SRI International

THE PHOENIX IMAGE SEGMENTATION SYSTEM: DESCRIPTION AND EVALUATION

Technical Note No. 289

December 1982

By: Kenneth I. Laws, Computer Scientist

Artificial Intelligence Center
Computer Science and Technology Division

Program Development by:

Steven Shafer
Takeo Kanade
Duane Williams

Carnegie-Mellon University
Department of Computer Science

SRI Project 1009

The work reported herein was supported by the Defense Advanced Research Projects Agency under Contract No. MDA903-79-C-0588.

SRI International®

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

Foreword

The primary purpose of the Image Understanding (IU) Testbed is to provide a means for transferring technology from the DARPA-sponsored IU research program to DMA and to other organizations in the defense community.

The approach taken to achieve this purpose has two components:

(1) The establishment of a uniform environment as compatible as practical with the environments of research centers at universities participating in the IU research program. Thus, organizations obtaining copies of the Testbed can receive a continuing flow of new results derived from on-going research.

(2) The acquisition, integration, testing, and evaluation of selected scene analysis techniques that represent mature examples of generic areas of research activity. These contributions from participants in the IU research program will allow organizations with Testbed copies to begin the immediate exploration of applications of IU technology to problems in automated cartography and other areas of scene analysis.

The IU Testbed project was carried out under DARPA contract No. MDA903-79-C-0599. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the United States Government.

This report describes the PHOENIX segmentation package contributed by Carnegie-Mellon University and presents an evaluation of its characteristics and features.

Andrew J. Hanson
Testbed Coordinator
Artificial Intelligence Center
SRI International

Abstract

PHOENIX is a computer program for segmenting images into homogeneous closed regions. It uses histogram analysis, thresholding, and connected-components analysis to produce a partial segmentation, then resegments each region until various stopping criteria are satisfied. Its major contributions over other recursive segmenters are a sophisticated control interface, optional use of more than one histogram-dependent intensity threshold during tentative segmentation of each region, and spatial analysis of resulting subregions as a form of "look-ahead" for choosing between promising spectral features at each step.

PHOENIX was contributed to the DARPA Image Understanding Testbed at SRI by Carnegie-Mellon University. This report summarizes applications for which PHOENIX is suited, the history and nature of the algorithm, details of the Testbed implementation, the manner in which PHOENIX is invoked and controlled, the type of results that can be expected, and suggestions for further development. Baseline parameter sets are given for producing reasonable segmentations of typical imagery.

Table of Contents

Section 1

Introduction

PHOENIX is a program for segmenting an image into homogeneous regions. It combines histogram analysis with spatial analysis to find connected regions having uniform color or other properties. Small noise patches are merged with their surrounding or neighboring regions. Regions may then be further segmented by the same algorithm.

Many researchers have contributed to this segmentation technique, as documented in Section 3. The current PHOENIX program was designed by Steven Shafer and Takeo Kanade at Carnegie-Mellon University (CMU), with much of the programming done by Duane Williams and Marc Lowe. Drs. Raj Reddy at CMU and Hans-Hellmut Nagel at the University of Hamburg have guided and supervised much of the development.

The CMU PHOENIX code has been adapted for the DARPA Image Understanding Testbed at SRI International. Many of the testbed support routines provided by CMU were adapted for the Testbed by Kenneth Laws at SRI. Particular credit is due to Steven Shafer for the CI driver and related string manipulation routines, David Smith for the image access software, and David McKeown, assisted by Steve Clark, Joe Mattis, and Jerry Denlinger, for the Grinnell display software. All of this software is written in the C language.

Very few changes were required in the PHOENIX software or in the algorithm itself. The information in this document should thus be considered supplementary to the material cited in the references. User documentation provided by CMU [Smith80, Clark81, McKeown81, Shafer82] forms the basis for some sections of this report.

This document includes both a users' guide to the PHOENIX segmenter and an evaluation of the algorithm. The initial portion introduces the segmenter and describes it in general terms. Section 2 briefly describes the algorithm and the tasks for which it is appropriate; Section 3 surveys the historical development of these techniques and presents the current algorithm in detail.

The next portion of this report constitutes a users' guide. Section 4 describes the current Testbed implementation and how it differs from the original CMU contribution. Section 5 instructs the user in the mechanics of using the PHOENIX software.

The remainder of the report body summarizes the evaluation results. Section 6 describes in detail the meaning of the user-specified parameters, documents the performance that may be expected in various circumstances, and presents the results of evaluation tests. The groups of parameter values developed in this section are a significant scientific contribution. Section 7 outlines a number of suggestions for improving the algorithm and its implementation. Section 8 presents conclusions, including a brief statement of the special strengths and weaknesses of the PHOENIX approach.

Appendix A suggests alternate approaches to similar data analysis problems, and Appendix B gives the details of the connected-component extraction algorithm. An

extensive reference list provides entry points to the image segmentation literature cited in the text.

Section 2

Background

This section presents a management view of the PHOENIX program. The segmentation algorithm is briefly sketched. Typical applications and potential applications requiring further development of the algorithm are discussed, and related applications for which other algorithms are better suited are noted.

2.1. General Description

PHOENIX is a program for segmenting images into homogeneous connected regions. An input image typically has red, green, and blue image planes, although monochrome images, gradient and texture planes, and other pixel-oriented data may also be used. Each of the data planes is called a *feature* or *feature plane*.

Figure 1.1 illustrates the image segmentation process. Segmentation begins with the entire image considered to be a single region. Phoenix "fetches" this region and attempts to segment it. If it fails, the program halts and waits for further instructions; if it succeeds, it fetches each of the new regions in turn and attempts to segment it. A *segmentation queue* keeps track of the regions that are awaiting further analysis; a *terminal queue* keeps track of those that have been declared *terminal* regions.

Having fetched a region, PHOENIX computes a vector of intensity counts (a *histogram*) for each feature plane. Thresholds (or histogram *cutpoints*) are selected that are likely to isolate significant homogeneous regions in the image. A set of thresholds for one feature is called an *interval set* because each threshold defines a histogram interval extending from the previous cutpoint to and including the new one.

The most promising interval sets are passed to a spatial analysis phase that thresholds the corresponding feature plane and extracts connected components. Very small connected patches are considered noise and are merged with surrounding regions.

The feature and interval sets providing the best segmentation (*i.e.*, the one with the least noise area) are chosen. Each of the resulting segments is added to the knowledge base and segmentation map and is queued for further segmentation using the same algorithm.

This process halts when the recursive segmentation reaches a preset depth, when all regions have been segmented as finely as various user-specified parameters permit, or when the user terminates execution. The segmentation is saved, and may be reloaded and edited or continued later. The resulting region map and region description file may be used by other programs.

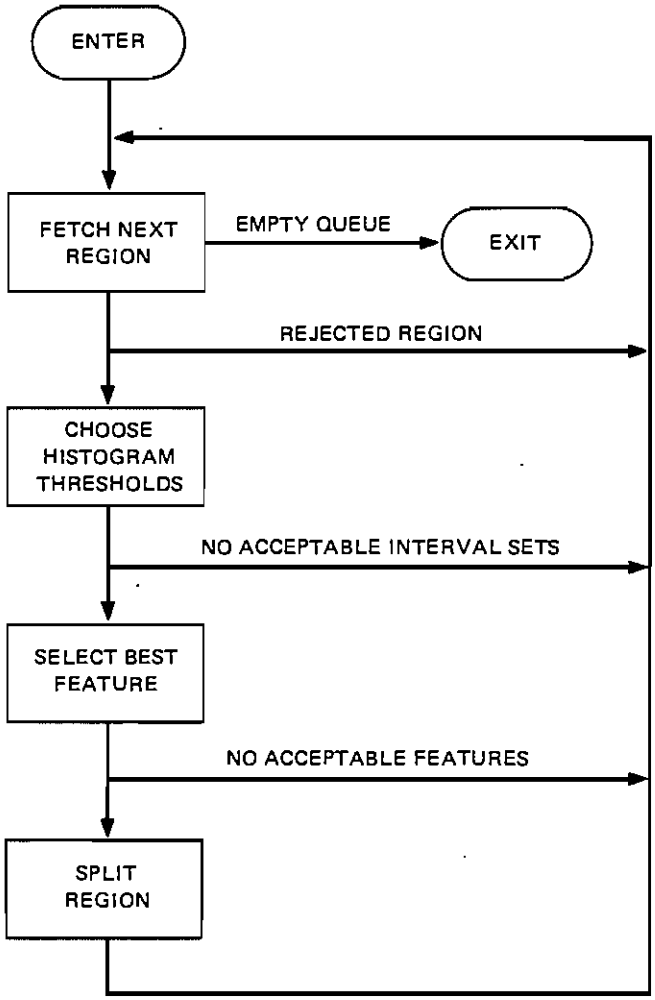

FIGURE 1.1 BASIC CONTROL SEQUENCE

2.2. Typical Applications

The PHOENIX program may be used in any application requiring that an image be partitioned into homogeneous regions. This segmentation may be useful in itself, or may be a precursor to a semantic partitioning that assigns meaningful labels to composite regions.

The initial segmentation by itself is most useful for image coding applications. Since there are far fewer regions than pixels, it may be efficient to store or transmit an image as a list of regions. This would be particularly effective in time-sequenced imagery, since only those regions that change need to be coded for each frame. The amount of compression possible depends on scene content and on the acceptable coding error. One scheme [Yan77] uses run coding to transmit the region map, or *cartoon*, and then adds a low-amplitude correction signal to fill in the details.

This same separation of the image signal may be useful in image enhancement. Enhancement within each region separately can bring out details that are otherwise obscured by illumination effects. This is similar to separate processing of low-frequency and high-frequency signal bands, but preserves edge structure better.

Region boundaries located by PHOENIX may be used to measure image blur or the transfer function of the imaging system. This information can be used in image restoration and in estimating scene depth from the amount of blur.

The PHOENIX region descriptions may be used for microscopic particle counting or for counting of nonoccluded industrial parts. PHOENIX will not distinguish touching objects, but area measurement (for uniform particles) or shape analysis (*e.g.*, [Arcelli71, Brenner77, Lemkin79, Jain80, Rutkowski81]) can make this separation. Simple size and shape descriptors may also be adequate for some medical cell classification problems.

Another application is in macrotexture analysis. Macrotextures are those that have large primitive elements forming some type of pattern. A checkerboard is a regular macrotexture; orchards, agricultural fields, and housing developments in aerial images are less regular; and tree leaves or microscopic mineral domains may be very irregular. The first step in analyzing such a texture is to identify the primitive elements, either by template matching or by segmentation [Tomita82].

Segmentation maps may also be useful in registration (*i.e.*, alignment) of two images [Ratkovic79a-c]. The two maps are first matched, giving an approximate global registration. The low-amplitude correction signals for each pair of regions are then used for precise local registration. This seems to be a good way to determine image warp coefficients, and may also be useful in tracking slowly moving objects in cluttered backgrounds.

An attempt has been made [Price76, Price78a, Price78b] to use region information for change detection in complex urban and industrial scenes. Many regions remain constant from one image to another, but others might move or change form. Region descriptions in either image that could not be matched (in shape, position, and possibly intensity) were specially flagged for user attention. The method was sophisticated enough to match similar regions at differing positions, but could not determine whether they were two similar objects or a single one that had moved.

Segmentation's most promising application, although one where it has yet to prove its worth, is in general-purpose image understanding [Fischler79, Faugeras80,

Rubin80, Ohta80b]. Segmentation and linear delineation are considered to be the first steps in feature extraction, followed by texture analysis, determination of surface orientation, and object recognition. These research topics will be discussed below.

2.3. Potential Extensions

The following applications might be feasible if PHOENIX were modified, used in a non-standard fashion, or integrated into a more sophisticated system.

Crude region knowledge may be the key to obtaining more precise knowledge: this is known as planning. Preliminary segmentation (often on a reduced image) can be used to determine which areas should be examined in more detail. Specialized structure detectors may then be applied within the regions or along the region boundaries. If the analysis is done in real time, higher resolution data may be obtained by rescanning portions of the original scene. In missile guidance, for instance, higher resolution imagery becomes available as the missile approaches its target.

Many natural scenes are better described by textured regions than by regions of homogeneous intensity. PHOENIX can be used to find textured regions if texture feature planes are provided as input. Many texture measures or transforms have been suggested [Haralick73, Carlton77, Schachter77, Mitchell78, Tanimoto78, Coleman79, Schachter79, Laws80, Lee82], but their use in PHOENIX will probably require more sophisticated feature selection and processing.

If texture-based segments are available, it becomes feasible to classify each region as to its texture type or materials category (assuming sufficient resolution). Adjacent regions that receive the same classification may then be merged to produce a better segmentation. (Note, however, that it may or may not be desirable to merge two fields that have the same crop type but different plowing directions, or two cloud patches that may be at different elevations. The merging algorithm needs knowledge about both the scene domain and the intended application.)

Segment maps may also be used as input to an object identification or intelligent cueing system. The system should be capable of recognizing objects composed of several regions. In some circumstances it may also have to guess at those which are contained within part of a region and, if possible, use additional processing to confirm the hypothesis.

2.4. Related Applications

This section describes applications that are similar to PHOENIX segmentation applications, but differ in some fundamental fashion. While the difficulties with applying PHOENIX might be overcome, other techniques would often be more appropriate.

Cueing is the initial detection of interesting objects in a scene. While cueing using a segmentation map may be possible, the effort of computing the map may be far greater than that required for threshold detection, interest-point or corner detection, unusual-pattern detection [Haralick75b, Winkler78], statistical classification, blob detection [Klein77, Deal79, Tisdale79, Danker81], prototype matching [Aggarwal78], or other techniques. Thus PHOENIX should only be used for cueing if the segmentation is required for other purposes.

Background

Object recognition is often combined with cueing when only certain objects are of interest. The problem of locating predictable signatures is best solved with matched filtering or template matching. A particularly efficient and flexible template matching method is based on the Rochester generalized Hough transform. (For a review see [Laws83].) More general object detection requires image understanding, and segmentation may be a useful preprocessing technique.

Linear delineation is the extraction of image edges, region boundaries, and elongated features. Region boundaries can be found using PHOENIX, but thin, elongated, or nonclosed features tend to be missed. A complete image understanding system will need both region extraction and linear delineation operators [Nevatia77a]. Representative techniques are described in Appendix A.

PHOENIX segments images using a recursive thresholding algorithm. The regions identified at each step are relatively uniform in one feature, and terminal regions tend to be uniform in all features. In some domains this method will fail. In extracting an illuminated sphere or cylinder, for instance, the important property is continuity rather than uniformity. Edge-based linear delineation systems are much better at segmenting smoothly-varying imagery.

Image understanding and object recognition require that many sources of knowledge be applied [Barrow75]. In particular, the system may require knowledge of sensor characteristics [Garvey76a], 3-D or physical domain knowledge [Fischler79, Fischler82], illumination and reflectance models [Horn77], semantic knowledge of likely adjacencies [Yakimovsky73a, Yakimovsky73b, Feldman74, Barrow76, Tenenbaum76a, Tenenbaum76b, Tenenbaum80], or models of likely target configurations [Price81]. It is not yet known whether segmentation should be a precursor to such analysis or should be tightly integrated with it.

Section 3

Description

This section presents the history of recursive image segmentation and a detailed statement of the PHOENIX algorithm. The historical information is intended to clarify the major issues in recursive segmentation and to provide entry points into the literature.

3.1. Historical Development

Histogram thresholding was an early segmentation technique [Prewitt66]. One or more histogram cutpoints were chosen near valleys in the intensity histogram. Connected-components analysis was then used to extract regions entirely darker or brighter than the corresponding intensity threshold level. There were difficulties, however; if an image contained many regions with overlapping histogram peaks, there would then be no obvious or useful thresholds. One solution, used by Chow and Kaneko [Chow70], was to partition an image into smaller subimages until distinct peaks appeared or the windows became so small that the histograms degenerated.

The earliest use of recursive region-splitting by histogram thresholding was for analysis of black-and-white cell images [Prewitt70]. Connected components were extracted from the thresholded image and were used for further segmentation. For other early approaches to segmentation see Appendix A.

Tsuji and Tomita [Tsuji73, Tomita73] at Osaka University used recursive region-splitting to segment macrotexture images. The shape statistics of the primitive elements were compiled into histograms. The smoothed histogram with the most distinct valleys was used for classifying the elements into two or more sets. Connected components were extracted (with some overlap allowed), and very small regions were merged with their neighbors, if possible. Boundaries of the regions were computed and compared with scene models, and those regions not corresponding to known object types were scheduled for further partitioning.

Robertson *et al.* [Robertson73] at Purdue University pursued the notion of histogram thresholding for segmentation of multispectral scenes. They also used recursive segmentation along rectangular boundaries, foreshadowing later development of the quadtree segmentation representation.

Several researchers investigated segmentation of natural textures where primitive elements could not be extracted. Kasvand [Kasvand74] used a primitive constant threshold with texture measures based on local standard deviation, gradient, second derivative, and other features. Zucker *et al.* [Zucker75] computed response to a spot detector at every point in a scene and tried to segment the resulting histogram. Satisfactory results could only be obtained if the spot detector was approximately matched to the texture coarseness and if nonmaximal suppression was used to reduce blurring due to the measurement window size. These researchers did not use recursive segmentation.

Description

Other researchers were attempting to segment color imagery using multidimensional histogram analysis. Tenenbaum *et al.* [Tenenbaum74] at Stanford Research Institute (now SRI) projected the three-dimensional color space onto the chromaticity plane and segmented on pixel hue. Even with thresholds based on prominent scene objects [Tenenbaum75], there were difficulties with overlapping hue distributions in landscape scenes and with color-coordinated decor in indoor scenes, as well as with an abundance of small texture regions. Neither texture features nor recursive segmentation were used.

Ohlander [Ohlander75] at Carnegie-Mellon University adapted the Tsuji algorithm for color images by computing histograms of three color features (RGB) and six color transformations (YIQ and HSD). A simple texture feature was also computed to identify microtexture regions. These features were used for recursive segmentation within arbitrary region boundaries. At each stage the histogram with the most prominent isolated peak was chosen for segmentation. Pixels related to the peak were then extracted and represented by a bit mask. (All those with higher or lower feature values were represented by the complement of the mask over the original region.) High-resolution, and hence large pictures and long processing times, was needed to accurately isolate textured regions and locate objects in natural imagery. Interactive thresholding inside textured areas was also necessary to segment a city skyline scene.

Schachter *et al.* [Schachter75] at the University of Maryland were also studying color image segmentation at this time. They chose to store the full three-dimensional histogram as a binary tree. They report that a leaf node is needed for every five or ten pixels in the image. (This would increase if texture measures were included.) Clusters in the tree were found by a single-linkage (or chained nearest-neighbor) algorithm. Nonrecursive segmentation was then done by assignment of pixels to the cluster classes. A similar method was later used for texture segmentation of monochrome imagery [Schachter77].

Kender at CMU analyzed the color transformations used by Tenenbaum and Ohlander; he concluded that inherent singularities and quantization effects were capable of introducing false peaks and valleys [Kender76, Kender77]. This effect is particularly noticeable in the hue feature, but also affects saturation and other normalized chromaticity coordinates; he recommended that saturation only be used in regions of high luminance, with hue used only in high saturation as well. (Note that most natural imagery has low to moderate saturation.) The YIQ transform used in color television transmission was found to have fewer problems, although its usefulness in segmentation was not evaluated. Kender also proposed an improved computational algorithm for hue.

Mui *et al.* [Mui76] brought together iterative segmentation and spatial analysis for the segmentation of blood cell images. An initial threshold segmentation was used to determine scene parameters and initial histogram cluster centers. Refined clusters were then found in the "color-density" histogram, and these were mapped back to the spatial domain. Similar techniques have been used in many medical image-analysis systems [Aggarwal77, Cahn77].

A key concept of later segmentation systems is *planning*, or heuristic guidance. Planning was introduced by Kelly [Kelly70] in the recognition of human images. A reduced image was first used to find the face or body outline, then individual features were sought in higher-resolution imagery. *Ad hoc* rules were used to identify the mouth, eyes, pupils, and other facial features. Kelly later applied planning to edge detection [Kelly71]. Planning, or hierarchical image feature extraction, was also the

foundation of other pyramid or processing-cone systems [Uhr72, Harlow73, Hanson74, Tanimoto75, Klinger76, Levine76, Dyer81].

