extended to include window types of the appropriate kind (e.g. reduction, horizontal, and projection) and basic operations on these types. The procedural definition of DIFF is shown in Figure 5b and should be self-explanatory. Now, let us examine the definition of the reduction function AVE as described in Figure 6. It employs a square 2x2 neighborhood and simply averages the four values in the two specified planes and stores the results in two new planes at the level above. By applying a reduction sequence of AVE's up the cone, the running partial sums E and ESQ can be computed up to level k.

The results of applying MEASURE-EDGENESS to the image shown in Figure 7 are presented in Figure 8 for selected levels of the cone.



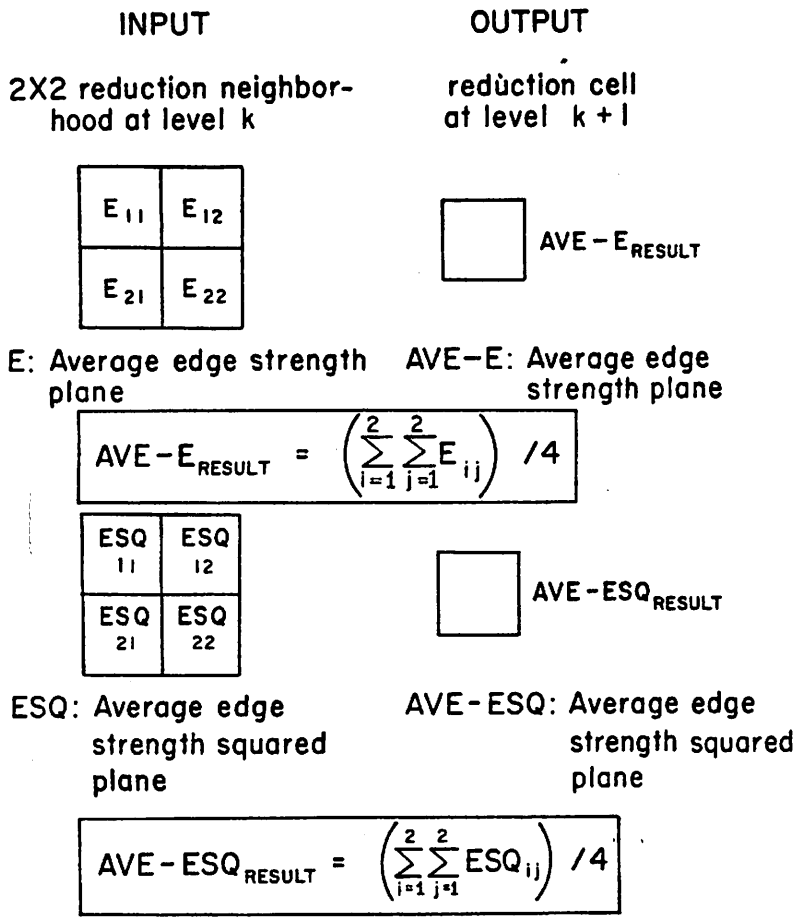
$$E_{result} = E_{22} = MAX\left\{\left|I_{22}-I_{23}\right|, \left|I_{22}-I_{32}\right|\right\}$$

(a)

```
Procedure: DIFF ( I ; E ) ;
/* Notation: I is intensity plane, E is edge plane */ ;
Window: W( 3,3) of type horizontal ;
Begin  /* find max of horizontal and vertical edges
       around central cell of W and store in edge
       plane */
   E[2,2] ← MAX( ABS (I[2,2] - I[2,3]) ,
                 ABS (I[2,2] - I[3,2]) )
End
```

(b)

Figure 5. Specification of the local differentiation operator DIFF(I,E). (a) The differentiation function is applied in the horizontal processing mode to a 3×3 window of data; in this example intensity is the data stored in plane I. The function computes the maximum response to two 1×2 edge masks and stores the result in the central cell of the window in a plane named E. (b) DIFF in a procedural representation using a PASCAL-like notation. I is the input plane on which is defined the 3×3 horizontal window; E is the name of the output plane.




```
procedure AVE (E, ESQ; AVE-E, AVE-ESQ)
/* Notation: E, ESQ: input planes
      AVE-E, AVE-ESQ: output planes */
window E (2,2), ESQ (2,2) of type reduction;
begin /* compute average edge strength and
   edge strength squared */
   SUM-E ← 0;
   SUM-ESQ ← 0;
   foreach X ε W(E) do
      SUM-E ← SUM-E+X;
   foreach X ε W (ESQ) do
      SUM-ESQ ← SUM-ESQ+X;
   AVE-E ← SUM-E/4;
   AVE-ESQ ← SUM-ESQ/4;
end
```

(b)

Figure 6. Specification of the local averaging function AVE. (a) AVE computes two results which are the averages of data in 2×2 reduction neighborhoods provided as two distinct input planes. An equivalent result is obtained by the individual application of a single function with one argument to the two planes. (b) Functional representation of AVE in a PASCAL-like notation. Two input planes and two output planes are specified. The window type specifies reduction processing, which fixes the output planes at level k+1 if the input planes are at level k. The mode of processing is specified by a declaration of the window type; the specification of reduction window fixes the output planes at level k+1 for input planes at level k. Elements of data are accessed by window W applied to plane E.

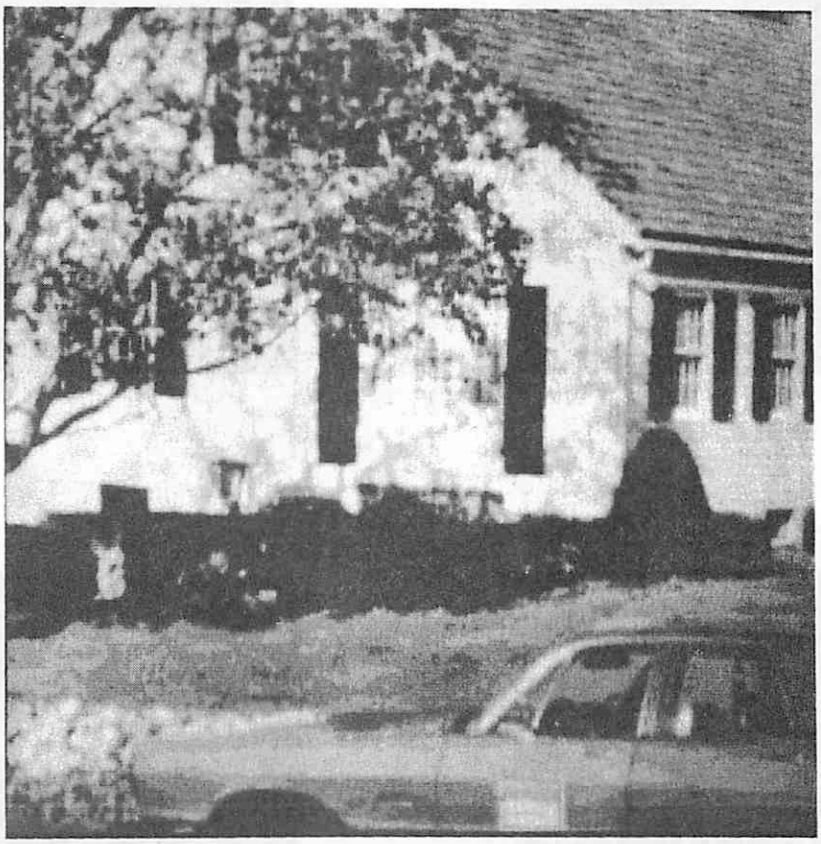


Figure 7. House image. This image is an intensity image of 256×256 resolution. Thus, level 0 in the cone corresponds to a resolution of 256×256 pixels. It was derived by selecting one quarter of a 512×512 color image (6 bits/color). The image contains a variety of image characteristics, including smooth regions, weakly and heavily textured regions, and strong isolated boundaries.

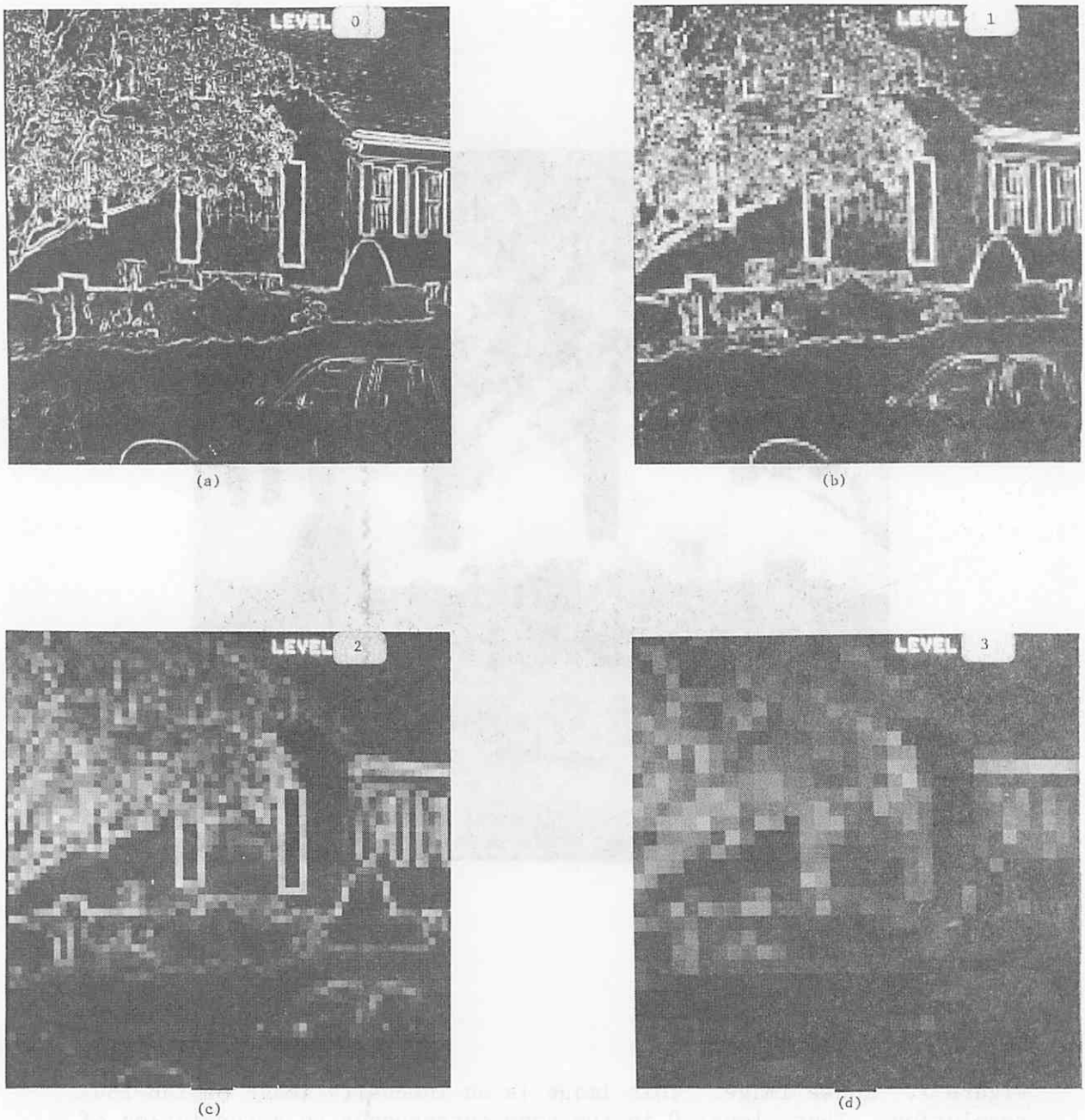


Figure 8. Results from MEASURE-EDGENESS for selected levels in the cone. (a) Output from the differentiation operator DIFF(I,E) at level 0. (b)-(d) Average edge strength at levels 1, 2 and 3, respectively, encoded by brightness. At level 3, each cell "sees" a window of data of size 16×16 at level 0. Note that only the average edge strength is shown; average edge strength squared is also used in the computation of variance in subsequent algorithms.

III. Cone Algorithms

Algorithms designed to be incorporated into an image understanding system must extract a variety of information [HAN78c] from a scene and often can become quite complex. One class of problems involves the segmentation of an image, that is the paritioning of an image into areas -- or regions -- based on invariance of some subset of visual features. One general class of these algorithms involves extraction of boundaries between regions [HAN80], while a second class involves the grouping of pixels directly into regions [NAG79]. If the image being analyzed has any significant degree of textural variation, then the problems encountered in extracting this information are greatly magnified. In such cases it is necessary to extract features that typify the textural variation in order to carry out the segmentation. Whether or not there are strong textural characteristics present, after producing a segmented image, various properties of the regions and boundaries must be extracted and processed using stored knowledge of real-world events in order to form reasonable hypotheses concerning their identities [HAN78b].

By now it should be clear that a wide range of algorithms for manipulating images is required in order to perform reliable image segmentation and feature extraction. The range of algorithms which have been implemented within the processing cone structure is shown in Table I. In this paper we present only two algorithms, which have been chosen to illustrate the capabilities of the processing cone.