Price at CMU brought together recursive region-splitting and planning [Price76]. His PLAN program for segmentation and symbolic matching used a refinement of the Ohlander algorithm on a reduced image, then applied the same thresholds within a slightly enlarged mask area in the full-resolution image. This two-stage approach reduced segmentation time by a factor of about ten. The color features used were a modification of Ohlander and Kender's YIQ and HSD, although LANDSAT spectral band features were also used. Price introduced several texture measures for monochromatic segmentation and added a spatial smoothing step to remove small holes from the binary masks. Less human interaction was required during histogram analysis, region extraction, and database maintenance than for Ohlander's system.

Aggarwal *et al.* [Underwood77, Ali79, Sarabi81] at the University of Texas have used a different approach for the segmentation of color images. They have mapped the image data into a three-dimensional intensity and chromaticity histogram. The bivariate marginal histograms may be displayed for interactive cluster identification, or a binary tree structure similar to that of Schachter *et al.* may be used for automated cluster identification. A version of the system used discriminant analysis to detect diseased citrus trees in infrared color imagery. An advantage of the chromaticity coordinates is that shadow regions in the image may often be easily identified.

Ohta *et al.* have further investigated color transforms for recursive segmentation [Ohta80a, Ohta80b]. They computed color histograms using the Karhunen-Loeve color transform – an expensive method because the transform is different for each region. Ohta found that the transform principal axes tended to cluster around

$$I_1 = red + blue + green$$

$$I_2 = red - blue$$

$$I_3 = 2\,red - (green + blue)$$

and recommended that these features be used. (The second and third features may be negative, so that either an offset is necessary or the segmentation code must be able to handle negative pixel values.)

Ohta's transform is similar to the YIQ system and to the opponent color process recommended by several authors [Sloan75, Nagin78]. The transform is linear, and hence avoids the instabilities that Kender found in saturation, hue, and normalized chromaticity coordinates. Nagin expressed some theoretical reservations about his own opponent features, but concluded that they "consistently provided more discrimination than the original RGB data."

Nagin also explored the use of "conservative" histogram thresholding (*i.e.*, suppressing doubtful classifications) combined with region growing, and showed how the image segmentation algorithm itself could be used for segmentation of two-dimensional histograms [Nagin77, Nagin78]. Other two-dimensional histogram analysis systems have been built by Milgram *et al.* [Milgram79, Milgram80] to segment monochrome images using pixel edge-strength in addition to intensity.

Meanwhile, work on recursive segmentation has continued at CMU. The current PHOENIX program is a VAX 11/780 implementation of Shafer and Kanade's KIWI program for the PDP 11/40. A related system named MOOSE [Shafer80] is being studied

at the University of Hamburg for symbolic motion analysis. The algorithm used in these systems is described below. The use of multiple histogram intervals, spatial analysis look-ahead, and the interactive control system are major innovations incorporated into PHOENIX.

3.2. Algorithm Description

Image segmentation reduces a pixel array to a map or list of significant regions. This greatly reduces the number of entities to be dealt with while increasing our knowledge about the image. (The increased knowledge, or information, may be measured by the reduced number of bits required to code the image. More importantly, the extracted segments are usually related to objects in the imaged scene.)

It is difficult to talk about the complexity of the segmentation task without discussing particular techniques, although this has been attempted [Gurari82]. For a survey of statistical image models for classification and segmentation see [Rosenfeld79].

There are many approaches to image segmentation, and each has its domain of applicability. Edge-based methods attempt to derive closed regions from linear discontinuities. Region-growing methods extend small homogeneous regions by incorporating neighboring pixels or regions. Region-splitting (or thresholding) methods subdivide initial regions by identifying more homogeneous subregions. All of these techniques are discussed further in Appendix A. The following describes the PHOENIX algorithm for image segmentation.

3.2.1. General Approach

The PHOENIX algorithm is a region-splitting technique. It has the advantage that a partial segmentation is meaningful, and only those regions satisfying higher-level criteria need to be considered for further segmentation.

A scene is assumed to be composed of numerous connected regions, each of which is approximately uniform in texture and, if untextured, in all of its other pixel properties. The luminance image of an untextured scene then resembles a mosaic of flat-topped "mesas." These regions may be related to portions of objects, to whole objects, or to clumps of objects. (We will temporarily ignore shadows, occlusions, and other complications.)

The segmentation algorithm must identify image regions that correspond to such scene regions. The job is complicated by imaging blur, spatial and intensity quantization, and other artifacts of the imaging process. The most serious problems, however, arise when the scene contains sloped facets [Haralick80] or continuous gradients that violate the assumed mesa model.

PHOENIX finds uniform regions by recursively splitting nonuniform ones, beginning with the whole image, into smaller regions. (See Appendix A for a discussion of splitting techniques.) The connected components associated with each intensity slice are then extracted. This process is not necessarily cheap, but there is evidence that it is well-suited to a parallel architecture such as the human visual system. Price [Price76] lists counts of machine operations required to perform many of the recursive segmentation steps. The amount of computation per region is nearly independent of the number of subregions found, so there is a bonus if the technique finds many subregions in a single pass.

Description

One way to locate many regions is to analyze the histograms of many features. PHOENIX and the Ohlander-type segmenters typically use three independent color features per pixel, plus one or more texture features. (For monochrome imagery, only intensity and the texture features are available.) Although joint histogram analysis is possible, it is of the same order of difficulty as the original image segmentation problem. PHOENIX opts for simplicity by analyzing only the one-dimensional marginal (or single-feature) histograms, augmented by one-dimensional histograms of linear or nonlinear feature combinations.

Each pass of the Ohlander/Price algorithm found segments related to a single peak in a single marginal histogram. The PHOENIX program is able to use multiple histogram intervals to increase the number of regions found in one pass, although typical operation uses only one threshold per feature in order to minimize noise and segmentation errors.

3.2.2. Color Features

Although color transformations are not strictly a part of the PHOENIX program, they are fundamental to its theoretical basis and to its typical operation.

Color features are needed when two regions to be distinguished have similar intensity (and texture), but different hue or saturation. Even if the regions are not adjacent, their intensity histograms will overlap and prevent discrimination. Hue, saturation, or other color features may be used to break the deadlock.

Color features for image processing research are typically generated by scanning a color photograph through color filters (*e.g.*, Wratten filters 25, 47B, and 58) to get red, green, and blue feature planes. Real-time systems often use an electronic color camera to generate YIQ features, which correspond roughly to perceptual brightness, cyan *vs.* orange, and magenta *vs.* green. ('I' stands for *in-phase*, 'Q' for *quadrature*.) The two color systems are equivalent, and we shall henceforth assume that the primary input is in the RGB coordinates.

Each color system constitutes a three-dimensional color space, that can express most of the colors perceived by humans. (The full detailed spectrum that, *e.g.*, astronomers and physicists depend upon has been lost, just as it is in the human visual system.) A few purples and highly saturated colors are not precisely representable, the colors recorded with different films or cameras may differ, and digital quantization limits the fineness of color distinctions, but the three-component representation is adequate for most purposes.

Typical quantization is eight bits per color axis, or 16.8 million cells for an entire three-dimensional histogram. Repeated cluster analysis in such a histogram is not attractive, although nonhistogram methods of multidimensional pattern recognition are available. The PHOENIX package instead uses an adaptation of the one-dimensional histogram segmentation developed by Tsuji, Tomita, and Ohlander.

Any one-dimensional histogram is equivalent to a projection of the three-dimensional data onto a line (or curve) through the color space. If the scene contains many regions, their histogram peaks are likely to overlap and obscure any useful details in the composite histogram. The overlap is different for projections at different angles, and it is often possible to isolate peaks from some of the regions by using many different projections.

Description

Several projections, or transformations, were discussed in Section 3.1; many others are possible. The authors of PHOENIX have generally stayed with Ohlander's choice of RGB, YIQ, and HSD (hue, saturation, and intensity) projections, although they note the instabilities of the HSD system near the D axis [Shafer82]. (The HSD system is also known as the HSI or IHS system. The symbol D is used here to avoid confusion with the YIQ system. It comes from *density*, a measure of the amount of silver deposited at a given point in a photographic negative.)

The color transforms are generally computed by the method of Kender [Kender76, Kender77]. The YIQ coefficients are

$$Y = 0.509R + 1.000G + 0.194B$$

$$I = 1.000R - 0.460G - 0.540B + M$$

$$Q = 0.403R - 1.000G + 0.597B + M$$

where M is the highest possible intensity value in the original RGB features, typically 255. These formulas have been linearly scaled to maintain quantization accuracy (via the unit coefficient). The addition of M is simply for convenience in digital representation. (The Q feature can be negated before adding M to better match the green gun on a color monitor.)

The HSD coordinates were introduced by Tenenbaum *et al.* [Tenenbaum74] to mimic human color perception. Briefly they are

$$H = \arccos \frac{(R-G)+(R-B)}{2\sqrt{(R-G)(R-G)+(R-B)(G-B)}}$$

$$S = m\left(1 - 3\frac{min(R,G,B)}{R+G+B}\right)$$

$$D = \frac{(R+G+B)}{3}$$

where m is the maximum desired saturation value. Hue is normalized by subtracting it from 2π if $B>G$, and some care must be taken in rounding the values near 2π if the number is quantized. Note that these formulas contain singularities due to division by zero: Kender recommends detecting these cases and treating them as special values. See [Kender76, p. 35] for a computational algorithm.

3.2.3. Texture Features

Only the intensity feature (D or perhaps Y) is available for monochrome imagery. This is occasionally adequate for segmenting simple scenes with large objects (as in cell counting [Prewitt70] or some types of industrial inspection), but aerial scenes usually show so many regions that the composite histogram is unimodal. Recursive segmentation can only proceed by using additional texture features or special control strategies (see Section 3.2.6).

Structural texture features can be used [Tsuji73, Tomita82], but the PHOENIX program is best adapted for statistical texture features that can be measured at each point. There are many such measures. Ohlander used a simple Sobel-edge "busyness" feature to identify textured regions in color imagery. Price used local edge density, variance, and range to segment aerial and side-looking radar imagery. (He suggested that local minimum or maximum pixel values could be used to distinguish some regions.) Fourier and other spatial transforms are popular [Pavlidis75,

Tanimoto78]. Local gradient or edge strength could also be used, although the histogram analysis must be more sophisticated [Milgram79, Milgram80].

Ohlander used texture only to remove busy regions from further consideration by the color segmentation system. Price also used texture this way, but was able to segment monochrome images using texture features in place of color features. PHOENIX carries this integration even further by using a limited form of look-ahead: at each step only those features producing "clean" spatial segmentations are kept. Thus texture and color features may be used together. (A more intelligent system would understand the nature of each feature plane, and failure of a color feature to provide compact regions would activate a texture analysis subsystem. This has not yet been tried.)

3.2.4. Histogram Analysis

It was stated earlier that each region in a scene is modeled as a uniform patch in the image. Such a model implies that the histograms should contain only sharp spikes. A more appropriate model, allowing for some texture and imaging effects, is that each region produces a noisy Gaussian peak in the histogram.

Methods do exist for decomposing a function into Gaussian peaks. This is known as the mixture density problem [Wolfe70] and is important in information theory, statistics, chemistry, and other fields. Very little of this theory has been applied to image processing [Chow70, Rosenfeld76b, Postaire81]. PHOENIX is able to use its spatial knowledge to avoid the difficulties of these methods, although at the cost of making some errors in threshold placement. These errors cause the break-up of some small regions and shifting of region boundaries on others.

Ohlander and Price used a hierarchy of heuristic rules for selecting the most prominent peak within a set of histograms [Ohlander78, Price79, Nevatia82]. The peak was delimited by two thresholds that defined an intensity interval and its complement. PHOENIX uses similar heuristics, but concentrates on the valleys (*i.e.*, local minima) in the histogram set. Usually a single valley, resulting in one threshold and two intervals, is selected for each feature. Spatial analysis is then used to select the best threshold/feature combination. Using only one threshold per pass reduces the chance of segmentation errors, although it does increase the number of passes required.

The PHOENIX histogram analysis uses region growing instead of recursive segmentation. A histogram is first smoothed with an unweighted moving average. It is then broken into intervals such that each begins just to the right of a valley (*i.e.*, at the next higher intensity), contains a peak, and ends on the next valley. A valley is considered to be the right *shoulder* of its left interval and the left shoulder of its right interval. The leftmost and rightmost intervals always have exterior shoulders of zero height.

A series of heuristics is then applied to screen out noise peaks. Each test is applied to all the intervals in the histogram (providing there are enough intervals for the test to be meaningful – two for some tests, three for others). When an interval is eliminated, it is merged with the neighbor sharing the higher of its two shoulders. The screening test is then applied again to the merged interval; previous tests are not reapplied.

Peak-to-shoulder ratio is tested first. An interval is only retained if the ratio of

Description

peak height to the higher of its two shoulders, expressed in percent, is at least as great as the **maxmin** threshold. (See Section 5.1 for more about this and other user-supplied thresholds.)

Peak area is then compared to an absolute threshold, **absarea**, and to the **relarea** percentage of the total histogram (or region) area. Only peaks larger than these thresholds are retained.

The intervals surviving to this point should be reasonable candidates, and it is reasonably safe to use global histogram descriptors in the test conditions. The second-highest peak is now found, and peaks less than a percentage, **height**, of it are merged. The lowest (interior) valley is then found, and any interval whose right shoulder is more than **absmin** times this is merged with its right neighbor. (The parameter seems to be misnamed since the criterion is relative rather than absolute.)

A final screening is made to reduce the interval set to **intsmax** intervals. This is done by repeatedly merging regions with low peak-to-shoulder ratios until only **intsmax**−1 valleys remain.

A score is also computed for each interval set. This score is the maximum (apparently MOOSE used the minimum) over all intervals of the function

$$1000\ \frac{\textit{peak height} - \textit{higher shoulder}}{\textit{peak height}}$$

This formula assigns the maximum score to an interval set containing a peak with shoulders of zero height.

3.2.5. Spatial Analysis

PHOENIX next chooses features (and corresponding interval sets) for spatial evaluation. The best **isetsmax** interval sets will be chosen, provided that each has a score of at least **absscore** and at least **relscore** percent of the highest interval set score.

Each selected interval set is then tested for segmentation quality. The histogram cutpoints are applied to the feature plane as intensity thresholds and connected components are extracted. (See Appendix B for the extraction algorithm.) Applying the thresholds introduces *segmentation noise* of three kinds: border placement errors, small *noise patches* that do not correspond to scene objects, and thin *necks* connecting patches that should be separated.

Border placement errors occur when the threshold separating two patches is influenced by histogram contributions from other nonadjacent patches. The effect can be so severe that small regions are split apart and/or absorbed into neighboring regions. This can be combated by conservative thresholding [Milgram79] or by some type of post-analysis using the statistics of only the two regions involved. (See [Milgram77] and [Milgram80] for methods of combining edge evidence with histogram analysis.) PHOENIX currently ignores such errors.

Price's PLAN program used a fast (but still time-consuming) spatial smoothing step to eliminate noise regions and connecting necks. Unfortunately the method also rounded corners and straightened thin diagonal objects. A more intelligent method would need to determine which pixels were noise regions or necks and to alter only those.

Description

The PHOENIX spatial analysis is able to deal with noise regions, but not with connecting necks. (Large regions joined by a neck will usually be split in a later segmentation pass.) PHOENIX calls a subroutine to determine whether a connected patch is a noise region or a true region. At present this subroutine performs only an area test, with patches smaller than **noise** pixels considered to be noise regions.

After each feature has been evaluated, the one producing the least total noise area is accepted as the segmentation feature (providing that the noise area is less than a percentage, **retain**, of the total region area). The subregions obtained with that interval set are added to the segmentation record and the next segmentation pass is scheduled.

3.2.6. Control Strategies

PHOENIX uses a spatial analysis look-ahead to improve the selection of a segmentation interval set, just as modern chess-playing programs use dynamic evaluation to validate moves that seem good to a static evaluator. Spatial analysis improves on selection by the interval set score about 40% of the time [Shafer82], although the order in which features are selected may have little effect in many of these cases.

Several other high-level control strategies have been proposed to overcome specific problems. Ohlander and Price, for instance, used ordering of texture and color feature sets to guarantee that some features would be tried before others. PHOENIX has no such ordering because the spatial analysis rejects any inappropriate feature that would cause the breakup of a region. The program developers recognize, however, that such methods might save computation time or be otherwise useful; they have added such a facility to a later version of PHOENIX than is documented here [Shafer82].

Two methods of reducing computation time are *planning* and *focusing*. Planning was discussed in Section 3.1. It involves use of thresholds and region masks derived from reduced images to speed segmentation of full-resolution images. PHOENIX does not incorporate planning.

Focusing is the use of interest operators, motion detectors, or higher-level knowledge to crop the image around objects of interest [Shafer80]. This concentrates expensive resources on appropriate tasks, but does run the risk of missing unexpected objects in the scene. PHOENIX does not include an automatic focusing mechanism, but the user may control which regions of the image are to be segmented further. The user may also "prune" regions where the subregion structure turns out to be uninteresting.

Another difficult problem is the initiation of action when the original set of features is insufficient to identify a usable threshold. This often occurs in monochrome segmentation, because the single luminance feature has insufficient degrees of freedom for separating the overlapping peaks of many small regions. Texture features also tend to be unimodal unless the scene contains large areas of distinctive texture (such as agricultural fields [Keng77a]).

Color features are typically multimodal, making it easy to begin segmentation of even large scenes. Some possible explanations for this phenomenon are:

Description

* Co-evolution of natural visual systems and of the environment they operate in may have produced a teleological segmentation of natural scenes into colored areas corresponding to functional entities. (Note that color is nearly always absent in caves and in deep ocean environments.) Man has continued this trend in the construction and decoration of technological artifacts. While colored objects are "intended" to contrast with their backgrounds, natural textures are more often accidental or intended for concealment. Further, since our understanding of texture is poorly developed, the texture measures we are using may have little discriminating power to begin with.
* Color is a point property, and can be measured very precisely. Texture is a local neighborhood property, and current methods of computation inherently blur the scene. If texture is measured over 15x15 windows, a single pixel from a different texture source contaminates the measured texture at 224 locations around it. The measurement windows of any two adjacent pixels have 93% overlap. This tends to smooth the texture histograms. For methods to combat this (by nonmaximal suppression and by choice of window size) see [Zucker75].
* Perceptual color is a three-dimensional space. Projecting it to a one-dimensional space (*e.g.*, luminance) often destroys cluster separability; multiple projections must be used to retain sufficient degrees of freedom. Texture space may well have dozens of dimensions, and we have been measuring it along too few axes for good separability.
* Color features are measured through "leaky" filters that permit some response to other colors. Consider a picture of a red flower against a green background. If the color filters were ideal, the red histogram would have a single peak representing the flower and the green histogram would have a single peak due to the background. Only by blending the two histograms, as occurs now with our broad filters, could histogram analysis find a starting point. Many of our texture measures are designed to be orthogonal, and it may similarly be necessary to use linear and nonlinear combinations of texture features to augment their effectiveness. (Combinations of texture and color may also be possible [Rosenfeld80].)
* Grahame Smith of SRI has suggested that multiple filtering may also play a role. Imagery for image understanding research has typically passed through at least two filtering processes during capture on film and subsequent digitization. The combined effect may introduce deeper histogram valleys than were present in the original scene. Texture measures are not subject to these influences.
* As Kender has pointed out, quantization and aliasing in digital transformations introduce false peaks and valleys into an otherwise uniform histogram. The effect on natural scenes has not been fully studied, but it is very likely that hue and perhaps saturation exhibit these effects. Various noise sources, particularly the picket fence effect of digital contrast improvement, may also introduce sharp peaks and valleys into the color histograms. Texture measures are often computed using floating-point arithmetic, and so avoid these effects.

Whatever the reason, luminance and texture features alone are often too unimodal to initiate segmentation of a large region. A higher-level control strategy is needed to get the segmenter off dead center. Once it has broken the image into regions, there is often enough peak separation to continue to a reasonable segmentation.