Features and algorithms

Average (blurring)
Spatial differentiation - edge formation
Thresholding
Texture measures:
- Edge orientation
- Edge contrast / unit area
- Weak edge suppression
- Strong edge density
- Variance
- Texture coarseness - spot size *
- Density of local extrema
- Average of local extrema

Functions of spatial gray level adjacency matrices
Region growing (all regions simultaneously)
Region masking
Region size
Region compactness - p^2/A
Shape matching
Relaxation labelling algorithms
Hierarchial relaxation labelling

* Approximation to original algorithm [HAN74, ROS74]

Table I. A sample of the image analysis algorithms which have been implemented in the cone.

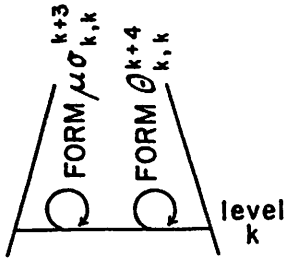
III.1 A Simple Example: Weak Edge Suppression By Local Thresholds

Edge operators are used in image analysis to transform a gray-level intensity image into a "line drawing" which denotes boundaries of regions of relatively similar gray level. When an edge mask is convolved with an image, however, there usually is a significant degree of variation in the responses, ranging from strong responses from high-contrast boundaries to weak responses from low-contrast boundaries or noise. In order to emphasize the more important high-contrast boundaries, the weaker responses can be removed via a thresholding operation. The problem, though, is that the threshold must vary dynamically not only with the global contrast characteristics of the image, but also with the local characteristics within the image (e.g., the content of subimages). What is perceived as a significant edge in one area of the image may not be perceived as such in another area, depending on the context in which the edge is found. The algorithm that we present here will dynamically determine thresholds for local subimages as a function of the average and variance of edge contrast over the subimage.

Figure 9 describes an algorithm which first computes edge strength and then suppresses weak edges based on a statistical analysis of the edges over rectangular receptive fields. This algorithm utilizes the MEASURE-EDGENESS algorithm described in Section II.4. Using the output of MEASURE-EDGENESS, the mean and standard deviation for all the 2**k x 2**k neighborhoods up the cone is computed by executing FORM-μ-σ at level k (k=1,2,...,9). At any level the computation of μ and σ is a simple function of E and ESQ during one horizontal iteration. The threshold THETA-E can be a simple linear function of

THRESHOLD-WEAK-EDGES: MEASURE-EDGENESS,
FORM-THRESHOLD,
REMOVE-WEAK-EDGES.

FORM-THRESHOLD: FORM $\mu\sigma_{k,k}^{k+3}$(AVE-E, AVE-ESQ; μ_E, σ_E),
FORM $\theta_{k,k}^{k+4}$($\mu_E, \sigma_E; \theta_E$)



FORM $\mu\sigma_{k,k}^{k+3}$(AVE-E, AVE-ESQ; μ_E, σ_E)

1X1 horizontal neighborhood:

μ_E = AVE-E

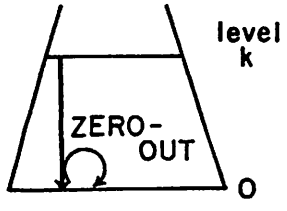
$\sigma_E = \left[\text{AVE-ESQ} - (\text{AVE-E})^2\right]^{1/2}$

FORM $\theta_{k,k}^{k+4}$(μ_E, σ_E; THETA-E)

1X1 horizontal neighborhood:

THETA-E = $\mu_E + \omega\sigma_E$ ω user supplied

REMOVE-WEAK-EDGES: ZERO-OUT$_{(0,k;0)}^{k+5}$ (E, THETA-E; E)



1X1 horizontal neighborhood at level 0
1ancestral cell at level k

$$E = \begin{cases} E \text{ if } E \geq \theta_E \\ 0 \text{ otherwise} \end{cases}$$

Figure 9. Definition of THRESHOLD-WEAK-EDGES. This function first computes edge strength using MEASURE-EDGENESS, and then suppresses weak edges by performing a statistical analysis of the edge strength over rectangular receptive fields to form a view of the data, but the global effect of the analysis can be controlled by selecting the level in the cone at which the threshold is formed. A single horizontal pass at level 0 removes edges below threshold. The weakness of the current algorithm is that thresholds for neighborhoods can change abruptly because the windows over which data is extracted are non-overlapping.

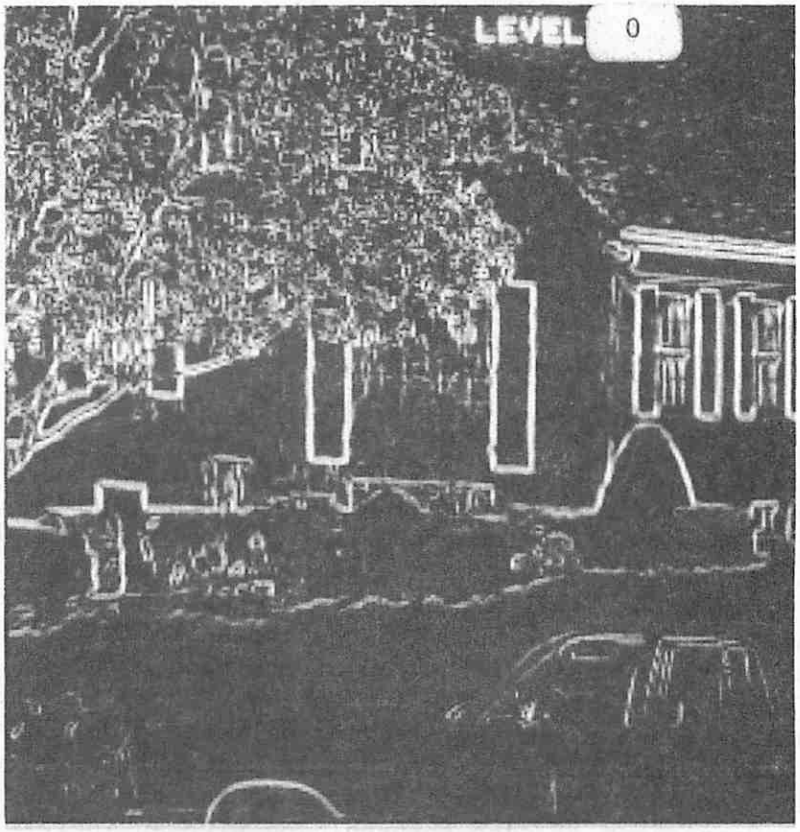
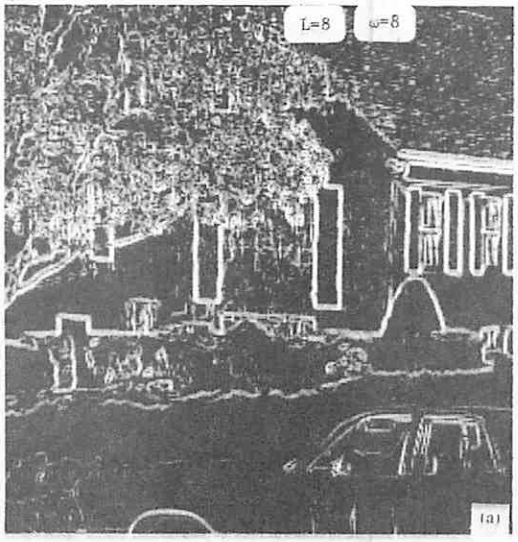
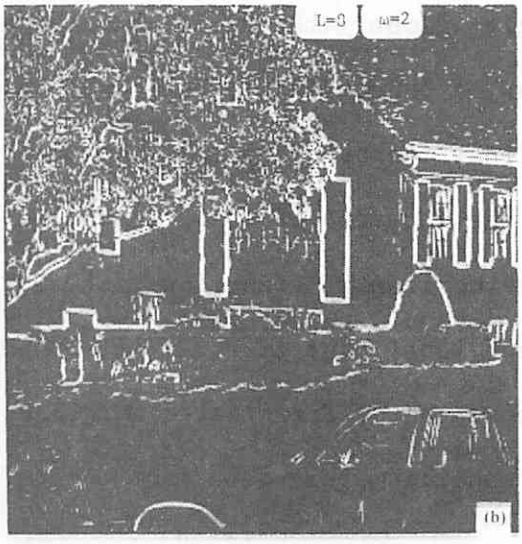
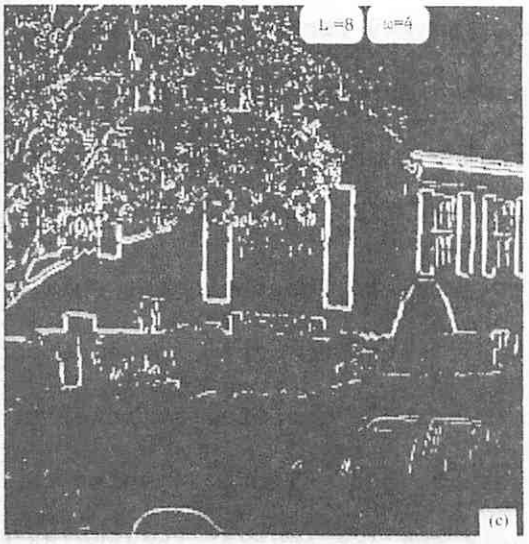


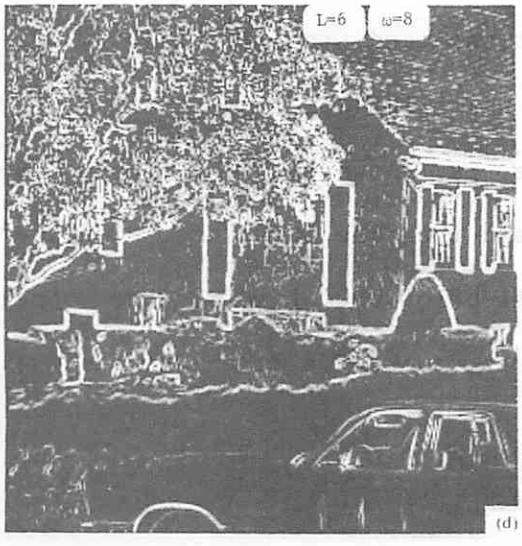
Figure 10. Differentiated image at level 0.

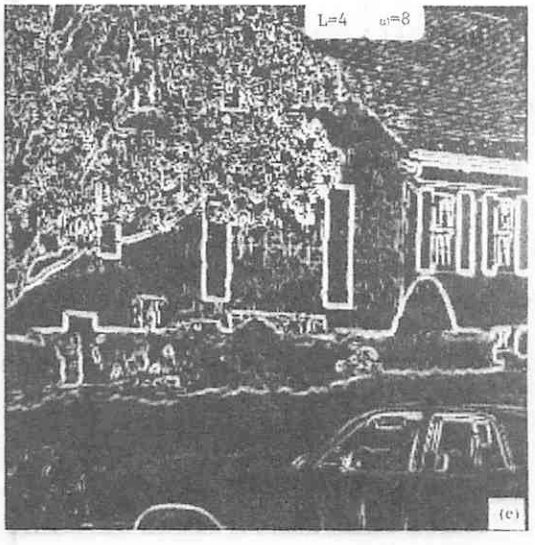
Figure 11. Weak edge suppression by local thresholds. The set of images demonstrates the effect of choice of level and edge threshold ($\theta = \mu+\omega\sigma$) on the removal of weak edges. The level defines the local neighborhood over which the edge threshold is computed. When L=8 the threshold is global because it is computed at the top of the cone. (a)-(c) Comparison of edge thresholding with increasing ω shows that more and more weak edges are removed, leaving only the globally strongest edges. Note that much weak texture is removed (e.g., in the wall and roof) at the expense of some of the car boundary. (a,d,e) Comparison of edge thresholding with an increasingly more local receptive field over which θ is computed, shows that globally weak but locally strong edges can be retained. The car boundary is more complete and the weak edges in the wall are still removed, but more of the roof texture remains.


L=8
ω=8
(a)


L=8
ω=2
(b)


L=8
ω=4
(c)


L=6
ω=8
(d)


L=4
ω=8
(e)

μ and σ :

$$\text{THETA-E} = f(\mu,\sigma) = \mu + w\sigma$$

where w can be a positive or negative real number, depending upon the degree to which edges are to be suppressed. A local value for THETA-E will be computed for each cell at level k by executing FORM at that level. Thus, each non-overlapping square 2**k x 2**k neighborhood at level 0 will have its own context-sensitive threshold based upon the mean and variance of the contrast of edges in its field of view. If the cell at the top of the cone is employed (in our case k=9 for a 512x512 image), then a single global threshold has been computed for the whole image. At one level below, a threshold has been computed for each quadrant of the image, etc. Only one final iteration pass is needed at level 0 (or, in general, the level from which the sequence of computations was initiated) to threshold the entire image. The horizontal function ZERO-OUT merely sets to zero the edge contrast E of each local edge if it is less than the threshold computed at level k. Figure 10 shows the differentiated image at level 0 and Figure 11 shows the effect of thresholding this data from several levels and for several thresholds.