Description

Price solved this problem using *partitioning*. The image was arbitrarily broken into smaller sections, and the histogram of each was computed. Each histogram was treated as the histogram of a feature over the whole image. Thus if a peak was found in the histogram of one image section, it was used to threshold the entire image. PHOENIX has no such mechanism, although its spatial analysis step would make such an action less dangerous.

Another, much simpler, heuristic would be to gradually weaken all thresholds until some histogram became segmentable. In the limit this would require that a feature threshold be arbitrarily chosen. Although this sounds crude, it may be exactly what is currently happening in the color domain. If the arbitrary threshold proved effective, any inappropriate segmentation that it caused could later be undone in an editing step.

Section 4

Implementation

This section documents the SRI Testbed implementation of PHOENIX, which is very little changed from the original CMU implementation. It is intended as a guide for system maintainers and for programmers making modifications to the PHOENIX system. The terms used in this section may be a little cryptic: they are either defined elsewhere in this report or come from the supporting operating systems.

The SRI Testbed uses the EUNICE operating system, which is a Berkeley UNIX[1] emulator for VAX computers using DEC's VMS operating system. EUNICE was developed at SRI to permit simultaneous access to UNIX and VMS software and system services, and to implement improvements to UNIX such as significantly faster image I/O. EUNICE is now a commercial product maintained by The Woolongong Group in Mountain View, California.

Some of the directory and file names were truncated for compatibility with an early EUNICE environment. (This is no longer necessary, although it may still be desirable for VMS compatibility.) The main program, subroutines, and help files are in directory /iu/tb/src/phoenix. Major subdirectories are:

demo	- standard parameter sets;
display	- display routines;
do	- phase control scheduler;
flags	- flag parsing routines;
help	- help system text files;
include	- macro definition files;
k1	- command operators; ,
main	- PHOENIX main program;
misc	- I/O and misc. functions;
new	- new region maintenance;
queue	- queue maintenance routines;
v	- scheduling control functions.

To compile the PHOENIX program, just connect to this directory and type "make". You may type "make -n" to see what will happen if you do this. Additional options are documented in the header of the makefile.

Other major functions of the PHOENIX package have been moved to the /iu/tb/lib/visionlib histlib, intervlib, patchlib, and polygnlib directories because these subroutines may be of use to other programs. There is currently no documentation on these routines other than that in the source code headers.

Source code and help files for the CI driver are in /iu/tb/lib/cilib. For extensive documentation type "man ci" or run "vtroff -man /iu/tb/man/man3/ci.3c". The CI driver

[1]UNIX is a Trademark of Bell Laboratories.

uses command-line parsing routines in cilib/cmuarglib and in /iu/tb/lib/sublib/asklib; both of these may someday be replaced by the Testbed argument parsing routines in sublib/arglib.

Other utility routines contributed by CMU have been distributed to /iu/tb/lib/dsplib/gmrlib, /iu/tb/lib/imglib, and /iu/tb/lib/sublib, and are documented in /iu/tb/man/man3. Some of these have been modified or rewritten for the Testbed environment; the image access code, for instance, reads Testbed image headers as well as CMU image headers. Output map files are now created with Testbed headers.

One modification to PHOENIX was in the manual decision logic of the *fetch* phase, as controlled by the 'm' flag. The original code displayed the next region *after* the user had made a choice; the Testbed version now displays it before the choice is made.

SRI has also added a detailed display of the threshold selection heuristics during the *interval* phase. This is turned on by the 'H' flag, and takes effect if rundisplay is specified. Each feature histogram in turn is displayed in white. The thresholds before a heuristic takes effect are shown in blue; those remaining after the screening are shown in green. The user types a carriage return to proceed with the next heuristic. This integrates well with the retry facility for redoing the *histogram* or *interval* phases.

The original PHOENIX code assumed an upper-left origin for image and graphics display. CMU provided the conversion macros for changing image display to a lower-left origin as used on the Testbed. Since Testbed image format is also the inverse of the CMU format, the macros had the effect of displaying images right side up but in the lower-left corner of the screen.

Unfortunately the associated graphic displays did not use the coordinate conversion macros, and could not easily be made to do so. (Maintaining the original layout would require that all histograms and text be displayed upside down.) We have moved or interchanged some of the graphic components instead.

The original code limited rundisplay to images of 111 rows or fewer because of the multiquadrant display layout. With our altered layout, it was possible to extend this to 256 rows, although there is still a minor problem with text overwriting the images.

Modifications were needed in two of the threshold selection heuristics. The maxmin heuristic was rejecting nearly all thresholds if either of the outermost histogram bins held a large value; this was fixed by defining the outermost interval minima to be zero rather than the bin values. The absmin heuristic was rejecting all thresholds at nonzero bins if the global minimum was zero; this was fixed by clipping the global minimum to be at least one.

Several heuristic thresholds were permitted to take meaningless values (*e.g.*, **relarea** > 50 and **intsmax** = 1). These limits have been tightened. The **hsmooth** variable was originally limited to 20; we have extended it to 100. Several other arbitrary limits have been extended and default parameter values have been changed to the moderate values developed in Section 6. Some of the original and new defaults are:

splitmin:	1	-->	40
hsmooth:	1	-->	9
maxmin:	200	-->	160
absarea:	20	-->	10

Implementation

relarea:	5	-->	2
height:	70	-->	20
absscore:	550	-->	700
relscore:	85	-->	80

Other minor changes included reducing debugging printout in clrscreen.c, adding printout of rejected feature set scores as well as accepted ones, changing the colors of some display elements, and fixing a bug in display of monochrome images. We have not yet removed a restriction against using red, green, or blue feature planes without using all three as segmenter inputs.

Several PHOENIX demonstrations have been set up in subdirectories of /iu/testbed/demo. The *chair* directory contains the original demonstration contributed by CMU: segmentation of an orange chair from a white background. The *show9* command shows the original red, green, and blue feature planes and the hue, saturation, intensity, y, i, and q feature planes computed with SRI's *convert* program. The *demo* command runs the interactive segmentation using only the red, green, and blue features. You may restore demo.ckp file to see the finished segmentation produced by shell file ckp.csh [using the original CMU parameter defaults].

The *portland* directory also has a *show9* command and a *demo* script that loads the final results for segmentation of the 512x512 *portland* image using strict and then moderate heuristics. You may run this script and then browse using the 'history', 'describe', 'display', and other informational commands. (Use the '*' and 'help *' commands to find out what is available.) You may also restore the mild.ckp file to see the effect of the mild (permissive) heuristics. This directory also has a *skydemo* script designed to show off PHOENIX as a skyline finder: just type 'Control-Z' or 'exit' to step to successive results.

The *demo* command in the *skyline* directory is very similar. It shows the results of segmenting a reduced *bishop* image using strict, then moderate, and finally mild heuristics.

Section 5

Program Documentation

This section constitutes a users' guide to the PHOENIX package as it is implemented on the SRI Image Understanding Testbed. As with any reference manual, it has occasionally been necessary to refer to terms before they are defined and discussed in detail. A preliminary scan through the section may be helpful on the initial reading. Additional information is available on-line, as described below.

5.1. Interactive Usage

The program requires one or more registered picture files as input. These typically represent red, green, and blue image planes, and perhaps intensity, hue, saturation, and other transformations as well. Texture planes may also be provided; they are not computed by PHOENIX. The program produces a region map, which is a picture file having 16-bit region numbers as the pixel values.

The set of input pictures is specified by a template and a *-f* flag followed by a set of feature keywords. For example:

```
phoenix /iu/tb/pic/chair/4.img -f red green blue ...
```

specifies that the files 4red.img, 4green.img, and 4blue.img in the directory /iu/tb/pic/chair are to be used as the input pictures.

Once started, the user typically sets some flag values to control the scheduling process and display options, then issues the *segment* command. This begins segmentation of the image, which will continue to the halting point specified by the A, B, and C flags (see below). The user may also interrupt processing with the 'Control-C' key, and may then examine or alter the current status.

Segmentation is normally done by depth level, with all regions at one depth segmented before any of their subregions are processed. The segmentation of a single region at a single depth constitutes a *pass*, and consists of a region-dependent sequence of various *phases*.

5.1.1. Invocation Options

The following options may be specified on the initial command line. All other commands must be typed in interactively or piped in using a batch script. (See Section 5.2.)

-e Echo commands as they are read from a file. If this is not specified, initialization commands will execute invisibly. Interactive script commands invoked with '<' are not affected by this flag.

-f feature ...
Feature plane specifications as illustrated above. See [Clark81] for a full description of the picture naming system.

-i file
-I file
Read initialization commands from *file* before accepting commands from the terminal. Only one such file may be specified. The *-I* form exits without accepting terminal input.

-o file
-O file
This mandatory parameter specifies the output map file. The *-o* form will create a new file; if it already exists, PHOENIX will ask whether you want to overwrite it. The *-O* form will open an existing map file.

-r region#
-R region#
These two parameters are used with existing (*-O*) map files to specify the current (*-r*) region for further segmentation and the highest (*-R*) region number to be updated.

-s Execute a single *segment* command and then exit. This is usually combined with initialization commands (see *-i*) to set 'flags = ABC' for continuous segmentation and perhaps 'flags = q' to squelch tty output.

5.1.2. Interactive Commands

PHOENIX is implemented using the CI command interpreter. Type '?' for a list of commands that are available from the CI driver itself. Most of these have to do with interactive help facilities.

The following PHOENIX commands can be entered from the keyboard at any stopping point during the session. Arguments that are not specified as part of the command will be requested.

abort
Terminate segmentation of the current region. The region is put back at the head of the segmentation queue.

checkpoint datafile mapfile
Save the current state of the segmentation. Global variables are stored in *datafile* and a copy of the region map is written to *mapfile*. Histograms, interval sets, and other temporary data structures are not saved.

If you specify a nonexistent directory, you are asked whether it should be created. If either of the specified files already exists, you are asked whether you want to overwrite the file or specify a new one. Simply typing a carriage return will abort the command.

The datafile contains a reference to the mapfile name, so you may not use operating system commands to rename this file. (You may move both files to a new directory as long as the same name is used.) The best way to rename checkpoint files is to restore them and then write them out again under new names.

clear {s|t}
Remove all regions from a specified queue. You are asked whether you really want to do this.

describe [type] [identifier]
Describe a data structure. You must specify the type of object and which one you want. Currently you can ask about regions, histograms, and interval sets. Regions are identified by number; histograms and interval sets by feature. You may specify 'all' features if you wish.

display [type] [identifier]
Display an object or data structure. You may display the image, particular regions, or the current segmentation map overlay. You may also display the current region histograms or interval sets during phases when they exist. All of these displays temporarily erase any rundisplay output.

dqueue {s|t} howmany
Remove *howmany* regions from the head of a queue.

exit Terminate the PHOENIX session. The region map is properly closed before exiting.

history [region#]
Print the history of a region. Describes the region's ancestors, beginning with the earliest.

list {s|t} [howmany] [nth]
List the elements of a queue. Lists *howmany* elements of the specified queue, starting with the nth element from the head of the queue if nth is positive, from the tail if negative. In either case the elements are listed starting with the one nearest the head of the queue.

prune [region#]
Prune a portion of the segmentation tree. Moves the specified region back to the head of the segmentation queue, deleting all of its descendents from the region tree and from the output region map. You are asked whether you really want to do this.

queue {s|t} region# [region# ...]
Add regions to a queue. Adds the indicated regions, one at a time, to the head of the specified queue. They will appear on the queue in reverse order, *i.e.*, the last one listed will be at the head of the queue. No check is made for duplicate regions or for segmented regions being added to the segmentation queue.

If you accidentally invoke this command (*e.g.*, while trying to *quit*), just specify 0 for the region number.

release
Relinquish the display.

restore datafile
Restore a checkpoint file (which must correspond to the current image, but may have different feature planes). The state existing when the checkpoint was taken is restored, except that actions following the last *collect* phase will be forgotten and the display is not restored. You cannot restore a checkpoint if you are running PHOENIX in a different directory from when you created the files.

retry phase
Re-execute a previous segmentation phase. This is only valid when in the middle of segmenting a region. Data structures created since the previous start of the indicated phase are deleted. Table 4.1 shows the transitions that are permitted; current states appear on the left and desired states along the top.

segment
Run the next segmentation phase. The next scheduled phase of segmentation will be executed.

transfer {s|t} howmany
Transfer regions from one queue to the other. Moves *howmany* elements from the head of the indicated queue to the head of the other queue, one at a time. Hence the elements will end up in reversed order. No check is made for duplicate regions.

Table 4.1. Legal Retry Transitions

	ftch	hist	intv	good	next	thrs	ptch	eval	slct	clct
fetch	-	-	-	-	-	-	-	-	-	-
histogram	-	Y	-	-	-	-	-	-	-	-
interval	-	Y	Y	-	-	-	-	-	-	-
goodfeatures	-	Y	Y	Y	-	-	-	-	-	-
nextfeature	-	Y	Y	Y	-	-	-	-	-	-
threshold	-	Y	Y	Y	-	Y	-	-	-	-
patch	-	Y	Y	Y	-	Y	Y	-	-	-
evaluate	-	Y	Y	Y	-	Y	Y	Y	-	-
selection	-	Y	Y	Y	-	-	-	-	Y	-
collect	-	-	-	-	-	-	-	-	-	-

5.1.3. Execution Phases

Interaction with PHOENIX is normally through the control of the *execution phases*. These are "packets" of executable code that together constitute the *segment* command. The normal sequence of segmentation phases is illustrated in Figure 5.1. The user may interrupt the segmentation at any time, either at scheduled interrupts (see flags A, B, and C) or by using the 'Control-C' or 'delete' keys. Execution resumes when another *segment* command is issued.

The phases, in order of normal execution, are listed below. (The descriptions assume that rundisplay has been set to *yes*; otherwise displays must be requested using the *display* command.) To begin or continue this phase sequence, issue the **segment** command. The phases that are then run depend on the control flags which have been set.

fetch
: The next region is fetched from the segmentation queue. (Initially the entire image is one region.) If the 'd' flag is set, a description of the region is printed. The region is expanded by pixel replication and is displayed above the original image (where the region center is marked by the cursor). If the region has an area less than splitmin pixels, or if the 'm' flag is set and the user declines the region, it is declared terminal and a *collect* phase is scheduled. Otherwise the region is passed to the *histogram* phase. You will not be allowed to resegment a region that has already been segmented.

histogram
: A region histogram for each color or feature is computed. Each histogram is smoothed using an unweighted moving average if the hsmooth variable is set greater than 1.

interval
: Each feature histogram is broken into intervals (as described below) and is displayed with the thresholds marked in red. An interval-set quality measure is computed for each feature and boxes are drawn around histograms with acceptable scores. If none was acceptable, the region is declared terminal and a *collect* phase is scheduled; otherwise (if the 'd' flag is set) the interval-set score for each feature is printed.

goodfeatures
: This initializes the spatial evaluation loop [phases *nextfeature* to *evaluate*; see Figure 5.1]. If there are no candidate features, the region is declared terminal and a *collect* phase is scheduled.

nextfeature
: The next good feature is chosen. If all have been evaluated, a *selection* phase is scheduled.

threshold
: The region is thresholded (or level-sliced) using the chosen interval set, and is displayed with a different intensity for each interval. Corresponding intensities are indicated on the feature histogram. The connected components are outlined on the expanded original and

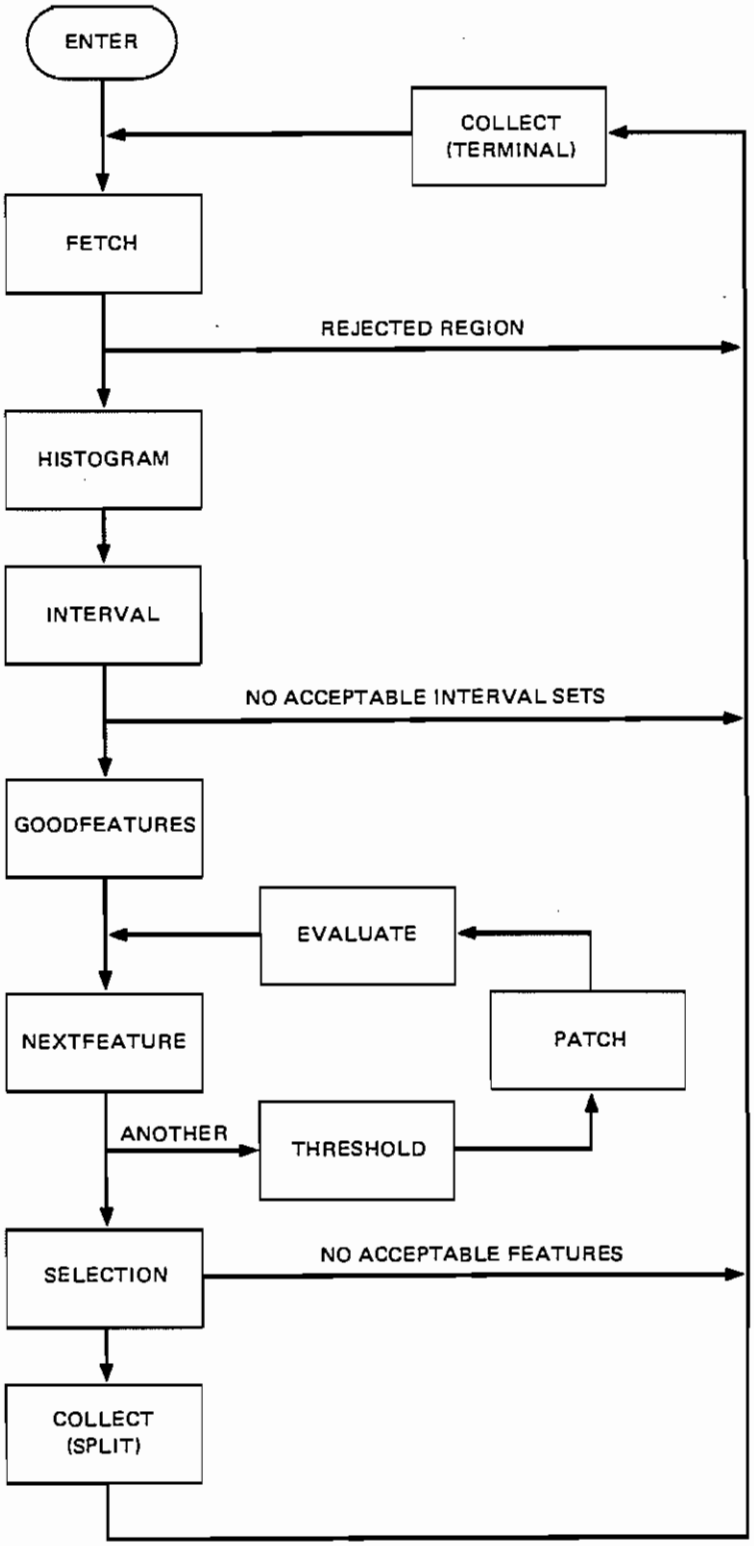

FIGURE 5.1 SEGMENTATION PHASE SEQUENCE

thresholded region images. (This outlining is separate from the boundary extraction done in the following *patch* and *collect* phases.)

patch
: Connected components for each intensity interval are extracted. These components are stored as *patches*, which are run-coded representations together with a few shape descriptors (linear dimensions, area, centroid, number of holes, *etc.*). Patches in the foreground (selected intensity) are 4-connected, while the corresponding background is considered 8-connected.

evaluate
: Patches are classified as either valid regions or noise regions. At present this is determined by comparing the patch area to the **noise** threshold, without regard to patch shape. Each noise region is marked with a dot in both the expanded original image and the thresholded image. The feature is then evaluated by computing the percentage of noise area over the whole image. A *nextfeature* phase is always scheduled to follow this one.

selection
: When all features have been evaluated, the one with the least noise area is selected for segmenting the region. A feature is disqualified if the noise area exceeds the **retain** threshold, or if any one of its intervals failed to produce a valid patch. If no suitable feature is found, the original region is declared terminal; in either case a *collect* phase is scheduled next.

collect
: If the original region has been declared terminal, it is moved to the head of the terminal queue. Otherwise the valid patches (merged with their contained noise patches) are converted to regions. This involves computing the polygon boundaries of the new regions, updating the history list, adding the regions to the segmentation queue, inserting them in the stored region map, and drawing them on the original image display. (These outlines accumulate so that the overlay on the original image always represents the current state of the segmentation. If the user edits the segmentation history or asks for other displays, the outlines may not correspond to the full segmentation.)

This order of execution may be altered in several ways. If an error occurs, *e.g.*, a memory allocation failure, the same phase will be rescheduled as the next phase. The user may also interrupt processing and attempt to schedule a previous phase with the retry command. The system permits some retries and forbids others, depending on the last completed phase and the next scheduled phase. It will object if either a *fetch* phase or the phase you specify is already scheduled next, or if you try to jump forward in the phase sequence. It will also object if you try to jump into the middle of a loop; in particular, you may not retry a *nextfeature* phase.

The segmentation stops when there are no more regions on the segmentation queue. The user may then (or at any time before) ask for various displays and information, edit the segmentation, or save the current region map and region description file. A saved state may be reloaded later and processing may continue.

5.1.4. Status Variables

PHOENIX maintains a set of read-only variables or status query commands. To query the value just type the name of the variable. Although the values may not be set directly, some of them may be changed by other PHOENIX commands.

features
: Features currently being used in segmentation. This is just the list of names following the *-f* flag on the initial command line.

images
: Picture files being used in the segmentation.

lastphase
: Last segmentation phase completed.

nextphase
: Next segmentation phase to be run.

phases
: A list of all the segmentation phases. The last and next phases are designated.

regions
: The number and range of existing regions.

time
: Real time and CPU time spent in each phase and in the entire PHOENIX run. (Real time for a restored segmentation is not meaningful.)

See also the execution flags and control variables documented below.

5.1.5. Execution Flags

Flags (on/off variables) may be used to control execution of the entire program or of any phase. Local phase flags take precedence over global flag settings.

To find out what flags are set, type *flags*. You may turn off all flags by typing *flags* = -*. To selectively turn flags on and off, use a command like *flags* = *-AB+g*, where the plus sign may be omitted if there is no preceding minus sign. The following flags are available:

A (default)
: Begin the next non-*fetch* phase without interrupt. This permits the current *segmentation pass* on the current region to run to completion.

B
: Permit same-depth *fetch* phases without interrupt. Segmentation of the current regions will run to completion, but their children will not be segmented until the next *segment* command is given. Flag B is only meaningful if flag A is set.

C Permit *fetch* phases that initiate new levels of segmentation. Segmentation of the entire image will run to completion. This is only meaningful if flags A and B are set.

D Enable debug printout. (See also flag G.) This option turns on printout of storage management messages.

G Print display subroutine entry, exit, and debugging messages.

H If rundisplay is turned on, step through a detailed display of threshold selection heuristics during the *interval* phase. [This is an SRI addition.]

P Pause rather than stop on interrupts caused by having flags A, B, or C turned off. (To continue after a pause, type a carriage return. To continue after a stop, type *seg*[*ment*].)

d Describe fetched regions and feature quality statistics.

g Order regions on the queues globally by area. This overrides the 'o' flag.

m Request a manual decision on whether to further segment a fetched region. If rundisplay is active, the region will be displayed. The 'd' flag should usually be set so that there is a further basis for the choice.

o (default)
Order regions by area within each depth. If neither this nor the 'g' flag are set, regions are simply added to the tail of the segmentation queue as they are generated.

q Execute quietly, without normal tty output. This does not affect output due to the 'G' or 'd' flags, nor echoing of prompts and commands.

v (default)
Autoverbose mode. Run in verbose (as opposed to quiet) mode for regions with area greater than autoarea. This has precedence over the 'q' flag, but only takes effect locally during a *collect* phase and then permanently during a *fetch* phase.

To set a local phase flag, use a command of the form *during <phase> = -*+AB*. This string will be used to modify the flags variable during the specified phase.

To examine the current modifier value, type *during <phase>*. There is currently no command to display all of the local flag settings at once.

Resetting (disabling) a "during <phase>" option is a little difficult. There is no command to reset all of the phase modifiers at once. To reset them individually you should specify "during <phase> = +".

5.1.6. Control Variables

The user displays and decision logic used by PHOENIX may be fine tuned by setting various option variables and thresholds. To turn on the *rundisplay* option, for instance, type *rundisplay = yes*. To ask for the current value, just type *rundisplay*. Abbreviations are accepted.

The following affect the *fetch* phase or the PHOENIX session as a whole. Default values are listed in parentheses.

autoarea (0)
: Maximum region area for the autoverbose option (flag v) to select quite mode.

depth (infinite)
: Maximum depth of the segmentation tree. Regions at this depth will not be split further. (This test is currently made at the end of the *collect* phase.)

rundisplay (no)
: Use a real-time multiquadrant presentation of processing results. This cannot be used for images larger than 128x128.

  The original image is displayed in the lower-left quadrant with all region boundaries overlayed in red and the cursor centered in the current region. A window containing the current region is expanded by pixel replication and displayed in the upper-left quadrant. Histograms and interval sets are displayed along the right side of the screen. During spatial analysis, the lower-right quadrant contains the selected histogram and the upper-right quadrant displays the thresholded region window. Patches are outlined in green in both of the expanded windows, and noise regions are marked by blue dots.

splitmin (40)
: Minimum area for a region to be automatically considered for splitting. This is an absolute area, not a percentage of the image area.

A fetched region is first histogrammed, and each feature histogram is smoothed. This is controlled by

hsmooth (9)
: Histogram smoothing window. Smoothing is done with an unweighted moving average; the outermost bin values are assumed replicated beyond the ends of the histogram.

The heart of the PHOENIX system is the *interval* phase, since histogram segmentation is the major step in color image segmentation. The variables that control this process, along with their default values, are listed below in the order of their application.

maxmin (160)
: Lowest acceptable peak-to-valley-height ratio expressed as a percentage.

absarea (10)
: Minimum area for an interval to be retained.

relarea (2)
: Minimum acceptable percentage of total histogram area.

height (20)
: Minimum peak height as a percentage of the second-highest peak. This test is skipped if there are only two intervals.

absmin (10)
: Maximum retained valley height as a multiple of the lowest (or "absolute minimum") valley in the histogram. Intervals separated by higher valleys will be merged. This test is skipped if there are only two intervals.

intsmax (2)
: Maximum number of intervals in each final interval set. The intervals will be reduced to this number by merging (*i.e.*, eliminating histogram cutpoints), starting with the highest valley.

Each interval set containing more than one interval is then assigned a score:

$$1000\ \frac{\textit{peak height} - \textit{higher shoulder}}{\textit{peak height}}$$

Interval sets with low scores are not considered for spatial analysis. Thresholds used in the spatial evaluation, or *goodfeatures* to *evaluate* phases, are:

absscore (700)
: Minimum acceptable interval set score. Less promising interval sets will not be selected for spatial evaluation.

relscore (80)
: Minimum acceptable percentage of the highest set score. Features with lower interval set scores will not be considered.

isetsmax (3)
: Maximum number of interval sets (features) that will be evaluated.

noise (10)
: Minimum area of noise regions. Only regions larger than this are retained.

The following affect the *selection* and *collect* phases:

retain (20)
Maximum acceptable noise area as a percentage of a region's total area. Regions with more noise content will not be retained.

tolerance (0.1)
Tolerance for polygon fitting. This affects only the output description and has nothing to do with the region-splitting algorithm.

5.2. Batch Execution

The PHOENIX program offers two methods of invoking prestored commands. The first is the invocation of CI command files, either interactively or with the *-i* command-line flag. For example, you might give the command

```
> <chair.cmd
```

where the file chair.cmd contains the commands

```
flags = -BC+APdov
rundisplay = yes
```

In this case the PHOENIX program will set the flags for a moderately interactive session with the special *rundisplay* turned on.

The second method is to drive the entire PHOENIX session from an operating system script. A UNIX C-shell script might look like:

```
# PHOENIX segmentation system.
# Supply the image name as an argument.

rm -f $1.map
phoenix /iu/tb/pic/$1/.img -f red green blue
    -o $1.map -i $1.cmd <<!
  flags = ABCPdov
  depth = 4
  rundisplay = no
  segment
!
echo "Finished."
```

This script is designed to run without user interaction or visible displays. It does print some information during processing, but does not wait for you to look at it. (You can temporarily halt the processing if your terminal accepts *hold* or *^S^Q* handshaking commands.)

To save the typed terminal output you should pipe the standard output to a file. The UNIX method for doing this is to add *>session.log* to the *phoenix* command within the script or to the UNIX command line that invokes the script. You may also use the UNIX *script* or *tee* commands to route the typed output to a file and to your terminal.

The actual submission of this shell script is described in the UNIX Programmer's Manual. You should run it in foreground mode if you want to interact with the program. If you run it in background mode, be sure to pipe the output to a log file so that it won't appear on your terminal. You can monitor the log file during execution

(using the *cat* or *trail -f* commands) to make sure everything is running smoothly, although the log file will typically run somewhat behind the actual program execution. You can also halt the process or reconnect it to your terminal if you wish.

Section 6

Evaluation

This section documents the performance of the PHOENIX program in test runs on a variety of imagery. Rules are given for setting various scene-dependent parameters, and performance characteristics are evaluated. The section ends with an application of PHOENIX to the problem of skyline delineation.

6.1. Parameter Settings

PHOENIX is a moderately complex system with numerous execution options and 14 user-settable variables that control the segmentation process itself. We will describe the effects of each option alone and in combination with others.

In addition, we will describe the threshold variable settings for "mild", "moderate", and "strict" screening of potential feature thresholds. These correspond to permissive, moderate, and cautious segmentations. These three categories reduce the 14 variables to a manageable single parameter.

Also listed are the minimum and maximum legal values for the SRI version of PHOENIX; the "disabled" value turns off a heuristic completely, and a "drastic" value makes it so strict that very few histogram cutpoints will get through.

We recommend that PHOENIX command files be used as a mechanism for quickly loading sets of commands. We have used files named *strict.cmd*, *moderate.cmd*, and *mild.cmd* in directory /iu/tb/src/phoenix to store the corresponding 14 threshold settings (with the exception that **intsmax** is always set to 2). Files named *run.cmd*, *tst.cmd*, and *display.cmd* store commonly used flag settings and control variables. Each user should develop such command files for the tasks he commonly performs. (PHOENIX should also permit a directory search path to be specified so that standard files could be used or selectively overridden. The underlying CI driver supports this.)

Flags

The flag mechanism controls the amount of interaction between the user and the system. Some flags tell the scheduler whether to proceed autonomously or to stop and ask for commands; others control verbose printout and debugging messages. Several sets of flags (*e.g.*, 'ABC' or 'go') might be better represented by single variables than by interacting flags, but the flag mechanism is useful for allowing "during <phase>" control.

These control options are reasonably straightforward; which flags you should set depends upon what you want to do. They do not affect the segmentation algorithm, so there is no danger of setting them "incorrectly."

Evaluation

The 'm,' 'g,' and 'o' flags come closest to affecting the segmentation process. The 'm' flag allows you to override the *splitmin* heuristic and decide manually whether each region should be split further. This is a valuable option, although a time-consuming one. There is also a need for a more general facility that would accept arbitrary selection criteria for transferring regions from one queue to another. (Although specific tests can be added to the current C-based driver, it would be much easier to implement such screening in a LISP-based driver language.)

The 'g' and 'o' flags control the order in which newly-created regions are added to the segmentation queue. If flag g is set, the regions are ordered globally by size. If 'o' is set (and 'g' is not) the regions are ordered by size within each segmentation depth. If neither is set, new regions are simply added sequentially to the tail of the queue. (There is no provision for resorting the queue when you switch from one to another of these options. Either this should be implemented or the queues should be unordered with selection done during the *fetch* phase.)

Global ordering by size is useful for interactive sessions. The segmenter begins with the largest region and keeps whittling off small subregions until the large region is homogeneous. Then it picks the largest subregion and does the same. With this method there is enough continuity so that you can keep track of what is happening. It would also be good for cueing applications where it is important to find small "blips" quickly.

Depth ordering by size is more useful for automatic segmentation. It has the property that an interrupted session provides a good partial segmentation into regions of similar prominence. If run to completion, the segmentation is identical to that produced with global ordering.

Global ordering by size is equivalent to a depth-first search through the segmentation tree, whereas the other two options (depth ordering by size and sequential ordering) are breadth-first searches. There is a need for more flexible best-first ordering, where the sorting criterion could be based on region shape, color, position, or other properties.

A final note: we suggest that the command "flags = A" should reset all flags other than A, instead of adding A to the current flag list. The "flags = -*+A" syntax could then be used to reset all the "during <phase>" flags as well as the global flags.

Rundisplay (no)

Rundisplay can take only two values: 'yes' or 'no.' It controls the special interactive display that is useful for exploring the system and for debugging. It is so useful, in fact, that any production version of the system should be extended to include some type of rundisplay even for images larger than 256x256.

Rundisplay allows the logic flow to be followed step by step. This has been extended by SRI (*via* the 'H' flag) to include the action of each heuristic cutpoint screening. It could be extended even further to include separate display of heuristics that are currently combined, such as the **absarea** and **relarea** or **abscore** and **relscore** procedures. On the whole, though, the current facility is excellent.

Evaluation

Autoarea (0)

Autoarea controls the size of region for which verbose printout is used if the 'v' flag is set. Normally this variable will be left at its default value of zero. This has no effect on the segmentation algorithm.

Depth (infinite)

```
Disabled:       INF    [Also disabled by flag `g'.]
Mild:            20
Moderate:        10
Strict:           4
Drastic:          1
```

Depth is active when depth ordering of the segmentation queue is used. It prevents segmentations of regions lower than the specified depth in the segmentation tree. (Larger numbers refer to lower depths.) The depth limit can be used to restrict processing time, although this could be better achieved with the **splitmin** threshold or with an actual threshold on time spent.

It is difficult to see how this parameter can be used effectively. Recursive segmentation depth is not a property of a region, but of the region and its context. A strict depth limit will cause differing segmentations of a region when differing orders of features are used to extract it from its background. We therefore recommend that this variable always be disabled or left at the mild setting.

Splitmin (40)

```
Disabled:         1
Mild:            20
Moderate:        40
Strict:         200
Drastic:        INF
```

Splitmin is the only control, other than using **depth** or direct manipulation of the segmentation queue, for which fetched regions are to be segmented further. Any region smaller than **splitmin** is declared terminal and is moved to the terminal, or 't,' queue. (This is useful for examining all regions. Just set **splitmin** to a very large number, turn on rundisplay, optionally set the 'd' flag, and begin segmentation. Each region will be displayed and described before it is rejected.)

The heuristic thresholds given above seem reasonable, but **splitmin** should really be determined by the size of target or object facets being sought. It should be at least twice the **absarea** and **noise** thresholds.

A second heuristic might be used to limit regions to a specified fraction of the image area, thus permitting consistent segmentation across different imaging resolutions. In fact, a more general screening facility could be implemented (particularly in a LISP-based driver) for selecting regions by shape, color, position, orientation, or other characteristic.

Evaluation

Hsmooth (9)

Disabled:	1
Mild:	5
Moderate:	9
Strict:	25
Drastic:	100

Hsmooth is the width of the averaging window used to smooth each feature histogram. (Any spatial smoothing of the feature planes themselves is outside the province of PHOENIX. Such smoothing combats the breakup of textured regions.) Histogram smoothing eliminates many false cutpoints that are due to texture, digitization effects, or color transformations. It also improves the reliability of several other heuristics, as described below. The amount of smoothing required is often quite large because PHOENIX has difficulty distinguishing even small notches from broad valleys between peaks.

Histogram smoothing is done with an unweighted moving average computed by replicating the outermost bin values to plus and minus infinity. This is simple to implement, but may introduce artificial peaks when used on small regions with scattered histogram values. A center-weighted moving average would have better filter characteristics.

This smoothing turns out to be very important – and different values are required at different times. Strict smoothing can be used on peaks that are well separated. This simplifies the task of later heuristics, although cutpoint placement is not critical in such cases. Strict smoothing would be useful for properly splitting peaks that overlap slightly, but would cause the **maxmin** heuristics to discard the cutpoint altogether; moderate or even mild smoothing must be substituted. Mild smoothing is also required for finding small regions within large ones, but is insufficient for segmenting the noisy histogram of a small region.

The problem is that PHOENIX does not use explicit models of histogram peaks. It considers only very simple statistics of histogram intervals, such as apex and shoulder heights. It has no notion of valley width: all heuristics treat a single-bin notch as being identical to a very wide valley. Histogram smoothing is the only mechanism in PHOENIX for making such a distinction, and it is insufficient for the task.

Although modeling of histogram peaks and valleys is the best solution, some improvement could still be made in the histogram smoothing mechanism. A smoothed histogram should augment the original, not replace it. Each heuristic should be able to apply the smoothing that it requires. Then mild smoothing could be used for selection of the initial cutpoints and strict smoothing could be used for positioning of a final cutpoint.

Maxmin (160)

Disabled:	100
Mild:	130
Moderate:	160
Strict:	300
Drastic:	10000

Maxmin is the minimum acceptable ratio of apex height to higher shoulder. Any

interval failing this test is merged with the neighbor on the side of the higher shoulder. The test is then repeated on the combined interval. The overall effect on a set of cutpoints is to eliminate those that are on the sides or tops of major peaks.

The original version of PHOENIX had difficulty if an apex abutted either end of the histogram. The outer shoulder height was taken to be the apex height, and the interval would fail the **maxmin** test. Further, the merged interval would inherit this shoulder height and would also fail the test. This process continued until all intervals had been rejected. We have fixed this in the SRI version by assigning an outer shoulder height of zero to the outermost intervals; this represents the bin height at plus or minus infinity.

Maxmin is a powerful heuristic. With strict smoothing and all other heuristics disabled, **maxmin** alone is able to produce reasonable segmentations. It is even more powerful when combined with the area heuristics. With mild or moderate smoothing, **maxmin** passes clusters of cutpoints in the noise regions between major peaks. This is fine if the clusters can be thinned by the **absarea** and **relarea** heuristics, but a poor selection may be made if they are left for the **intsmax** heuristic.

The problem here is that PHOENIX has no "quality" score for histogram valleys. It assumes that cutpoint bin height is an adequate measure, whereas width and depth relative to the neighboring peaks are also important. PHOENIX can only incorporate such knowledge by smoothing the histogram, and the amount of smoothing required depends on how separated the peaks are.

Absarea (10)

```
Disabled:      1
Mild:          5
Moderate:     10
Strict:      100
Drastic:     INF
```

Absarea is the minimum histogram area that a usable interval may contain. It should usually be set to the same value as the **noise** threshold. (Perhaps the two thresholds should be combined.)

This threshold is tied to the pixel resolution, and so will cause differing effects in images of differing resolution. The value really depends on the size of objects you are trying to find, and on the number of pieces that such an object might be broken into by texture characteristics.

Relarea (2)

```
Disabled:      0
Mild:          1
Moderate:      2
Strict:       10
Drastic:      50    [30 is very strict.]
```

Relarea is the minimum percentage of the histogram area that a usable interval may contain. This intended to eliminate noise peaks (PHOENIX has no explicit model of histogram noise statistics) and to conserve processing time by skipping doubtful intervals.

Evaluation

This is a questionable heuristic since the effect on a particular interval depends on the total area in peaks that may be quite distant. Small peaks are best skipped if larger ones are available in any feature, but there are times when segmentation must be done on the small peaks or not at all. If CPU time is not a problem, it is best to pass these small intervals on to other heuristics and to spatial analysis.

Absarea and **relarea** should both be reduced slightly to allow for PHOENIX's tendency to clip the tails of major peaks. (This is due to the lack of a statistical or semantic model for histogram peaks. A notch in the tail of a major peak is treated the same as a wide valley, and the area heuristics often merge the clipped tail to the wrong side.) Small thresholds for the area heuristics allow multiple cutpoints to survive for screening by the later heuristics.