Note the simplicity of the local operations and the flexibility for choosing the size of the receptive field over which THETA-E is computed merely by moving to various levels in the cone. There is, however, one less than satisfactory characteristic of this algorithm and the cone structure. Consider the quadrants projected on level 0 from one level below the top of the cone. The boundaries of these neighborhoods are abrupt, so that a line segment (i.e.

a contiguous sequence of edges) crossing quadrants may have one edge above threshold in one quadrant, and yet have an adjacent equivalent strength edge below threshold in the other quadrant. Each local edge does not have its own symmetrically centered neighborhood for computing its optimum threshold. This problem is seen in various forms in several algorithms. In this case the effect of this problem might be reduced somewhat by utilizing averaged or interpolated thresholds from one or two levels below for those level 0 cells that are near a border of the receptive field from level k. This only requires an additional check of the location of each level 0 cell (which is available in our simulated architecture).

IV. An Example: Density of Strong Edges

One of the primary problems in the analysis of two dimensional images is the segmentation of the image into meaningful regions. A patterned shirt or a wood grained table top possesses textural characteristics whose analysis would be aided by quantification of two-dimensional patterns of variation. In particular it would be very useful to determine features of such patterns which are invariant and which therefore allow the extraction of such regions.

One elementary measure of textural variation, Average Edge Contrast/Unit Area, has been examined by Rosenfeld, et al [ROS71]. However, this measure has a serious shortcoming in that it is a function of both contrast and density of edges. Consequently, a low density of strong (high contrast) edges could produce the same value as a high density of weak (low contrast) edges. Thus, it is useful to compute edge density with contrast factored out. This can be

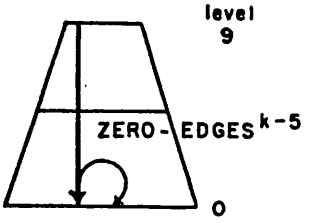
achieved by defining a test for the existence of an edge. Density of edges can then be computed as a function only of those edges which pass the test. Now, one should note that a threshold on edge contrast can be used as the test for detecting the presence of 'strong' edges. Once edges have been turned on or off, contrast need not be used in the computation of the density; weak edges can be suppressed and only the density of strong edges will be computed. Conversely, by turning off strong edges, only the density of weak edges is computed. Clearly, the density of intermediate edge values can be computed in a similar manner.

An algorithm for computing the density of strong edges is shown in Figure 12 and the results obtained for the image of Figure 7 are shown in Figure 13. The test for strong edges is straightforward. The algorithm of FORM-THRESHOLD (refer to Figure 9) determines the mean and standard deviation for edge contrast. It is computed across the entire image by moving data to the top of the cone and then a threshold $\theta = \mu + w\sigma$ is computed. w is positive here because strong edges will be assumed to have a contrast above the mean of all edges. For w=1, only edges whose contrast is greater than one standard deviation above the average edge contrast are maintained as strong edges. At this point, in place of edge contrast a flag of 1 in the plane E-SUM at level 0 is used to mark the presence of a strong edge. The processing is completed in a straightforward manner by counting the number of flags over a local area with the reduction function COUNT-STRONG-EDGES, and then normalizing by area in the horizontal operation GET-DENSITY.

It is worth summarizing the overall flow of processing up and down the

DENSITY-OF-STRONG-EDGES: MEASURE-EDGENESS
FORM-THRESHOLD,
FLAG-STRONG-EDGES,
COUNT-STRONG-EDGES.

FLAG-STRONG-EDGES: $\text{ZERO-EDGE}^{k+5}_{(0,9;0)}\left(E, \text{THETA-E}; \text{E-SUM}\right)$



1X1 horizontal neighborhood at level 0
1 ancestral cell at level 9

$$\text{E-SUM} = \begin{cases} 1 \text{ if } E \geq \text{THETA-E} \\ 0 \text{ otherwise} \end{cases}$$

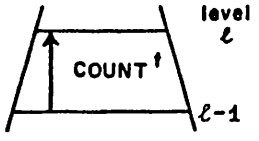
Note: ZERO-EDGES is the same as ZERO-OUT from figure 7 except that here the output is a flag (E-SUM) which is set if the edge is above the local threshold. In this way, the edge contrast is factored out of the current algorithm.

COUNT-STRONG-EDGES: $\text{COUNT}^{k+6}_{0,1}\left(\text{E-SUM}; \text{E-SUM}\right)$

$\vdots$

$\text{COUNT}^{k+\ell+5}_{\ell-1,\ell}\left(\text{E-SUM}; \text{E-SUM}\right)$,

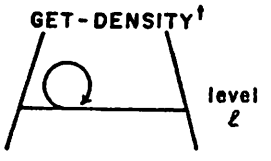
$\text{GET-DENSITY}^{k+\ell+6}_{\ell,\ell}\left(\text{E-SUM}; \text{DENSITY}\right)$



$\boxed{\text{COUNT}^{\uparrow}_{\ell-1,\ell}\left(\text{E-SUM}; \text{E-SUM}\right)}$

2X2 reduction neighborhood

$$\text{E-SUM} = \sum_{n=1}^{2}\sum_{m=1}^{2} \text{E-SUM}_{ij}$$



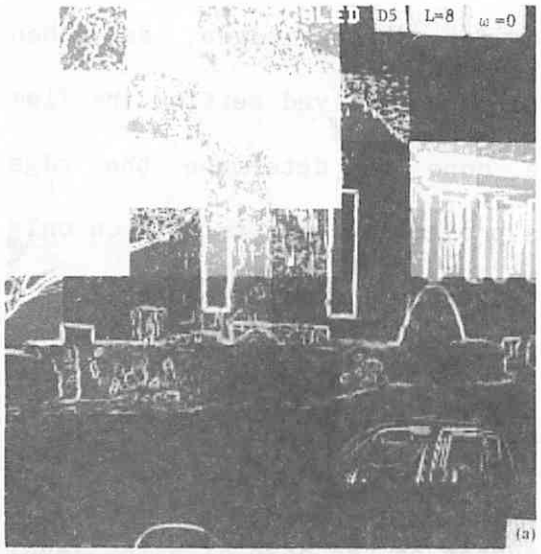
$\boxed{\text{GET-DENSITY}^{\uparrow}_{\ell,\ell}\left(\text{E-SUM}; \text{DENSITY}\right)}$

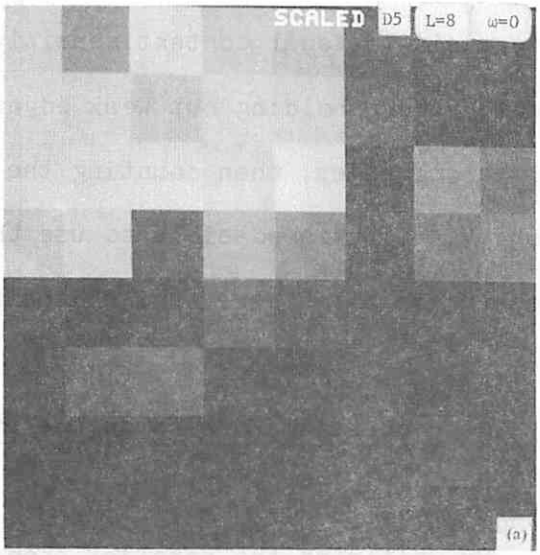
1X1 horizontal neighborhood at level ℓ

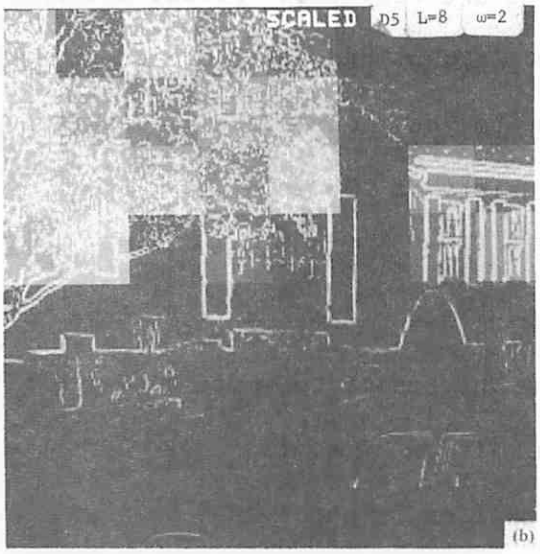
$$\text{DENSITY} = \frac{\text{E-SUM}}{2^{\ell} \times 2^{\ell}}$$

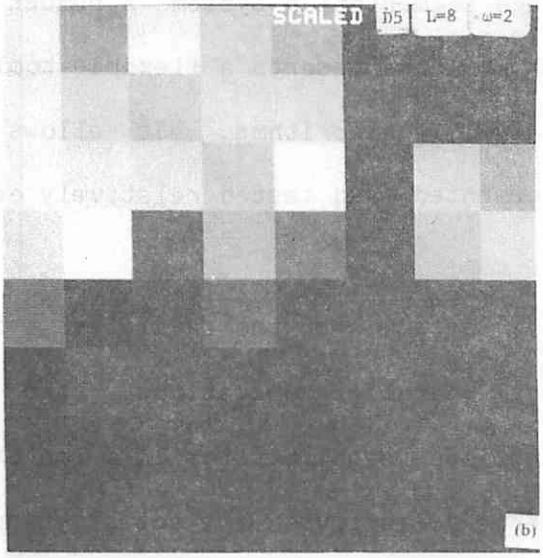
Figure 12. Algorithm for computing the density of strong edges. This algorithm computes the density of strong edges over local windows in the cone in such a way that the density is not a function of edge contrast. It is similar to the weak edge suppression algorithm except that edges that are above threshold are flagged with a value of 1. A simple counting operation to level ℓ in reduction mode results in the number of strong edges in the $2^{\ell} \times 2^{\ell}$ neighborhoods at level 0. A horizontal operation which normalizes these values by the area of the neighborhood gives the density of strong edges.

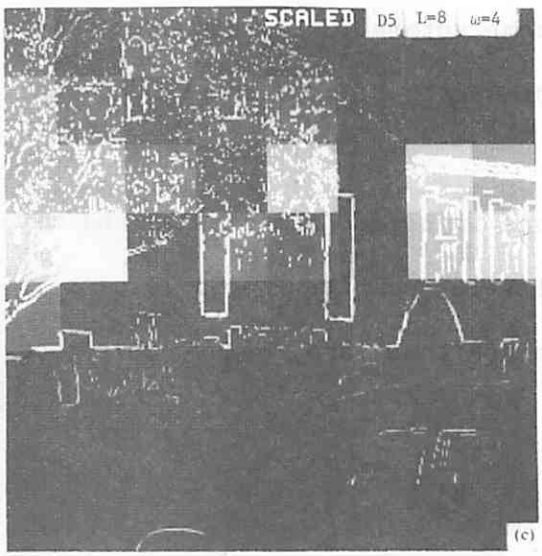
Figure 13. Density of strong edges. The images show the effect of increasing the edge threshold ($\theta = \mu+\omega\sigma$) upon the density of strong edges computed to level 5 in the cone. The weak edge threshold was a global threshold computed from level 8. In each pair of pictures, the left image is the strong edge density and the right image is the same data laid over the thresholded edge images. Increasing the weak edge threshold, which has the effect shown in Figure 11, results in a better separation between heavily textured areas (i.e., high density of strong edges) and the background. It may be possible to use the edge density data as a mask, delineating heavily textured areas.

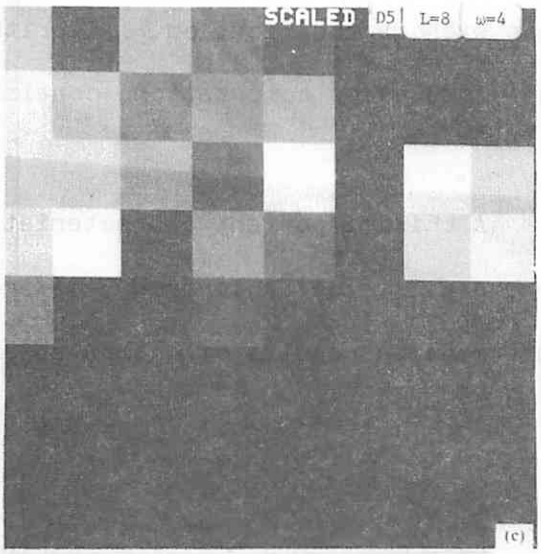

D5 L=8 ω=0
(a)


SCALED D5 L=8 ω=0
(a)


SCALED D5 L=8 ω=2
(b)


SCALED D5 L=8 ω=2
(b)


SCALED D5 L=8 ω=4
(c)


SCALED D5 L=8 ω=4
(c)

cone, which occurs twice. The first time involved spatial differentiation, determining a local context sensitive threshold for strong edges, and then finally thresholding out weak edges. The second pass involved setting the flag for strong edges, then counting the flag up the cone to determine the edge density. It is possible to use this data in a projection pass to turn on only those areas which have a high density of strong edges in a similar manner.