Height (20)

Disabled:	**0**
Mild:	**10**
Moderate:	**20**
Strict:	**50**
Drastic:	**100**

Height is the minimum acceptable apex height as a percentage of the second highest apex. (The test is skipped if there are fewer than three intervals.) Cutpoints between the highest histogram peaks are favored over those isolating low or noise peaks.

This is a questionable heuristic, for much the same reasons as **relarea**. It is difficult to choose a reasonable value because peak height is much less important than the separation between peaks. Further, the effect on a particular interval can depend upon distant peaks in the same histogram.

The effect can seem mysterious when the second-highest apex is not readily apparent. With mild smoothing the second-highest apex is often part of the main histogram peak separated by a small notch. The **height** heuristic then tends to eliminate all cutpoints that are not similar notches high on major peaks. A strict **maxmin** threshold can combat this by pushing secondary apexes down the side of the main peak. Strict smoothing can also be used to eliminate the notches, although the amount of smoothing needed varies with the histogram characteristics. The simplest solution is to simply disable this heuristic or use a very low threshold.

Absmin (10)

Disabled:	**1000**
Mild:	**30**
Moderate:	**10**
Strict:	**2**
Drastic:	**1**

Absmin screens cutpoints rather than interval statistics. It is the lowest acceptable multiple of the minimum cutpoint bin height. (The test is skipped if there are less than three intervals.) An interval is rejected if either shoulder is not at least **absmin** times the height of the lowest cutpoint bin in the histogram. Unfortunately the name of the heuristic does not make this clear.

Evaluation

The use of a multiplication factor (or ratio) as a threshold entails some difficulties. Unless strict smoothing is used, the global minimum is often zero. All cutpoints with nonzero bin heights are then rejected, and frequently only the global minimum itself will survive. There was no setting of **absmin** that would disable this behavior. We have therefore modified the ratio test in the SRI version so that the denominator is always at least one.

The heuristic is still unstable near zero, but is tolerable and perhaps even useful for large regions. A mild or moderate threshold tends to pass clusters of cuts in the valleys unless they have been thinned by the preceding area heuristics. A strict threshold performs surprisingly well all by itself; with strict smoothing there will be only one cutpoint in a valley, and with mild smoothing there is often a noise notch deep enough to eliminate the other cutpoints.

For small regions (under 100 pixels) this heuristic is useless. Cutpoints for these histograms are nearly always at zero height, so this heuristic cannot choose between them.

Intsmax (2)

Disabled:	**100**
Mild:	**6**
Moderate:	**3**
Strict:	**2**
Drastic:	**2**

Intsmax is the maximum number of intervals permitted in the final interval set for a feature. If more intervals reach this point, the one with the highest maxmin ratio (apex to higher shoulder) is merged with the neighbor on the higher-shoulder side. This process continues until the desired number is reached.

The effect depends on the number and nature of cutpoints passed by the previous heuristics. It tends to favor cutpoints in valleys because small amounts of noise produce high maxmin ratios. This behavior is reasonable, although for mild smoothing it favors noise notches over the centers of broad valleys. (Actually PHOENIX has no notion of the center of a valley. If given a flat valley, it will put the cutpoint on the leftmost bin. Only noise notches or high smoothing will pull the cutpoint to the center.) **Intsmax** may also pass a cluster of cutpoints in one valley in preference to cutpoints scattered through many valleys. The area heuristics may be used to combat this.

Multiple cutpoints can be investigated either in one segmentation phase of many intervals or in many phases of two intervals each. The former saves considerable computation, but gives poor results for reasons described in the **noise** section. It is best to set **intsmax** to two unless there is need to conserve computational resources.

Evaluation

Absscore (700)

Disabled:	0
Mild:	600
Moderate:	700
Strict:	930
Drastic:	1000

Absscore is the lowest interval set score that will be passed to the *threshold* phase. The score is currently just the maximum over the interval set of all the apex minus higher shoulder to higher shoulder ratios, which is equivalent to the maxmin ratio.

This heuristic partially duplicates the screening performed by the **maxmin** threshold, and should be coordinated with that value. The conversion formulas for the component ratios are

$$interval\ score = 1000 - \frac{100000}{maxmin}$$

$$maxmin = \frac{100000}{1000 - interval\ score}$$

It would be simpler if the maxmin ratio were used throughout.

Unfortunately this simple score is poorly suited to choosing a good interval set: one that will generate a segmentation with very few noise regions. Noise regions are a symptom of the worst threshold for an interval set, whereas this formula uses the best threshold. The minimum over the interval set would thus be more appropriate, although an area-weighted average might be better.

An even better score would consider peak and/or valley shapes. The current score is a very weak model, as can be seen for the case of an interval that contain several small histogram peaks: the ratioed apex and shoulder heights may belong to different peaks. The current score is also useless on small regions (*e.g.*, 100 pixels) since the cutpoints usually have zero height and every interval set has a perfect score of 1000.

Relscore (80)

Disabled:	0	
Mild:	65	
Moderate:	80	
Strict:	95	
Drastic:	100	[Single best score is verified.]

Relscore is the least percentage of the highest interval set score that will be passed to the *threshold* phase. This is intended to eliminate poor features when better ones are available, but is less effective for this than the **isetsmax** heuristic.

The difficulty arises because a very small peak separated by zeros will have a perfect score of 1000. (This becomes more likely with small regions or mild heuristics.) Other features will then be rejected if not within **relscore** of 1000, so that **relscore** is acting much like **absscore.** If the small peak is finally rejected by spatial analysis, the region is declared terminal and the other features are never tried. To prevent this, either use strict area heuristics or a very mild **relscore** of approximately one-tenth **absscore.**

Evaluation

Isetsmax (3)

```
Disabled:      100
Mild:            5
Moderate:        3
Strict:          2
Drastic:         1    [Single best score is verified.]
```

Isetsmax is the maximum number of interval sets (features) to be passed to the *threshold* phase. If more than this have survived screening, the **isetsmax** with the highest scores will be chosen. Rejected features will get a second chance in later PHOENIX passes only if one of the chosen features succeeds in segmenting the region.

Noise (10)

```
Disabled:        0    [Always segment on first feature.]
Mild:            5
Moderate:       10
Strict:         50
Drastic:     10000
```

Noise is the size of the largest area that is to be considered noise. This heuristic is applied after thresholding and connected-component extraction. Patches larger than **noise** pixels will be retained; others will be merged with surrounding regions. Note the similarity of this behavior with that of rejecting a cutpoint with the **absarea** threshold.

This is a very difficult threshold to set because the size of noise regions is dependent on the task, the object, and the image resolution. It might be worthwhile to add a relative noise heuristic that would judge the patch area in relation to the original region area. This capability is partially available through the **relarea** heuristic.

Even better would be a noise score or set of region-rejection heuristics that would consider boundary shape, contrast with surrounding regions, local noise statistics, and task-dependent semantic information.

Retain (20)

```
Disabled:      100
Mild:           40
Moderate:       20
Strict:          4
Drastic:         0
```

Retain is the maximum percentage of the original region area that may consist of merged noise regions. If the total noise area exceeds this, the feature will be rejected. It will also be rejected if any interval produces only noise patches, regardless of the noise percentage. After all interval sets have been tested, the one with the least noise area is selected for final conversion of patches to new regions. If two regions are tied, the first is chosen arbitrarily. (This is the only place where the input order of the features makes a difference.)

This heuristic is not intelligent enough for the burden placed upon it. It should be favoring large, compact areas (or other target shapes) as well as noiseless ones. At

present it is quite happy with a trivial segmentation of a tiny region vs. all the rest. This is fine for cueing applications, but poor for general use.

Use of multiple cutpoints (*i.e.*, **maxints** greater than 2) introduces additional noise and increases the likelihood that some interval will fail to produce a good patch. PHOENIX is unable to recover from this by deleting one cutpoint at a time or by retaining the good patches and discarding the rest.

One solution is to add relative noise heuristics as well as this absolute one. Noise could be expressed as a percentage of patch area: any patch containing too much noise would be rejected. It could also be expressed as a percentage of interval area. Either of these would integrate well with retention of all patches or intervals that are useful, without regard to the success of the feature as a whole.

Even better would be a set of region-acceptance heuristics that would consider boundary shape, contrast with surrounding regions, local noise statistics, and task-dependent semantic information. Such heuristics would be easiest to implement in a LISP-based driver.

6.2. Performance Statistics

To further evaluate PHOENIX, it is necessary to choose a task domain. We have selected skyline delineation. This is the problem of determining the skyline in an image that includes both ground and sky.

It should be noted that this problem is not always well defined. Images of cloud-shrouded mountain peaks or of fog rolling in over a mountain range present difficulties. There is also the case of a distant horizon seen over a nearby crest: the near skyline may be the one of operational importance.

We have chosen a range of images for testing. *Portland* shows a city skyline against a cloudy sky. *Mountain* is a distant mountain against a nearly clear sky. *Bishop* contains a near skyline and a distant one that merges with a cloudy sky; it is difficult for untrained observers to segment. All of these images were reduced to 128x128 to save execution time and to permit use of the rundisplay option.

An early test with the *portland* image at full 512x512 resolution was disappointing. It was done with red, green, and blue input feature planes and with the original default threshold settings. (In particular, **hsmooth** was 1 and **height** was 70. **Maxmin** was also set to 100 in order to get the segmentation started.) The resulting segmentation was erratic, locating many small details while missing several obvious regions. In particular one white building was not distinguished from the blue sky even though most of its windows were found. Numerous tiny patches of sky were segmented out for no apparent reason, yet an easily visible U.S. flag was not distinguished from the sky region.

Subsequent analysis and experimentation led to several improvements: minor software bugs were fixed, the strict/moderate/mild parameter scale was developed, and Kender's versions of the HSD and YIQ transforms were implemented.[1] The D and Y transforms are essentially redundant, and are also very similar to the red, green, and blue feature planes. They do not always segment identically, but the extra

[1]Subsequent correspondence with Steven Shafer indicates that CMU researchers have favored Ohta's transforms over the nonlinear HSD scale.

information is not worth the computational effort. We have used all nine features, however.

Hue was mapped to the range 0 to 179, with red at 0 (and 180), green at 60, and blue at 120. Achromatic pixels (*i.e.*, black, gray, and white) were mapped to 255; this ultimately made no difference since pixels with exactly equal red, green, and blue components are exceedingly rare. A less exact test for achromaticity might work better (or at least differently) for images with slight imbalances in the color strengths; the *bishop* image, for instance, is found to have red clouds even though they appear white.

Pixels containing blue mixed with red (*i.e.*, purples and violets) are also rare even in the hazy mountain scenes, so we found no particular problem with peaks in the hue histogram being split between the bottom and top portions of the scale. Saturation was more likely to have such instabilities; we found examples of dark or shadowed image regions that transformed to very high saturation values. The histograms resembled peaks with their left tails clipped at zero and moved up to the high end of the scale. Such areas were so small in our test imagery that they never caused any difficulties.

The I and Q color features computed by Kender's formulas must be divided by two (and then shifted to a nonnegative range) if they are to be stored in 8-bit image planes. Compression to eight bits is not really required by PHOENIX, but it seems a reasonable dynamic range. Experiments showed, however, that most of this range was being wasted. We chose to stretch I by a factor of two and Q by a factor of four prior to quantization, with clipping of extreme values. This greatly increased their usefulness for natural imagery, although it could fail for scenes containing large regions of saturated colors.

For skyline delineation, hue was the most important feature. Sky, clouds, and some vegetation all had hue values near blue or blue-green, whereas land and buildings were closer to red, orange/brown, and yellow. This might not hold true for other scenes, but, for our *portland* and *mountain* images, the hue feature and the strict parameter settings were nearly sufficient to extract the sky as a single region. For the *bishop* image they extracted the near skyline from the rather homogeneous background of distant land and cloudy sky.

Even better results were obtained by first segmenting with strict heuristics and then resegmenting with moderate heuristics. (This involves somewhat more computation than using the moderate heuristics alone, but did a better job of segmenting textured regions.) The strict heuristics typically produce three to five regions for a 128x128 image, and the moderate heuristics extend this to 12 to 30 regions. Further segmentation with the mild heuristics produces 60 to 100 regions, many of them shadows or contours in fairly smooth scene regions. Some of these contours may be due to instabilities in the color transforms, but most have visible interpretations.

Skyline determination was straightforward in the *portland* and *mountain* images because the sky was extracted as a single region. The *bishop* image was much more difficult. Strict and moderate heuristics separate the nearby land, blue sky, several large cloud areas, and a large region that included a distant valley, a rim of mountains, and a cloudy sky. The true skyline could only be segmented by using the mild heuristics. It was found as a single boundary, but could easily have been broken apart if the various thresholds had been slightly different. In any case the challenging problem of determining which regions were sky and which were land is not resolved by PHOENIX; it just passes the regions on to some unknown post-processor

(in this case a human visual system).

The *bishop* image exhibited another characteristic of PHOENIX. In segmenting the blue sky from large cloud masses, it misplaces the boundary slightly. This is because the histogram cutpoints are sensitive to global area effects rather than local spatial variations. (Shafer [Shafer80, Shafer82] discussed this as the "majority rule" problem.) The misclassified cloud patches are picked up during later segmentations, but are so small that many are remerged with the sky. PHOENIX currently has no way of detecting the spatial patterns of small noise patches that indicate a poorly chosen border or a string of mixed-source pixels.

A final test sequence was run on the full-resolution (500x500) *portland* image. Strict and even moderate heuristics were unable to segment the image when only the red, green, and blue feature planes were used; it was necessary to use the mild heuristics. The best approach would be to start the segmentation with mild thresholds and then return to strict or moderate ones for segmenting the subregions. Instead, we avoided such special interference and ran the segmentation to completion using mild heuristics. The full run (which, with the 'v' flag set, generated 19,000 lines of printout) required 33 minutes of CPU time:

PHASE	REAL	CPU
Fetch	0:00:13	0:00:08
Histogram	0:04:13	0:02:32
Interval	0:18:12	0:07:27
Goodfeatures	0:00:01	0:00:00
Nextfeature	0:00:01	0:00:01
Threshold	0:10:00	0:03:47
Patch	0:03:51	0:03:30
Evaluate	0:00:05	0:00:04
Selection	0:00:08	0:00:05
Collect	0:38:12	0:14:04
Segmentation	1:18:15	0:32:34

The final segmentation into 1182 regions (including nearly every window of every building) was much better than the original attempt, but still had difficulties distinguishing a glass-surfaced building from the sky that it reflected. The U.S. flag was segmented out as two small regions.

Another attempt was made using color transforms. This time the strict heuristics were able to segment sky from land using the hue feature. Results were very similar to those for the reduced *portland* image, although outlines were noisy and somewhat more "gerrymandered." Splitting on the hue features required less than two minutes of CPU (with perhaps an equal amount for computing the color transforms) and produced nine regions, one of which was the U.S. flag. Some vegetation and building surfaces were included in the sky region, including most of the glass-surfaced building. It took another minute to determine that the nine regions were homogeneous.

Further segmentation required switching to the moderate heuristics. Running this sequence to completion produced 153 regions after an additional 11 minutes. The sky was cleanly separated from the vegetation and buildings, but had been split into two major regions along a front in the cloud cover. The **noise** threshold of 10 was evidently too low for this task and image resolution, but only a few small regions were

retained. This combined strict/moderate segmentation of the color transforms was very successful at skyline delineation. (Segmentation using the moderate heuristics alone is also quite good.)

The regions found by PHOENIX are not smoothed in any way. Often they are narrow, twisted, or convoluted. This contrasts with human segmentation, which favors straight lines at the expense of region homogeneity. Despite this, the regions found with the strict and moderate thresholds are quite reasonable, and even the mild thresholds give acceptable segmentations. The best course seems to be to oversegment the image and then use some type of post-analysis to classify and merge the regions.

For the particular application of skyline delineation, PHOENIX is handicapped by its lack of knowledge about the task domain. It spends much of its time segmenting and resegmenting areas that are nowhere near the skyline. A more focused search would save computation and pass fewer regions for further analysis. Specific feature planes for land/sky segmentation might also be used to simplify the segmentation and classification task.

6.3. Summary

PHOENIX is a general-purpose segmentation system. It is designed to produce a reasonable segmentation on almost any type of imagery. Proper use of the system requires extensive knowledge of the algorithm and of the effects of various threshold settings, but the system can be made to produce reasonable segmentations.

A difficult part of the Testbed integration effort was the analysis and documentation of PHOENIX control options and heuristic thresholds. Eventually this work led to the strict/moderate/mild threshold settings specified above. The various settings were determined by analysis, by disabling most heuristics and testing the remainder in isolation, by watching the heuristics interact during segmentation of a simple *chair* image, and by refinement during segmentation of natural imagery. While possibly not optimal for any particular purpose, these threshold groupings provide a framework for fine adjustments.

Evaluation of segmentation software is a difficult task. There are few methods for comparing segmentations other than tabulation of pixel classification errors [Yasnoff77] or subjective evaluation on simulated or natural imagery [Nagin79, Ranade80]. We have subjectively evaluated PHOENIX's performance for a particular task using a variety of images.

PHOENIX performed adequately for the task of skyline delineation. We did not develop optimum parameters or procedures for this task, but used very general techniques developed for much simpler test imagery. The amount of computation that PHOENIX required to find the skyline varied with the difficulty of the scene, but it did succeed in all cases. The further problem of determining which regions constitute sky is beyond the domain of this system.

Section 7

Suggested Improvements

The process of evaluation has turned up numerous ways to improve the current PHOENIX implementation. Comments about existing features have been made at the appropriate points throughout this document. The following are additional suggestions for substantial modifications or needed research. Some of these would require major research projects or are beyond the scope of a segmentation program *per se*. (The large number of suggestions should not be taken as a criticism of the PHOENIX system. Rather it is a tribute that the approach is flexible enough to support such extensions and is promising enough to be worth the effort.)

- *Flexible Interaction*

  PHOENIX is both an automated segmentation system and an interactive one. The interactive control system is excellent, but could be improved if more of the dynamic decisions were based on queues or lists instead of compiled iterations. The user could then attach and detach feature planes, manually screen or add histogram cutpoints, select heuristics to be applied, accept or reject thresholded patches, *etc*.

- *Alternate Color Features*

  Our experience indicates that the hue feature is much more useful for skyline delineation than the original color features. Researchers at CMU have favored Ohta's transforms over HSD features for general work. LANDSAT analysts have used various ratios of color bands to emphasize water, vegetation, mineral deposits, *etc*. There may still be much to be gained by developing transforms suited to particular tasks.

- *Texture Transforms*

  Texture features supplied to PHOENIX evidently need to be combined in much the same way that color features are combined into YIQ and HSD versions. Combinations of texture and color features might also be useful for splitting the one-dimensional projections of multidimensional histogram peaks.

- *Additional Feature Types*

  PHOENIX has been developed primarily for color image segmentation, although it seems able to work in other multispectral and multitextural domains. Since additional features can only improve performance (at a cost in processing time), it may be desirable to add other computed scene

characteristics such as gradient and edge maps; stereo disparity; estimated illumination at each pixel; estimated surface distance, reflectance, curvature, and orientation [Horn77, Barrow81, Brady82]; optic flow [Thompson80]; and estimated material type.

* *Delayed Transforms*

PHOENIX currently accepts color transform features (YIQ, HSD, *etc.*) as input feature planes. This works well in a research environment, but might require more storage and computation than necessary for a production environment. These feature planes and histograms can be computed from the image as needed. Perhaps few regions would require thresholding on transformed values if RGB segmentation were first used wherever effective.

* *Histogram Stretching*

Some of the color transform features are likely to have a narrow range on any given image, making them useless for segmentation. Unfortunately the feature ranges vary from one image to another. Since PHOENIX is not sensitive to linear transformation of the features, it might be wise to stretch each feature to its full dynamic range prior to quantization. (This requires an initial pass through the image to determine the range.) This computation could even be done on a region by region basis. Note, however, that full nonlinear histogram equalization will prevent PHOENIX from segmenting the feature at all.

* *Adaptive Smoothing*

PHOENIX currently applies the same smoothing window to each of the feature histograms. An adaptive or iterative smoothing algorithm that suppressed noise without merging peaks would perform better.