V. The Processing Cone as an Operating System for Image Analysis

By utilizing the parallel processing cone system as the core of a low-level image segmentation system, a number of advantages are realized. First, the cone structure represents a flexible tool for the development and testing of image analysis algorithms. It allows algorithms to be constructed, debugged, documented, and tested relatively easily on a variety of images.

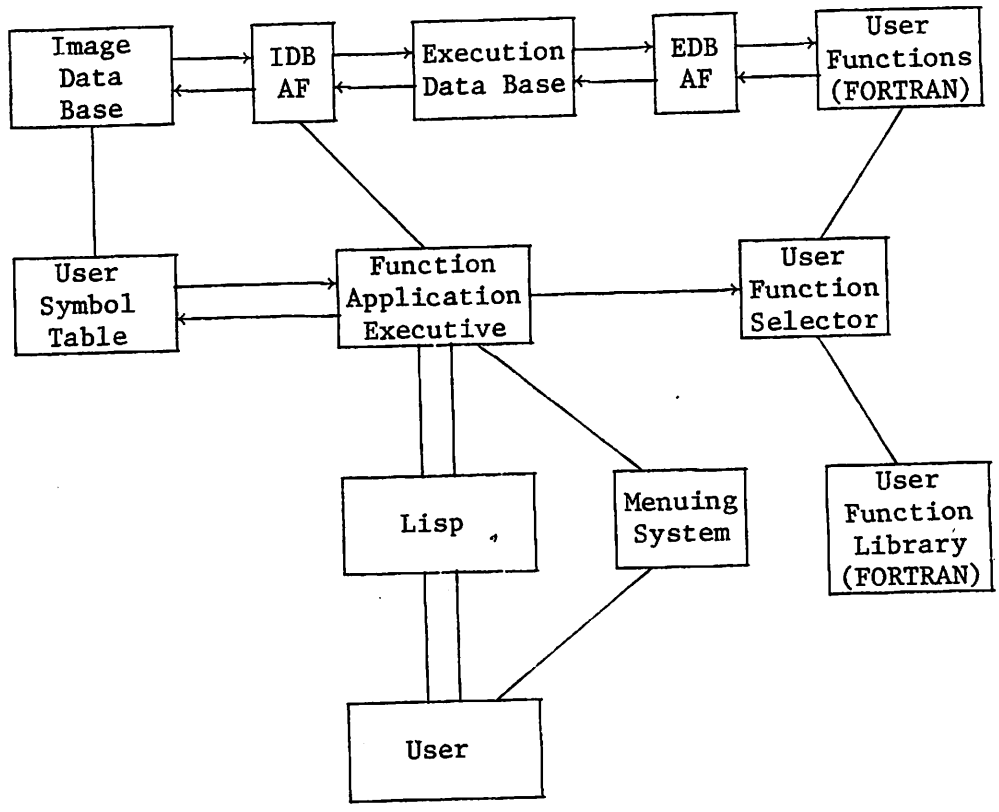
Second, when the facilities exist for defining algorithms as sequences of local parallel operations, then it is only necessary to write a series of functions, each in terms of a prototypical local window of data. All of the overhead involved in actually applying the function is handled by the system. Once a local function has been written it can be applied at any level in the cone without modification. Algorithms then become automatically structured into unit modules of computation, consistent with the tenets of good program design.

A third important characteristic of the cone involves data display and data movement. The hierarchy of resolution levels permits rapid access to intermediate results of computations while algorithms are being developed.

These levels also provide a type of focussing mechanism for applying the local function to arbitrarily smaller areas of the image. This can be achieved by placing a portion of the original data (a 2**k x 2**k subimage) at the appropriate higher level in the cone and beginning the processing at this level. It is particularly useful in facilitating program development by allowing low-computation execution of a function at a high level in the cone, and then being certain of its proper application at the low levels, with their attendant massive amounts of data. Whenever focus of computation or display of information is desired, then movement of subimages to higher levels is easily performed.

Finally, it should be emphasized that the cone is not restricted to the processing of only visual data. For example, a two dimensional histogram, perhaps of image properties, can be inserted into the cone at the appropriate level and processed as if it were an image. All of the algorithms developed for processing images, such as region growing or thresholding, then become immediately available.

A new general low-level image segmentation system based on the processing cone was recently implemented on a VAX 11/780 in our laboratory; this system is configured as shown in Figure 14. It provides a high level of user control over low level functions through the use of LISP as the control language. This choice was dictated somewhat by the requirement that the low-level system interface directly into a high-level interpretation system [HAN78b]. LISP provides the kind of interactive environment which is well suited to the development of complex control strategies for segmentation algorithms.



IDB ==> Image Data Base
EDB ==> Execution Data Base
AF ==> Accessing Functions

Figure 14. Overview of low-level image operating system.

The image data base is a hierarchical structure for long term storage of processing cone planes. The execution data base is a temporary paged data base structured for maximmum access speed. Communication between the two data bases is accomplished via the image data base accessing functions which transfer selected information between the IDB and EDB. The execution data base accessing functions communicate between the execution data base and the user functions; that is, they handle the overhead involved in returning windows of data to be processed. Since these windows may involve data from various levels and planes within the cone, they have been coded for maximum efficiency. The user requests function application (i.e. the sequence of local functions to be applied, their parameter bindings, data sources, etc.) via an interactive LISP interface. Thus, these control functions are written in LISP while the cone functions are written in FORTRAN or assembly language. The function application executive builds an appropriate environment for the execution of user function(s).

The system supports a number of data types directly accessible to the user and performs dynamic data type checking and verification via range descriptors and user specifications. The error handling system utilizes a menuing system to dynamically correct error conditions whenever possible. These characteristics, together with appropriate handling of default declarations, should make it possible for even naive users to design and run sophisticated image analysis and segmentation experiments. An extensive run time library of functions is under continuing evolution. An automatic documentation facility is built into the system; this facility maintains the processing history for each plane of data so that any plane in the image data base can be automatically reconstructed given the history. Since large amounts of data are usually generated during

algorithm development and experimentation, such a facility is crucial.

VI. Conclusions

Given the vast amount of data which must be processed during the analysis of an image, questions of efficiency assume an increased importance. These efficiency considerations imply that parallel architectures appear to be necessary for image analysis applications.