* *Luminance Screening*

Ohlander and Price [Ohlander78] segment first on high and low luminance (Y or D) values to avoid singularities in the color transforms. PHOENIX counts on spatial analysis to reject these unstable transform intervals, but might benefit from similarly extracting bright and dark regions before doing more general segmentation[1]. An alternative is to change the color transform code so that colors in the unstable regions are all mapped to special code values; PHOENIX might then need to understand these mappings.

* *Histogram Modeling*

Several comments were made in Section 6 about the deficiencies of PHOENIX's interval selection algorithm. The most severe problems relate to

[1]This capability is now available in the CMU version of PHOENIX.

its lack of a model for histogram peaks or valleys. Although its heuristics are cheap and often effective, there may be better alternatives. Statistical modeling has already been mentioned. Spline fitting, Kalman filtering, filtered gradient zero-crossing detection, and hierarchical waveform parsing [Ehrich76] are others. Another idea is to use one set of heuristics to assign a "valley center" score to each histogram bin and another set to select high-scoring bins that are spaced suitably far apart.

* *Circular Features*

Hue is computed on a circular interval, with red at both ends of the scale. The histogram analysis routines could be modified to understand this characteristic so that purple/red peaks would not be eliminated or split. The PLAN segmenter [Price76, Ohlander78] has this capability.

* *Feature Rejection*

A feature may fail to segment a region either because it contains broad peaks that cannot be resolved or because the histogram has degenerated to a narrow spike. Although the latter is not too common, some computation could be saved by eliminating such a feature from all further splitting of the region and its subregions.

* *Reordered Heuristics*

Questionable heuristics such as **relarea**, **height**, and **absmin** should be postponed as long as possible in order to develop context and perhaps eliminate the need for the decision. A supervisory system might be added to determine when these tests are required, and multiple spatial analyses might be performed as a final check.

* *Alternate Heuristics*

The absolute heuristics (*e.g.*, **absarea** and **noise**) can only be set in the context of a particular task and image resolution. Relative heuristics are generally better, although the PHOENIX versions often perform differently for small regions than for large ones. Also good are those, like **intsmax**, that rank order the histogram cutpoints and choose the top few.

PHOENIX and the Ohlander/Price segmenters use slightly different heuristics for segmenting histograms. In particular, Price's version prefers bimodal features and also considers the heights and slopes of neighboring peaks (to avoid chopping off the tail of a skewed peak). There is also a special heuristic for extracting a low-saturation interval. Such heuristics could easily be added to PHOENIX, although it is not clear how they would interacted with the existing heuristics.

Once histogram peaks have been found, Ohlander and Price use successively weaker acceptance criteria to choose a single histogram peak for thresholding. This differs from the PHOENIX approach, which uses successively

stronger rejection criteria to screen potential cutpoints. While either set of heuristics might be transformed to the other system, it is not clear how acceptance and rejection criteria could be made to work together.

A useful property of PHOENIX's heuristics is monotonicity; once used, later applications of the same heuristic would have no effect. If other heuristics were introduced that destroyed this property, it might be necessary to repeat the heuristics round-robin until the entire set produced no change in the cutpoints.

Perhaps the best advice is to make the heuristics so intelligent that each individually seldom makes a mistake, and to make them accessible to the user so that they can be refined when exceptions are found. This is the expert systems approach, with part of each heuristic being a test to determine when the rule is applicable.

* *Multivariate Histogram Analysis*

Clustering and multivariate histogram segmentation are discussed in Appendix A.6. There may be situations in which a single three-dimensional histogram analysis is more powerful and less expensive than PHOENIX's sequential univariate analyses of (typically) nine histograms. Histogram storage and analysis are becoming much less of a problem as computer hardware improves, and a single clear-cut decision in multidimensional space may often take the place of many doubtful decisions in the one-dimensional spaces.

* *Adaptive Cluster Analysis*

Most clusters in a multidimensional histogram space can be adequately separated by piecewise-linear decision boundaries. These decision surfaces can be found by standard cluster analysis techniques without storing multidimensional histograms. The advantages increase as the number of features considered increases, since the adaptive cluster methods require essentially the same analysis time regardless of dimensionality. There are additional advantages to using parametric (*e.g.*, Gaussian) methods where appropriate, since they are designed to optimally separate peaks from each other and from random noise.

* *Conservative Thresholding*

The region boundaries computed with PHOENIX are affected by global circumstances such as the number and size of other similar regions. An editing phase may correct the boundaries by moving them back and forth and by deleting noise regions, but will not be able to recover small regions that have been absorbed by their neighbors. The occurrence of such lost regions can be minimized by conservative thresholding [Nagin77]. Some type of region growing is then needed to merge pixels between regions. One method of adjusting region boundaries is given in [Barrett81].

Suggested Improvements

- *Noise Analysis*

  PHOENIX currently discards any region that is too small either in absolute area or as a percentage of its parent region. Even for task-independent segmentation this may be too simple; any meaningful interpretation of some small patches would sharpen the **retain** test based on the remaining noise. There may also be applications for which the small anomalous patches are important and cannot be discarded.

  In the most common situation, poorly segmented or mixed-source pixels are discovered along a region boundary. PHOENIX remerges these with the parent region instead of testing to see which region should properly contain them. (This could be done by disabling the noise heuristics and allowing a post-processor to make such decisions, but, with the noise heuristic disabled, PHOENIX has no way to choose which feature to use.)

  A more difficult case arises for occluded objects or "flocks" of related pixels. The disconnected parts have similar histograms and are located by the histogram analysis, but spatial analysis rejects the feature or merges the patches. This is right for most applications, but wrong for others. A more sophisticated system would analyze the small patches for shape, contrast, regular spacing, similarity to existing regions, multispectral signature, or other unifying criteria.

- *Planning*

  PHOENIX does not currently include the planning mechanisms developed by Price [Price76]. These would seem worth inclusion in either a research or a production system. The software involved is similar to that for conservative thresholding.

- *Partitioning*

  Another neglected feature of Price's system is partitioning of large regions. Price uses thresholds derived from the subregions to segment the entire scene – this gets the segmenter started when faced with unimodal histograms. An alternative is to analyze each subimage independently, then merge the region descriptions in a later editing step. The method performs badly if the arbitrary divisions are close to true region boundaries. While this can lead to some blockiness, it reduces computation time at a very small sacrifice in global information.

- *Selective Sampling*

  The problem of finding small regions within large ones may also be combatted by computing histograms only near pixels with high gradient [Weszka74]. Equally valid is the use of only low-gradient pixels; this resolves the centers of large regions but may produce poor boundaries. Such techniques could be used after recursive segmentation of a region can proceed no further. They are made easier if a gradient map is one of the input features.

Suggested Improvements

* *Relaxation Analysis*

  Often a histogram is obviously bimodal, but the peaks cannot be resolved. PHOENIX allows the feature to be used for splitting, but may not be sophisticated enough to merge the resulting noise regions into a meaningful segmentation. For such cases, or even for unimodal regions, more expensive analysis may be appropriate. Use of more features, partitioning, and selective sampling have already been discussed. If all else fails, one can modify the original image by nonlinear relaxation to smooth the subregion interiors and enhance the boundaries [Bhanu82].

* *Map Input*

  One method of adding planning and feedback is to feed crude segmentation maps to PHOENIX as feature planes. These maps might come from previous PHOENIX runs or from other segmenters. Using such maps requires different control structures and heuristics since the bin contents, not the overall histogram peaks and valleys, are the meaningful features. Phoenix can make partial use of such a segmentation map only by accepting it as the current state and then trying to split it further. A more flexible system might use multiple segmentation maps as guides to a multidimensional cluster analysis.

* *Adaptive Thresholds*

  The Ohlander and Price segmenters use a tightly constrained valley selection heuristic, then a weaker one. A similar interactive technique has been found useful with PHOENIX. This concept could be integrated with the PHOENIX control structure by automatically segmenting first with severe histogram smoothing and tight constraints, then with gradually relaxed constraints for regions that are deemed worthy of further effort. Each new region would go through this same sequence of tests. The cost of such a technique would be lessened if the pre-smoothed histogram were retained until a satisfactory segmentation was achieved.

* *Shape Analysis*

  PHOENIX currently chooses a region for segmentation without regard to the region's shape or context. Only the region size and segmentation depth are considered. It is possible that better segmentation could be achieved by considering shape during the *fetch* phase and also during spatial analysis. Extended regions such as rivers and roads may require heuristics different from those for compact regions.

* *Heuristic Training*

  The space of all heuristic orderings and threshold settings is too large for intuitive design. If the heuristics are to be extended or improved, some type of ordered search is required. This will require a set of training images with known region boundaries. PHOENIX can be modified so that segmentation

errors are flagged and evaluated at each step. A human operator or higher-level control system could then drive PHOENIX through the training set, adjusting the thresholds to achieve good performance. If ground-truth training images are not available, a much more sophisticated expert system will be required.

* *Additional Displays*

The rundisplay option is very good, and made it much easier to evaluate the existing heuristics. The SRI heuristic display (flag H) should be extended to show separately the action of the **absarea** and **relarea** heuristics, and of **absscore** and **relscore.** (It should also be modified to allow early escape from the full set of displays. The best solution would be to make each heuristic application a separate phase.)

The rundisplay layout of all feature histograms on a single screen is also excellent, although it could be improved by printing the interval set score with each histogram. A similar display should be implemented for the "display histograms" command, which currently shows the histograms one by one. For single-feature interval set display, each interval set area should be printed; a vertical scale on the histogram might also help.

For large images, where rundisplay is currently not available, it would be useful to be able to display the histograms of any region at any time. At present this usually involves moving the region to the segmentation queue and executing a *histogram* phase. This cannot be done if the region has already been segmented unless you are willing to prune the region.

Better displays are also needed for showing individual regions in context. This is currently done by drawing the region outline on the original image and marking the center with a blinking cursor. Unfortunately the outline is often difficult to see and the cursor is insufficient to indicate whether the inside or outside of the outline is meant. A better display would show either the region or its surround as a solid patch. (A keystroke could be used to flip between the two options.)

During rundisplay each region that is created is drawn as an outline on the original image. This overlay is supposed to represent the current state of the segmentation. It should be erased and redrawn when a region is pruned. Some of the other rundisplay components should be erased when a retry command makes them obsolete.

* *Immediate Feedback*

The noise area produced in a *threshold* phase is not reported until after all promising features have been analyzed. It would be better to report the results of the spatial analyses individually as well as jointly; the user could then match the noise statistic with the corresponding patch display. (The *evaluate* phase does little except compute and print these percentages. It could be eliminated.) Another improvement would be to inform the user about which heuristic rejected a particular interval set score.

Suggested Improvements

- *Verbosity Coordination*

  The PHOENIX code contains several mechanisms for controlling verbose printout and debugging messages. Various messages are controlled by compiler flags, global variables, PHOENIX flags, and by the SRI **printerr** package. It would be better if all were controlled by PHOENIX flags or variables. There should be an additional flag to print the name of each phase as it is begun; this would simplify debugging and retry commands.

- *Queue Management*

  PHOENIX maintains a segmentation queue and a terminal region queue. It is somewhat disconcerting when the same region appears on both, or when a region appears several times on one queue. PHOENIX does check each fetched region to make sure that it has not been segmented, but a better approach would be to ensure that the queues remain valid at all times. Adding a region to a queue should remove all other occurrences, and segmented regions should not be allowed on the segmentation queue. The queue manipulation routines should also be augmented with various screening options for transferring regions from one queue to the other.

- *Split/Merge Capability*

  One option that the user should have is to combine two neighboring regions. Eventually heuristics might be added for doing this automatically in appropriate circumstances. The segmentation history will require a general graph representation instead of a tree.

- *Explanatory Capability*

  It would also be helpful if enough history information were kept so that the system could answer questions about the final segmentation and the steps that led to it. This would include questions about why a particular region had been retained and why it had not been split further, what thresholds would be needed to segment it further, what effect those thresholds would have on other regions, *etc.* (Admittedly some of the answers might require extensive computation.) Such question-answering capabilities are common in expert systems. The answer to a "why did you" question is typically a printout of the rule that triggered the action.

- *Coroutine Implementation*

  PHOENIX can be driven by another program, but the interaction is clumsy. The driver program must invoke PHOENIX and send commands down a UNIX pipe. Output is obtained by sending a "checkpoint" command and then examining the resulting map and data file.[2]

[2]This solution was suggested by Steven Shafer at CMU. It avoids the checkpoint parsing overhead of repeatedly invoking new PHOENIX processes with the single-step option.

Suggested Improvements

For more flexible interaction, PHOENIX must either be implemented as a subroutine or as a server process. The subroutine approach gives the control program a dedicated process for segmenting a particular image; some communication protocol would be needed for conveying the new segmentation results. The server, or coroutine, implementation is more like having a separate piece of hardware for segmenting images: the control program would send requests, and PHOENIX would send back replies. This permits isolation of the PHOENIX history files so that no other program would have to load and parse them, but it does introduce complications if PHOENIX services are to be shared by several control programs.

Section 8

Conclusions

The PHOENIX segmentation system is one of several existing systems for recursively segmenting digital images. Its major contributions are the optional use of multiple thresholds, spatial analysis for choosing between good features, and a sophisticated control interface. Some of the strengths and weaknesses of the PHOENIX algorithm are listed below.

- PHOENIX, like other region-based methods, always yields closed region boundaries. This is not true of edge-based feature extraction methods, with the possible exception of boundary following and zero-crossing detection [see Appendix A]. Closed boundaries are the essence of segmentation and greatly simplify certain classification and mensuration tasks.

- PHOENIX is a hierarchical or recursive segmenter, which means that even a partial segmentation may be useful. This can save a great deal of computation if efforts are concentrated on those regions where further segmentation is critical. If PHOENIX is to be driven to its limits, other methods of segmenting to small, homogeneous regions may be more economical.

- PHOENIX is relatively insensitive to noise. Thresholds are determined by the feature histograms, where noise tends to average out. This contrasts with edge-based methods, where the local image characteristics can be highly perturbed by noise.

- Different segmentation problems require different amounts of histogram smoothing [Ranade80]. It generally works best to start PHOENIX with strong smoothing and strict heuristics and then to gradually weaken both. Some images, however, require mild smoothing or thresholds to get the segmentation started. An adaptive system would be desirable.

- PHOENIX has no notion of boundary straightness or smoothness. This may be good or bad depending on the scene characteristics and the analysis task. It easily extracts large homogeneous regions that may be adjacent to detailed, irregular regions (*e.g.*, lakes adjacent to dock areas or sky above a city); such tasks can be difficult for edge-based segmenters.

- PHOENIX tends to miss small regions within large ones because they contribute so little to the composite histogram. It is thus poorly suited for detecting vehicles and small buildings in aerial scenes, although there may be ways to adapt it to this use. It also tends to misplace the boundary between a large region and a small one, thus obscuring roads, rivers, and other thin regions. Boundaries found by edge-based methods are less affected by distant scene properties.

Conclusions

- PHOENIX may also fail to detect even long and highly-visible boundaries between two similar regions if the region textures cause their histograms to overlap. Edge-based methods are better able to detect local variations at the boundary.

- PHOENIX requires multispectral or "multitextural" input for effective operation, and may even require transformations and combinations of these feature planes. Edge-based techniques are better adapted to operation in a single feature plane.

- Since perfect segmentation is undefined and unobtainable, PHOENIX must oversegment an image in order to find all region boundaries that may be of use to any higher-level process. It is left for a segmentation editing step to merge segments that have no usefulness for some particular purpose. Without having such a step, or indeed even a purpose, it is very difficult to evaluate the segmenter output.

Selection of a segmentation algorithm and improvement of a particular software package are both highly dependent on the task to be performed. The PHOENIX segmentation system is a flexible starting point for further development. This report and the SRI Testbed environment help to make PHOENIX available as a benchmark system and as a research tool.

Appendix A

Alternate Segmentation Techniques

This appendix explores alternate methods of segmenting images. It is intended to clarify the issues involved in region extraction, and to introduce background and vocabulary needed to read the literature in this field. For other surveys see [Zucker76a], [Riseman77], and [Fu81].

A.1. Edge Methods

One approach to segmentation consists of detecting small edge elements and then linking them into region boundaries. Edge and region methods are nearly equivalent for simple scenes of cubes and wedges. In natural images, specific structures are best found with particular techniques [Nevatia77a]. Region methods locate irregular, homogeneous regions, but may ignore or conceal linear features; edge methods detect linear features and detailed (or possibly camouflaged) objects, but give fragmented region boundaries that may be difficult to interpret. Perhaps the two must be combined so that detected edges provide context for region growing and region knowledge can aid edge linking [Milgram77, Milgram78, Barrow81].

Sometimes edge detection and linking are combined [Pingle71, Montanari71, Martelli76]; this is called edge following or boundary tracking, and has advantages when closed regions are required. A similar method is run tracking [Nahi77, Nahi78], in which the object boundaries found on one row are used to aid location of boundaries on the next row. (This is similar to the PHOENIX connected-component extraction algorithm.)

A separate edge detection step is more popular because it is compatible with either single-pass or parallel implementation, and because the detected edge elements are also useful as texture primitives. Edge linking may be done using relaxation labeling [Riseman77, Zucker77, Prager80], expansion-contraction to close gaps [Perkins80], curve fitting, or clustering and heuristic linking [Jarvis75, Nevatia76, Fischler83].

Edges in digital images are difficult to define. A few edge detectors are based on theoretical models of scene edges [Hueckel71, Hueckel73, Horn77, Mitiche80, Haralick81, Brady82], but most are heuristic local gradient estimators [Davis75, Pratt78]. Some operators are small in order to approach a true local derivative, others are quite large to provide noise immunity. Comparative studies [Fram75, Bullock76, Abdou79] have not proven the superiority of any one operator for all classes of imagery.

Color edges are even more difficult to define. Either a single gradient map must be defined on the multivariate feature plane, or edges detected separately in each feature plane must somehow be combined [Nevatia77b, Robinson77]. For color data the method should match human perception of color edges, but we would like it to extend to texture features and other data as well.

Texture edges (*i.e.*, boundaries between regions of differing texture) are also important. The standard approach is to identify ordinary intensity edges in some texture transform of the image, but texture-specific methods have been developed [Thompson77, Deguchi78, Davis80, Davis82].

Some exciting advances have been made in the area of zero-crossing detection [Grimson80, Brady82]. The image is convolved with the second derivative of a Gaussian blur function (chosen to match hypothesized channels in the human visual system). Zero-crossings in the filtered image then form closed region boundaries whose positions can be estimated with sub-pixel accuracy [MacVicar-Whelan81]. Further, the sensitivity of the detector to edges of different widths can be controlled by the width of the Gaussian function, and the strength of the edge at a given point can be measured by the rate of change across the zero crossing. More work is needed to determine how to combine these multiple sources of evidence without losing the closed-region property.

A.2. Thresholding

Thresholding is a quick way of locating regions. Often an image function may be found that is maximal for the smooth interiors of regions and minimal for region boundaries. Other functions, such as the image itself, may be maximal in some region centers and minimal in others; boundary areas take on intermediate values. In either case, thresholding may be used to separate region interiors from edges.

Using successively lower thresholds generates a contour map; adding a stopping criterion makes this a segmentation algorithm. In forward-looking infrared (FLIR) target imagery it has been found that object shapes change very little as the threshold is varied, but noise regions change dramatically. Milgram [Milgram77] exploits this consistency by choosing the threshold giving the best match between corresponding region boundaries and the edge elements detected by another method; this has difficulties with small or textured regions [Ranade80].

There are three types of threshold: *constant*, *scene-dependent*, and *adaptive*. ([Weszka78] further classified thresholds as *global* if they depend only on pixel value, *local* if they depend on neighboring pixel values, and *dynamic* if they depend on spatial position.)

Constant thresholds are those having the same value for all images (*e.g.*, [Kasvand74]). Some real-time hardware systems use this technique, but it is rare for any function of diverse images to have an appropriate constant threshold.

Scene-dependent thresholds are constant for a given image, but may vary as a function of the sensor, illumination, analysis task, or image content. The threshold is typically set interactively by an observer or automatically by histogram analysis. Histogram thresholding was developed in the context of cell segmentation and identification [Prewitt66], and may still be the best technique for this purpose [Ranade80].

Relaxation processes have also been used to remap the histogram into a few dominant intensities [Rosenfeld78, Peleg78, Ranade80]; this is essentially a thresholding process. Like other histogram-based methods, it tends to ignore small regions that may be semantically meaningful.

Adaptive thresholds are set automatically as a function of local scene content; they

vary from point to point within an image. Such thresholds can adjust for changes in illumination within a scene. As usually implemented, the threshold is a function of the image data along a scan line [Serreyn78] or within a window. The threshold will work badly if a window contains no object or multiple objects with different intensities. An isolated small object may be overlooked in a large window, and a large object may be thresholded inconsistently across small windows.

Histograms computed over regions of mixed sizes are difficult to segment. Weszka *et al.* [Weszka74] suggest computing the histogram only over pixels near region boundaries (*i.e.*, pixels with high gradient). Further discussions of edge detection and texture analysis to set thresholds may be found in [Weszka78] and [Kohler81].

Panda and Rosenfeld [Panda78] found that intensity/edge-strength histograms of FLIR targets are trimodal, with peaks representing background, edge, and object. It was found insufficient to set a single threshold at the intensity value of the edge peak. Better methods used edge gradient to implement decision boundaries extending from the edge peak to the valley between the background and object peaks.

A.3. Iterative Modification

An alternative to adaptive thresholding is context-sensitive modification of the image itself. This is typically done by iterative relaxation or "competitive-cooperative" processes [Troy73, Hummel78, Zucker78, Kirby79, Nagin79, Eklundh80, Peleg80], although single-pass methods such as cluster analysis and pixel classification could be adapted to this purpose. (Relaxation output might be useful in training such a classifier.)

Unfortunately relaxation processes tend either to do very little or to be very sensitive to the updating rule, the image-dependent compatibility coefficients, or the class membership function for initially labeling each pixel. Various schemes have been proposed for estimating these quantities. Histogram segmentation, for instance, can be used to select the initial class membership function [Ranade80].

One use of relaxation is to get the segmenter started on scenes (or composite regions) with unimodal histograms [Bhanu82]. The relaxation process emphasizes spatial features that are too weak or space-variant to show up in the histogram. Such preprocessing can split a composite peak into subpeaks that are useful to a threshold segmenter. This is in contrast to relaxation methods applied to the histogram (see Section A.2), which can reduce the number of peaks but never create new ones.

A.4. Recursive Splitting

Uniform regions can be found by recursively splitting nonuniform regions (beginning with the whole image) into smaller regions. In the limit this produces single-valued and perhaps single-pixel regions. In some cases it may be desirable to split even uniform regions using region shape criteria [Lemkin79, Rutkowski81].

Since almost any area can be better represented (in a mean-square-error sense) by two small regions than by a single large one, it is difficult to determine when to stop splitting. Most splitting methods lack a justifiable stopping criterion. One possibility, derived from coding and information theory, is to use the number of bits required to

code a region before and after splitting as a measure of improvement; this is unfortunately dependent on the coding method.

Splitting is always costly since region descriptors (shape, variance, *etc.*) must be computed for each subregion. Suppose that a region is split into d subregions: all pixels in at least $d-1$ subregions must be reexamined to compute the new descriptors. To segment an image down to the pixel level requires

$$N^2 \left(1+\frac{d-1}{d}\log_d N^2\right)$$

pixel examinations, as opposed to N^2 for segmentation by merging or growing procedures.

The above analysis assumes a deterministic splitting algorithm. In quadrant subdivision, for example, regions are repeatedly split into four square subregions until homogeneous regions are found. (The number of pixels in a row or column is typically a power of two, making the subdivision trivial.) This method segments too finely so that a later merging step is required; even so, it is one of the fastest partitioning methods.

The most difficult step in other partitioning methods is deciding exactly where the new boundary should go. If the new boundary location is not known *a priori*, the region descriptors must be computed for each possible boundary. This can involve a very large search space and enormous computational costs. Functional approximation schemes [Pavlidis72] avoid this by using parametric solutions for the boundary and for the region descriptors. The PHOENIX algorithm offers another solution by choosing boundaries along significant intensity contours.

A.5. Classification

The purpose of segmentation is often classification. This can be reversed by using pixel classification to achieve segmentation. The basic problem is to classify an image window as one of several texture types. For a survey of multispectral classification in remote sensing see Haralick [Haralick76].

The method of maximum likelihood could be used if we had enough information about the texture classes. We would estimate the likelihood of the observed pattern under each hypothesis, then choose the texture class giving the highest likelihood. Unfortunately the required probability distributions are too large to be represented as histograms.

Nonparametric methods have been proposed for estimating and storing large distributions; see, for example, the set covering procedures of Read and Jayaramamurthy [Read72] and McCormick and Jayaramamurthy [McCormick75]. It seems sensible, however, to assume a parametric form for the distributions whenever it is possible to do so. This allows us to develop simple vector product scores for classifying pixels.

Image intensities seem to be well characterized by statistical moments. Ahuja *et al.* [Ahuja77] show that the first few moments are as useful as an entire histogram for classifying textures. Statistical methods have also been developed for classifying the spatial distributions of texture pixels [Haralick73, Mitchell78, Rosenfeld79, Laws80].

A simple nonparametric approach is to store an exemplar (or feature vector) for each known texture type. Each pixel to be classified is compared to each exemplar

and is assigned to the class of the most similar one. This has the advantage that it is easy to add additional texture exemplars.

The principal difficulty with any type of texture classification is that the region to compute texture statistics over cannot be known unless segmentation has already been accomplished. Typical image processing problems require analyses near the resolution limit of the imagery, and windowing errors are intolerable.

A.6. Clustering

Cluster analysis is identical to classification except that the classes are not known *a priori*. Various spectral or spatial descriptors of the pixels are analyzed for similarities, and those pixels that are judged similar to each other or to some prototypical seed pixels are assigned to the same class [Wacker69, Carlton77, Goldberg78, Yoo78, Coleman79, Mitchell79, Schachter79]. Spatial analysis then completes the segmentation; this analysis may include probabilistic relaxation [Nagin79, Kohler81] or other methods of noise cleaning and boundary smoothing. Clustering can also be used to merge regions found by thresholding or other methods [Haralick75a].

PHOENIX-style histogram segmentation is a type of cluster analysis. This is more evident when done in a multivariate space [Schachter75, Schachter77, Hanson78, Milgram79, Schachter79, Milgram80]. Multivariate histograms are typically quantized very coarsely in each feature in order to reduce storage requirements and analysis time. If finer quantization is required, either two passes should be made through the data (planning), or an adaptive accumulator scheme should be used [Schachter75, O'Roarke81, Sloan81]. Perhaps a better alternative is to use a parametric or adaptive (*perceptron*) cluster method not relying on histograms.

A.7. Region Growing

Region growing is based on the premise that it is easier to identify interior pixels than border pixels. One starts with a set of region seeds, preferably one seed per image region. Each region is then expanded like a wavefront, incorporating adjacent unassigned pixels. Growth stops when all pixels have been absorbed or when unassigned pixels are too dissimilar to be merged with adjacent regions. An editing phase may follow in which unassigned pixels are classified and neighboring regions are tested to see if they can be merged.

One method is to start with completely homogeneous regions and then merge neighbors that have statistically-similar pixel populations or classifications [Muerle68, Gupta74]. Another is to merge neighbors that are divided by "weak" boundaries or that together form a simple shape (the "phagocyte" heuristic) [Brice70]. Yet another is to accept any unassigned pixel as a region seed and to grow the region until its natural limits are found. The regions may be grown either sequentially [Jarvis75] or in parallel during a single scan [Yakimovsky76]. Any of these methods essentially combine connected-component extraction with region growing.

Region seeds are usually found by crude segmentation, retaining as seeds only those pixel groups most certain to belong together. The seeds may be chosen interactively [Garvey76b] or automatically. Often the seeds are chosen by adaptive thresholding or peak-finding algorithms applied to a gradient or edge transformation of the image. The segmentation is then done on the original image data. (Region growing typically

uses only monochrome input, although see Kettig [Kettig76].)

Levine and Leemet [Levine76] have developed an interesting method of obtaining region seeds from an edge map. The edges are thickened by pyramid reduction. As the successive reductions occur, they isolate and eventually cover the pixels in the more uniform region interiors. The last pixels to be enveloped are chosen as region seeds. The growing process that follows is essentially the reverse of this region shrinking.

The order in which pixels are considered for merging is a major concern. Truly parallel "best first" growth can be implemented on a sequential machine only by expensive schemes to repeatedly examine eligible pixels. Single scan methods have been proposed [Yakimovsky76, Somerville76], although a second scan is necessary to label the region map.

Deciding whether to merge a pixel with an adjacent region is equivalent to a one-sided hypothesis test. Some measure of membership must be computed and some threshold must be used. Often the pixel is compared with the region mean, using the region variance to set a threshold. Somerville and Mundy [Somerville76] use a planar approximation to the region, thus allowing for slope in the luminance function. Other researchers have compared the unassigned pixel only to the region pixels nearest it. For a survey of techniques see Zucker [Zucker76a].

A major problem with region growing is leakage, similar to chaining in cluster analysis. Two very dissimilar regions may be joined by an area of intermediate appearance: it is then possible for one region to grow across the neck and absorb pixels belonging to the other region. This can be remedied by recursive splitting or by a split-and-merge editing phase, but greatly complicates the segmentation process.

A.8. Merging

Another approach is region merging, beginning with uniform or single pixel regions. Those regions sharing a common border are eligible for merging. The border is eliminated if the combined region is sufficiently homogeneous. This differs from region growing in that both regions to be merged may be larger than one pixel.

The decision of whether to merge two regions can be based on the strength of the boundary between them. This leads to trouble when two distinct regions share a blurred or indistinct border. Merging can also be treated as a hypothesis test: the two regions are combined only if this gives an acceptable planar fit to the data.

The results of region merging may depend strongly on the order in which region pairs are tested for merging. Order independence may be achieved by considering all merges in parallel and allowing only the best merge to occur at any one time. This requires extra computation and "bookkeeping," leading many investigators to develop approximations to best-first merging.

Merging algorithms avoid recomputation if the uniformity measure for a combined region is a function of the statistics of its subregions. Maximum and minimum pixel values, for instance, can be computed from the subregion extrema: only the initial N^2 pixel examinations are needed. Unfortunately a large number of storage locations (N^2 per feature in theory, but less in practice) are required to hold the region statistics. Elaborate data structures may also be required to keep track of the numerous

irregularly-shaped regions.

Semantic merging integrates segmentation with interpretation. Yakimovsky and Feldman [Yakimovsky73a, Yakimovsky73b, Feldman74] suggest using real-world probabilities of region-type adjacencies. Such probabilities may be obtainable for limited domains such as X-ray analysis. A similar approach to region labeling has been proposed by Tenenbaum and Barrow [Barrow76, Tenenbaum76a, Tenenbaum76b].

A.9. Split-Merge

Most researchers using splitting or merging techniques alone have acknowledged the need for the complementary process as an editing step. At any stopping point in a segmentation there are usually some regions that should be split further and some that should be merged.

Merging techniques generally consider only two subregions at a time, and the final partitioning depends on the order of these comparisons. Splitting techniques are similarly limited by the order in which histogram peaks are chosen. The best possible partioning, by any particular criterion, might not be reachable by either technique. Integrated (or iterated) splitting and merging may also fall short of this ideal, but the combination is able to explore a larger space of possibilities.

Split-merge methods do not require accurate region seeds. Horowitz and Pavlidis [Horowitz74] start with arbitrary square neighborhoods. (This is particularly useful for computing Fourier texture measures over the seed regions [Pavlidis75].) Their algorithm breaks the nonuniform squares into uniform seeds, then combines neighboring fragments that are similar. The similarity measure may be based on intensity or on texture properties [Chen79]. No connected-components analysis is necessary if a segmentation tree is maintained.

Split-merge methods are able to use local information to determine each splitting, but the region boundaries tend to "cling" to the major rectilinear divisions. The splitting steps integrate well with quadtree representation of segmentation maps [Horowitz74, Klinger76, Hunter79, Samet79], but a merging step tends to destroy the quadtree structure. More elaborate linked tree structures have been developed [Burt80, PietikäinenB2] to solve this problem.

Although these methods have become strongly linked to quadtree representations, it is important to note that a split-merge approach is compatible with chain-code outlines, binary overlays, region maps, or other representations.

A.10. Spanning-Tree Methods

Several researchers have proposed tree structures to model the hierarchical structure of a scene. (Often neighbor relationships are stored, making the structure a graph rather than a tree.) The root node is the image itself; leaf nodes are the individual pixels or homogeneous regions. The scene may be segmented at any resolution by cutting branches of the tree [Kirsch71, Freuder76, Horowitz76, Horowitz78].

Burr and Chien [Burr76] apply minimal spanning tree methods to find strongly linked pixel groups separated from each other by weak links. The one-pass segmentation method of Yakimovsky [Yakimovsky76] builds a spatially constrained approximation

to the minimal spanning tree; it could be called a minimal spanning maze. A very similar segmentation system is developed in Narendra [Narendra77, Narendra80].

The spanning-tree methods all require that region interiors be smoother than border neighborhoods. They are thus unsuitable for locating textured regions unless the textures can be transformed to one or more feature planes with the property of region homogeneity. Macrotextures must be analyzed by identifying the primitive elements, then using structural methods to find texture regions.

A.11. Segmentation Editing

The preceding methods provide the best segmentation possible using local statistical analysis. The purpose of an editing phase is to improve the segmentation by using more global or application-dependent knowledge.

It is much easier to merge regions than to split them, since splitting requires that the best boundary be identified. Images are thus nearly always oversegmented to simplify the editing or interpretation phases that follow.

Syntactic editing analyzes the properties of regions and their spatial relationships. Pavlidis *et al.* [Tanimoto77, Horowitz78] use region adjacency graphs to identify noise regions. These are deleted and the pixels are reassigned to neighboring regions. Riseman and Arbib [Riseman77] use region adjacency graphs to identify composite textures. The regions are considered texture elements, and it is desired to find larger regions containing distinctive distributions of these primitives.

One of the main reasons for segmentation of textured images is to permit region-by-region classification, which should be more accurate than pixel-by-pixel methods. The classification can be done using multispectral discriminant analysis [Gupta74], cluster analysis on within-region textures [Lumia81], or model-based shape analysis [Brenner77, Pavlidis78, Jain80, Rutkowski81]. Primitive regions can then be merged if their signatures are classified identically.

Semantic merging integrates region growing with interpretation [Yakimovsky73a, Yakimovsky73b, Feldman74, Barrow76, Garvey76a, Garvey76b, Tenenbaum76a, Tenenbaum76b, Tenenbaum80, Fischler82]. Probabilities of region-type adjacencies may be even more applicable at the final editing and classification stage [Lumia81].

Another form of editing uses initial region knowledge to guide a more sophisticated segmenter. It may be possible, by examining the initial edge and interior points, to infer a classifying rule or grammar [Keng77a, Keng77b]. This *bootstrap* information can then be used to resegment the scene or to segment other similar scenes. Bootstrapping is particularly effective if ground-truth segmentations are used to infer the rules.

There is no reason why editing must be limited to a single pass. Iterative parallel algorithms have been suggested [Rosenfeld76b, Riseman77] in which each pixel's label or region membership is repeatedly updated as a function of its neighbors' labels. These competitive-cooperative processes have also been used for edge thinning and edge linking [Zucker76b, Zucker77]. The methods are very flexible and powerful, but little is known about constructing the label assignment functions.

After a scene has been segmented into regions, it is still necessary to determine which of these regions belong to composite objects. Even a simple object such as an

untextured block may have several distinct regions because of lighting effects. On the other hand, a single uniform region may be segmentable into a stack of blocks or a clump of particles on the basis of its outline [Arcelli71]. (The Price segmenter performs some shape analysis and region editing during connected-component extraction. This is a rather expensive step, and PHOENIX has left it for an external editing program.)

There have been many attempts to combine segmentation with semantic interpretation in natural scenes; see, for instance, [Preparata72, Tenenbaum73, Feldman74, Barrow76, Garvey76a, Garvey76b, Price76b, Sakai76, Tenenbaum76a, Tenenbaum76b, Levine77, Faugeras80, Tenenbaum80, Price81, Fischler82]. Such recognition requires domain-specific knowledge beyond the scope of this study.

Appendix B

Connected Component Extraction

The following information on the connected component extraction algorithm was provided by Duane Williams of Carnegie-Mellon University as part of the PHOENIX code. (For the algorithm used in the Ohlander/Price segmenter, see [Ohlander78]. Another algorithm is given in Kelly [Kelly70, pp. 54-55].)

This algorithm is the connected region extraction algorithm, *reganal*, developed for the KIWI segmentation program at Carnegie-Mellon University. It is based upon the method of Agrawala and Kulkarni [Agrawala77]. This implementation (and that of KIWI) differs, however, in several points from their algorithm.

This algorithm takes a binary image, and produces a list of descriptions of the component regions (patches) and their pixels (strips). The patches are represented by *patch* records, and include shape features and an indication of which patch contains this one (*i.e.*, surrounds it). The strips are described by *strip* records, which include a row, the columns on which the strip begins and ends, and a link to the next strip. The input image is actually a map of the interval numbers resulting from thresholding; this procedure is executed once for each interval, and considers pixels in that interval (given by the parameter *val*) to be '1,' all others to be '0.' A border of 0's is assumed to surround the image.

The algorithm proceeds by forming, for each row, a description of the strips of that row. This description includes, for each consecutive run of 1's or 0's, the column at which the run starts. The run ends one column before the next run starts. The runs for each row are compared with the runs of the previous row, by examining the locations of the endpoints, to determine how to propagate partial region labels from the previous row to the current row. The examination is performed by the *assign* procedure. This procedure can perform five actions: create a new region of 1's (newbody), create a new region of 0's (newhole), propagate a label from an existing region (extend), end a region of 1's (endbody), end a region of 0's (endhole).

The actions in *assign* take place within a big loop that scans one *segment* (run of 1's followed by a run of 0's) in the previous row, dealing with all segments in the current row that are encountered. At their leftmost endpoint, the runs in the current row are labeled. This big loop may encounter eight situations: four while scanning the 0's before the ones, and four while scanning the 1's before the next 0. Here, pictorially, are the possible situations; the letters A, B, *etc.*, mark the start of the next run; + indicates a 1 and - indicates a 0:

```
Case 1:
                prev. Row:      ...+++A----------B++++...
                This row:       ...+++++++W---...
                Description:    a run of 1's extends from before A, into the
                                                hole AB.
                Actions:        extend (AB) to (W...)
                Reasons:        the hole W... Touches the hole AB;
                                                the run ...W has already been labeled.
```

```
Case ii:
                prev. Row:      ...+++A------------B++++...
                This row:       ...++++++++++++++++++...
                Description:    a run of 1's extends from before A until
                                        somewhere past B.
                Actions:        endhole (AB)
                reasons:        the hole AB cannot continue below; the
                                        run on this row has already been labeled.


Case iii:
                prev. Row:      ...+++A------------B++++...
                This row        ...---V++W---...
                Description:    a run of 1's begins and ends within the
                                        run AB.
                Actions:        newbody (VW); extend (AB) to (W...)
                Reasons:        the body VW is created; the hole W...
                                        Touches the hole AB.


Case iv:
                prev. Row:      ...+++A------------B++++...
                This row:       ...---V+++++++...
                Description:    a run of 1's starts between A and B, and
                                        continues past B
                Actions:        extend (B...) to (V...)
                Reasons:        the body V... Touches the body B...
                                        The hole ...V has already been labeled.


Case v:
                prev. Row:      ...---B++++++++++++C----...
                This row:       ...-------V+++...
                Description:    a hole starts before B, and ends before C.
                Actions:        extend (BC) to (V...)
                Reasons:        the body V... Touches the body BC; the hole
                                        ...V has already been labeled.


Case vi:
                prev. Row:      ...---B++++++++++++C----...
                This row:       ...--------------------...
                Description:    a hole starts before B and ends after C.
                Actions:        endbody (BC)
                reasons:        the body BC does not continue below; the hole
                                        on this row has already been labeled.


Case vii:
                prev. Row:      ...---B++++++++++++C----...
                This row:       ...++U---V++...
                Description:    a hole starts and ends within the run BC.
                Actions:        newhole (UV); extend (BC) to (V...)
                Reasons:        the hole UV is new; the body V... Touches BC.


Case viii:
                prev. Row:      ...---B++++++++++++C----...
                This row:       ...++U----------...
                Description:    a hole starts between B and C, and ends
                                        after C.
                Actions:        extend (C...) to (U...)
                Reasons:        the body ...U has already been labeled; the
                                        hole U... Touches the hole C...
```

There are two cases not covered here: the case in which a hole starts before A and ends after B, and the case in which a run of 1's starts before B and ends after C. These cases need not be examined, since they involve no new bodies or holes, no ending bodies or holes, and no propagation of labels.