The computational structure of the processing cone is designed to facilitate the parallel processing of large arrays of visual data. It is general-purpose in that it may be programmed by defining a prototype computation to be performed on a local window (i.e. subarray) of data. In the cone, this prototype function will be applied simultaneously - and in parallel - to all local windows across the entire array. The user need only specify the definition of the function, the location of the source(s) of the data within the cone, a description of the size and shape of the local window, and the destination of the result(s) within the cone. The cone's operating system simulates lockstep computation just as if there were parallel arrays of synchronous microprocessors computing on each window; each microprocessor executes a copy of the prototype computation.

An important characteristic of the processing cone structure is its hierarchical organization into layers of decreasing resolution. There are three basic modes of processing available within the cone: reduction operations, horizontal (or lateral) operations, and projection operations. These correspond

to a flow of information up, laterally, and down the cone, respectively.

Due to the hierarchical organization into levels of decreasing resolution and the facility to transmit information down the cone (i.e., from lower resolution levels to higher resolution levels), planning-type algorithms can be constructed. Horizontal operations within the cone support algorithms of an iterative nature. Thus, as a computational structure, the cone is capable of supporting a wide range of algorithms useful in image analysis. As a conceptual structure, it provides a means of thinking about plausible parallel algorithms and as a research tool it allows the construction and testing of these algorithms.

Because of the highly organized nature of the cone, we have used it as the basis for a sophisticated image processing system. In this system, cones are dynamically assembled from a user-created library of planes. Image processing algorithms, structured as temporal sequences of local functions, are created and applied to the cone by means of a high level LISP control environment. The system provides a powerful user-oriented environment for use in image analysis research.

References

[BAL78] D.H. Ballard, C.M. Brown, and J.A. Feldman, "An Approach to Knowledge-Directed Image Analysis," in Computer Vision Systems (A. Hanson and E. Riseman, Eds.), Academic Press, New York, 1978.

[DUD73] R.O. Duda and P.E. Hart, Pattern Classification and Scene Analysis, John Wiley and Sons, 1973.

[HAN74] A. Hanson and E. Riseman, "Preprocessing Cones: A Computational Structure for Scene Analysis," COINS Technical Report 74C-7, University of Massachusetts, September 1974.

[HAN75] A.R. Hanson, E.M. Riseman, and P. Nagin, "Region Growing in Textured Outdoor Scenes," Proc. of 3rd Milwaukee Symposium on Automated Computation and Control, 407-417, 1975.

[HAN78a] A.R. Hanson and E.M. Riseman, "Segmentation of Natural Scenes," in Computer Vision Systems (A. Hanson and E. Riseman, Eds.), Academic Press, New York, 1978.

[HAN78b] A.R. Hanson and E.M. Riseman, "VISIONS: A Computer System for Interpreting Scenes," in Computer Vision Systems (A. Hanson and E. Riseman, Eds.), Academic Press, New York, 1978.

[HAN78c] A.R. Hanson and E.M. Riseman, Computer Vision Systems, Academic Press, New York, 1978.

[HAN80] A.R. Hanson, E.M. Riseman, and F.C. Glazer, "Edge Relaxation and Boundary Continuity," in Consistent Labeling Problems in Pattern Recognition (R. Haralick, Ed.), Plenum Press, New York, 1980.

[HAY74] K.C. Hayes, Jr., A.N. Shah, and A. Rosenfeld, "Texture Coarseness: Further Experiments," IEEE Trans. Systems, Man, and Cybernetics, 4, 467-472, 1974.

[KEL71] M.D. Kelly, "Edge Detection in Pictures by Computer Using Planning," Machine Intelligence, 6, 379-409, 1971.

[KLI76] A. Klinger and C.R. Dyer, "Experiments on Picture Processing Using Regular Decomposition," Computer Graphics and Image Processing, 5, 68-105, March 1976.

[LEV78] M.D. Levine, "A Knowledge Based Computer Vision System," in Computer Vision Systems (A. Hanson and E. Riseman, Eds.), Academic Press, New York, 1978.

[NAG77] P.A. Nagin, A.R. Hanson and E.M. Riseman, "Region Extraction and Description Through Planning," COINS Technical Report 77-8, University of Massachusetts, May 1977.

[NAG79] P.A. Nagin, "Studies in Image Segmentation Algorithms Based on Histogram Clustering and Relaxation," COINS Technical Report 79-15 and Ph.D. Thesis, University of Massachusetts, September 1979.

[PRI77] K. Price and R. Reddy, "Change Detection and Analysis in Multispectral Images," Proc. of 5th International Joint Conference on Artificial Intelligence, Cambridge, 619-625, 1977.

[RIS74] E.M. Riseman and A.R. Hanson, "The Design of a Semantically Directed Vision Processor," COINS Technical Report 74C-1, University of Massachusetts, January 1974.

[ROS71] A. Rosenfeld and M. Thurston, "Edge and Curve Detection for Visual Scene Analysis," IEEE Trans. Computers, 562-569, 1971.

[TAN75] S.L. Tanimoto and T. Pavlidis, "A Hierarchical Data Structure for Picture Processing," Computer Graphics and Image Processing, 2, 104-119, June 1975.

[TAN78] S.L. Tanimoto, "Regular Hierarchical Image and Processing Structures in Machine Vision," in Computer Vision Systems (A. Hanson and E. Riseman, Eds.), Academic Press, New York, 1978.

[UHR72] L. Uhr, "Layered `Recognition Cone' Networks that Preprocess, Classify, and Describe," IEEE Trans. Computers, 758-768, 1972.

UNCLASSIFIED

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

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER COINS TR 81-38	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* PROCESSING CONES: A COMPUTATIONAL STRUCTURE FOR IMAGE ANALYSIS		5. TYPE OF REPORT & PERIOD COVERED INTERIM 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Allen R. Hanson Edward M. Riseman		8. CONTRACT OR GRANT NUMBER(s) ONR N00014-75-C-0459
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer and Information Science Department 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, Virginia 22217		12. REPORT DATE 12/81 13. NUMBER OF PAGES 44
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Distribution of this document is 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)*

processing cones, image segmentation, image processing, parallel processing, segmentation algorithms, hierarchical data structures, visual information processing, scene analysis, texture analysis

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

A layered hierarchical parallel array architecture for image analysis applications, referred to as a processing cone, is described and sample algorithms are presented. A fundamental characteristic of the structure is its hierarchical organization into two-dimensional arrays of decreasing resolution. In this architecture, a protypical function is defined on a local window of data and applied uniformly to all windows in a parallel

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

UNCLASSIFIED

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

manner. Three basic modes of processing are supported in the cone: reduction operations (upward processing), horizontal operations (processing at a single level) and projection operations (downward processing). Complex image analysis algorithms are specified as temporal sequences of function applications. The cone structure forms the basis of a sophisticated operating system for an image analysis environment.