There are also two cases not completely examined, cases ii and vi, in which a body or hole ends. In case ii, we must note the fact that body B... is touching the run on the current row, which is touching body two possibilties. If ...A and B... are the same partial region then the hole AB is completely contained within that partial region; if ...A and B... are different, they are to be merged together. Both cases are discussed below in more detail. Similarly, in case vi, holes ...B and C... touch each other. In the program, these cases are handled by the *endbody* and *endhole* procedures.

It must also be kept in mind that a single partial region may, by cases iii and vii, be split up into any number of runs on a single row; some of these may be ended, some merged, and some extended on any given row of the image. So, we must keep track of exactly what has happened to a partial region throughout the scan of the entire row; then, we can do drastic things (like declaring a partial region to be really at its end) at the end of the row.

There is always an issue, in connectivity algorithms, of the exact definition of connectivity. Two definitions are the most common: 4-connectivity and 8-connectivity.

The definitions are these:

```
4-connected      8-connected
    x              xxx
   x+x             x+x
    x              xxx
```

(the pixel + is connected to all pixels x.)

For reasons pointed out by Rosenfeld [Rosenfeld76a], it is frequently desirable to have objects (*i.e.*, 1's) be 4-connected and holes (*i.e.*, 0's) 8-connected, or vice versa. In fact, this algorithm depends on this distinction. For the segmentation program, it is necessary to have objects be 4-connected in order to avoid some infinite-loop situations, for example, if the input is alternating 1 and 0 pixels, like a checkerboard. So, holes are 8-connected and objects are 4-connected. This may be reversed (objects 8-connected and holes 4-connected) by converting all the '<' signs in *assign* (where column numbers are being compared) to '<=,' and all the '<=' signs (again, only for comparisons of column numbers) to '<'.

A single row is represented by the *line* data structure. This contains the number of segments, *Lsegs*; the segments *Lseg* themselves; and two counters: *Lcurseg* and *Lcol*. These counters are used in *assign* for indicating the *current* segment (Lcurseg) and the column on which the next segment begins (Lcol). Each segment record indicates the column of the first 1, the column of the first 0; and the partial-region labels assigned to the 1's and the 0's.

There is assumed to be a region of 0's surrounding the image; this is called *outside*, and is represented by partial region *preg_outside*. This is accomplished by the following steps:

- The *firstrow* procedure pretends there is a row of 0's from before the first column until past the last one.
- The *runcode* procedure pretends there are 0's from the last 1 until past the end of the image.
- The *extend* procedure pretends there is another segment to the left of the first segment of the row, which has already been labeled as *outside*.
- The merging procedure (*etc.*) always merges other partial regions into *outside*; never *outside* into another partial region.
- The region description for the *outside* region is not meaningful, and may contain garbage.
- The *lastrow* procedure pretends there is a row of 0's from before the first column until past the last one.

Peculiarities of PR_desc:

The fields of the region description of a partial region (*PR_desc*) are used in a special way: *R_rstart* is indeed the first row of the partial region. *R_rows*, however, is the last row number rather than the number of rows. Similarly, *R_cstart* is the starting column, but *cols* is the ending column. *R_area*, *R_holes*, and *R_harea* are normal. The centroid, however, is accumulated in *R_rcent* and *R_ccent* as the sum of the row (and column) number for each pixel of the partial region. Then, when the patch record is written, *R_rcent* is divided by *R_area* (also, *R_ccent* is divided by *R_area*) to compute the actual coordinates of the centroid.

The most important piece of information in the patch record is the link to the containing patch, *P_outer*. However, as stated in the paragraph above, the patch record is created when its partial region(s) comes to an end; this is before the patch record for the containing patch has been created! So, it is necessary to remember, when a patch is created, which partial region contains it. Then, when a patch is made from this partial region, the link in the contained patch can be updated. This remembering is accomplished via the *PR_inner* and *P_next* links: each partial region points (via *P_inner*) to a patch it contains, which points (via *P_next*) to the next one, and so on. When the partial region is converted to a patch, this list is scanned, and the new patch number is placed into the *P_outer field*.

There is one problem with the above structure: when partial region A is merged into partial region B, both A and B have these lists of contained patches. The lists could be combined by traversing one list and updating the link of the last patch, *etc*. However, the lists may become quite long, and it is not attractive to have to scan through them (potentially many times, as partial regions are merged). So, instead, each partial region has a list of other partial regions that have been merged into it (*PR_piece*), with the last partial region on the list containing *PREG_NIL* as its *PR_piece* field. When a partial region is converted to a patch, this list is traversed and all the patches contained by all these partial regions are updated. The partial regions may then be freed so they may be used again.

There is, however, a further problem. Since all these merged partial regions are kept around, there may be references to them (*i.e.*, segments that are labeled with these merged partial regions, other partial regions indicating that these partial regions surround them, *etc.*). So, whenever such a reference is made, it is necessary to find which partial region is really indicated (thus, if A is merged into B and we refer to A, we really want to talk about B). The *PR_whole* field is a link to the partial region used after merging, and the *root* function traces down these links to find the intended partial region. Note that *PR_piece* is not the exact inverse of *preg_whole*. The *PR_whole* fields form a list from the active partial region through all those partial regions merged with it. If, however, A is merged with B and B is merged with C, then the *PR_whole* field of A points to B and *PR_whole* of B points to C. If, then, D is merged into C, PR_whole of D also points to C. In this example, the *PR_piece* fields form a real linked list:

```
C -> D -> B -> A
```

while the *PR_whole* fields form a tree:

```
A
!
B D
!/
C
```

(with links pointing down, in this picture).

The run codes normally indicate a row, the columns at which the run starts and ends,

and a link to the next run for the same region. When partial regions are merged, they each indicate a linked list of runs; somehow, these must be merged as well. This is accomplished by a special run whose row number is the special value *S_MERGE*. This run has two fields: pointers to the two linked lists to be merged. During the actual traversal of the runs, both lists must be examined when a merge run is encountered.

The column numbers used in this procedure are sometimes tricky. Normally, for each run of 1's and 0's (*i.e.*, in the segment record), the column of the start of each run is stored. This means that the last column of a run of 1's is actually the start of the next run of 0's, minus one. In the partial region records, *cols* is this value; actually, one plus the rightmost column of the partial region. When patch records are created, the proper conversion is performed. Also, when run records are stored, the column of the end of the run is really the last column of the run; *i.e.*, 1 has already been subtracted.

References

[Abdou79] I.E. Abdou and W.K. Pratt, "Quantitative Design and Evaluation of Enhancement/Thresholding Edge Detectors," *Proc. IEEE*, Vol. 67, No. 5, pp. 753-763, May 1979.

[Aggarwal77] R.K. Aggarwal and J.W. Bacus, "A Multi-Spectral Approach for Scene Analysis of Cervical Cytology Smears," *J. Histochem. Cytochem.*, Vol. 25, pp. 668-680, 1977.

[Aggarwal78] R.K. Aggarwal, "Adaptive Image Segmentation using Prototype Similarity," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Chicago, pp. 354-359, 1978.

[Agrawala77] A.K. Agrawala and A.V. Kulkarni, "A Sequential Approach to the Extraction of Shape Features," *Computer Graphics and Image Processing*, Vol. 6, pp. 538-557, 1977.

[Ahuja77] N. Ahuja, L.S. Davis, R.M. Haralick, and D.P. Panda, *Image Segmentation Based on Local Gray Level Patterns*, Report TR-551, U. of Maryland, College Park, June 1977.

[Ali79] M. Ali, W.N. Martin, and J.K. Aggarwal, "Color-Based Computer Analysis of Aerial Photographs," *Computer Graphics and Image Processing*, Vol. 9, pp. 282-293, 1979.

[Arcelli71] C. Arcelli and S. Levialdi, "Picture Processing and Overlapping Blobs," *IEEE Trans. on Computers*, pp. 1111-1115, Sep. 1971.

[Barrett81] W.A. Barrett, "An Iterative Algorithm for Multiple Threshold Detection," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Dallas, pp. 273-278, Aug. 1981.

[Ballard82] D.H. Ballard and C.M. Brown, *Computer Vision*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1982.

[Barrow75] H.G. Barrow and J.M. Tenenbaum, *Representation and Use of Knowledge in Vision*, TN 108, Artificial Intelligence Center, SRI International, July 1975.

[Barrow76] H.G. Barrow and J.M. Tenenbaum, *MSYS: A System for Reasoning about Scenes*, TN 121, Artificial Intelligence Center, SRI International, Apr. 1976.

[Barrow81] H.G. Barrow and J.M. Tenenbaum, "Computational Vision," *Proc. IEEE*, Vol. 69, No. 5, pp. 572-595, May 1981.

[Bhanu82] B. Bhanu and O.D. Faugeras, "Segmentation of Images Having Unimodal Distributions," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-4, No. 4, pp. 408-419, July 1972.

[Brady82] M. Brady, "Computational Approaches to Image Understanding," *Computing Surveys*, Vol. 14, No. 1, pp. 3-71, Mar. 1982.

[Brenner77] J.F. Brenner *et al.*, "Scene Segmentation Techniques for the Analysis of Routine Bone Marrow Smears from Acute Lymphoblastic Leukemia Patients," *J. Histochem. Cytochem.*, Vol. 25, pp. 601-613, 1977.

[Brice70] C.R. Brice and C.L. Fennema, "Scene Analysis using Regions," *Artificial Intelligence*, Vol. 1, pp. 205-226, Fall 1970.

[Bullock76] B.L. Bullock, "Finding Structure in Outdoor Scenes," in C.H. Chen (ed.), *Pattern Recognition and Artificial Intelligence*, Academic Press, New York, 1976.

[Burr76] D.J. Burr and R.T. Chien, "The Minimal Spanning Tree in Visual Data Segmentation," *Proc. 3rd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 519-523, Nov. 1976.

[Burt80] P. Burt, T.H. Hong, and A. Rosenfeld, *Segmentation and Estimation of Image Region Properties through Cooperative Hierarchical Computation*, TR-927, Computer Vision Laboratory, Computer Science Center, Univ. of Maryland, College Park, Aug. 1980.

[Cahn77] R.L. Cahn, R.S. Poulsen, and G. Toussaint, "Segmentation of Cervical Cell Images," *J. Histochem. Cytochem.*, Vol. 25, pp. 681-688, 1977.

[Carlton77] S.G. Carlton and O.R. Mitchell, "Image Segmentation using Texture and Gray Level," *Proc. Image Understanding Workshop*, pp. 71-77, Apr. 1977.

[Chen79] P.C. Chen and T. Pavlidis, "Segmentation by Texture Using a Co-Occurrence Matrix and a Split-and-Merge Algorithm," *Computer Graphics and Image Processing*, Vol. 10, pp. 172-182, 1979.

[Chow70] C.K. Chow and T. Kaneko, *Boundary Detection of Radiographic Images by a Threshold Method*, IBM Research Report RC-3203, 1970. Also in *Proc. IFIP 71*, TA-7, p. 130, 1971, and in S. Watanabe (ed.), *Frontiers of Pattern Recognition*, Academic Press, NY, pp. 61-82, 1972.

[Clark81] S.J. Clark, *Image File Naming Conventions*, IUS Document CMU004, Computer Science Dept., Carnegie-Mellon Univ., Mar. 1981.

[Coleman79] G.B. Coleman and H.C. Andrews, "Image Segmentation by Clustering," *Proc. IEEE*, Vol. 67, No. 5, pp. 773-785, May 1979.

[Danker81] A.J. Danker and A. Rosenfeld, "Blob Detection by Relaxation," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-3, No. 1, pp. 79-92, Jan. 1981.

[Davis75] L.S. Davis, "A Survey of Edge Detection Techniques," *Computer Graphics and Image Processing*, Vol. 4, No. 1, pp. 248-270, Sep. 1975.

[Davis80] L.S. Davis and A. Mitiche, "Edge Detection in Textures," *Computer Graphics and Image Processing*, Vol. 12, No. 1, pp. 25-39, Jan. 1980.

[Davis82] L.S. Davis and A. Mitiche, "MITES (mit-es): A Model-Driven, Iterative Texture Segmentation Algorithm," *Computer Graphics and Image Processing*, Vol. 19, pp. 95-110, 1982.

[Deal79] B. Deal, C.M. Lo, R. Taylor, V. Norwood, H. Henning, T. Daggett, T. Noda, J. Powers, G. Guzman, H. Greenberger, G. Towner, and G. Parker, *Automatic Target Cuer First Quarter Report*, Northrop Corp., Electro-Mechanical Division, Anaheim, CA, Report NORT-79Y100, Oct. 1979.

[Deguchi78] K. Deguchi and I. Morishita, "Texture Characterization and Texture-Based Image Partitioning Using Two-Dimensional Linear Estimation Techniques," *IEEE Trans. on Computers*, Vol. C-27, No. 8, pp. 739-745, Aug. 1978.

[Dyer81] C.R. Dyer, "A VLSI Pyramid Machine for Hierarchical Parallel Image Processing," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Dallas, pp. 381-386, Aug. 1981.

[Ehrich76] R. Ehrich and J.P. Foith, "Representation of Random Waveforms by Relational Trees," *IEEE Trans. on Computers*, Vol. C-25, pp. 725-736, July 1976.

[Eklundh80] J.O. Eklundh, H. Yamamoto, and A. Rosenfeld, "A Relaxation Method for Multispectral Pixel Classification," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 1, pp. 72-75, Jan. 1980.

[Faugeras80] O.D. Faugeras and K.E. Price, "Semantic Description of Aerial Images Using Stochastic Labeling," *Proc. ARPA Image Understanding Workshop*, pp. 89-94, Apr. 1980. Also in *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-3, No. 6, pp. 633-642, Nov. 1981.

[Feldman74] J.A. Feldman and Y. Yakimovsky, "Decision Theory and Artificial Intelligence: I. A Semantics-Based Region Analyzer," *Artificial Intelligence*, Vol. 5, pp. 349-372, Winter 1974.

[Fischler79] M.A. Fischler, G.J. Agin, H.G. Barrow, R.C. Bolles, L. Quam, J.M. Tenenbaum, and H.C. Wolf, *Interactive Aids for Cartography and Photo Interpretation*, Final Technical Report, Artificial Intelligence Center, SRI International, 1979.

[Fischler82] M.A. Fischler, S.T. Barnard, R.C. Bolles, M. Lowry, L. Quam, G. Smith, and A. Witkin, *Modeling and Using Physical Constraints in Scene Analysis*, TN 267, Artificial Intelligence Center, SRI International, Sep. 1982.

[Fischler83] M.A. Fischler and H.C. Wolf, *A General Approach to Machine Perception of Linear Structure in* Imaged Data," TN 276, Artificial Intelligence Center, SRI International, Feb. 1983.

[Fram75] J.R. Fram and E.S. Deutsch, "On the Quantitative Evaluation of Edge Detection Schemes and their Comparison with Human Performance," *IEEE Trans. on Computers*, Vol. C-24, No. 6, pp. 616-628, June 1975.

[Freuder76] E. Freuder, "Affinity: A Relative Approach to Region Growing," *Computer Graphics and Image Processing*, Vol. 5, pp. 254-264, 1976.

[Fu81] K.S. Fu and J.K. Mui, "A Survey on Image Segmentation," *Pattern Recognition*, Vol. 13, pp. 3-16, 1981.

[Garvey76a] T.D. Garvey, *Perceptual Strategies for Purposive Vision*, TN 117, Artificial Intelligence Center, SRI International, Sep. 1976.

[Garvey76b] T.D. Garvey and J.M. Tenenbaum, *Application of Interactive Scene Analysis Techniques to Cartography*, TN 127, Artificial Intelligence Center, SRI International, Sep. 1976.

[Goldberg76] M. Goldberg and S. Shlien, "A Cluster Scheme for Multispectral Images," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-6, pp. 86-92, 1976.

[Grimson80] W.E.L. Grimson, "Aspects of a Computational Theory of Human Stereo Vision," *Proc. Image Understanding Workshop*, College Park, MD, pp. 128-149, Apr. 1980.

[Gupta74] J.N. Gupta and P.A. Wintz, "Computer Processing Algorithm for Locating Boundaries in Digital Pictures," *Proc. 2nd Int. Jnt. Conf. on Pattern Recognition*, Copenhagen, pp. 155-156, Aug. 1974.

[Gurari82] E.M. Gurari and H. Wechsler, "On the Difficulties Involved in the Segmentation of Pictures," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-4, No. 3, pp. 304-306, May 1982.

[Hanson74] A.R. Hanson and E.M. Riseman, *Preprocessing Cones: A Computational Structure for Scene Analysis*, Computer and Information Science Report 74C-7, Univ. of Mass. at Amherst, Sep. 1974.

[Hanson75a] A.R. Hanson and E.M. Riseman, *The Design of a Semantically Directed Vision Processor*, Computer and Information Science Report 75C-1, Univ. of Mass. at Amherst, Feb. 1975.

[Hanson75b] A.R. Hanson, E.M. Riseman, and P. Nagin, *Region Growing in Textured Outdoor Scenes*, Computer and Information Science Report 75C-3, Univ. of Mass. at Amherst, Feb. 1975.

[Hanson78] A.R. Hanson and E.M. Riseman, "Segmentation of Natural Scenes," in A.R. Hanson and E.M. Riseman (eds.), *Computer Vision Systems*, New York: Academic Press, 1978. See also color plate 5-4 in [Ballard82].

[Haralick73] R.M. Haralick, K. Shanmugam, and I. Dinstein, "Textural Features for Image Classification," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-3, pp. 610-621, Nov. 1973.

[Haralick75a] R.M. Haralick and I. Dinstein, "A Spatial Clustering Procedure for Multi-Image Data," *IEEE Trans. on Circuits and Systems*, Vol. CAS-22, pp. 440-450, 1975.

[Haralick75b] R.M. Haralick, "A Resolution Preserving Textural Transform for Images," *Proc. Computer Graphics, Pattern Recognition, and Data Structure*, Los Angeles, pp. 51-61, May 1975.

[Haralick76] R.M. Haralick, "Automatic Remote Sensor Image Processing," in A. Rosenfeld (ed.), *Digital Picture Analysis*, Berlin, Germany: Springer, pp. 5-63, 1976.

[Haralick80] R.M. Haralick, "Edge and Region Analysis for Digital Image Data," *Computer Graphics and Image Processing*, Vol. 12, pp. 60-73, 1980.

[Haralick81] R.M. Haralick, "The Digital Edge," *IEEE Conf. on Pattern Rec. and Image Processing*, Dallas, pp. 285-291, Aug. 1981.

[Harlow73] C.A. Harlow and S.A. Eisenbeis, "The Analysis of Radiographic Images," *IEEE Trans. on Computers*, Vol. C-22, pp. 678-688, 1973.

[Horn77] B.K.P. Horn, "Understanding Image Intensity," *Artificial Intelligence*, Vol. 8, pp. 201-231, 1977.

[Horowitz74] S.L. Horowitz and T. Pavlidis, "Picture Segmentation by a Directed Split-and-Merge Procedure," *Proc. 2nd Int. Jnt. Conf. on Pattern Recognition*, Copenhagen, pp. 424-433, Aug. 1974.

[Horowitz76] S.L. Horowitz and T. Pavlidis, "Picture Segmentation by a Tree Traversal Algorithm," *J. of the ACM*, Vol. 23, No. 2, pp. 368-388, Apr. 1976.

[Horowitz78] S.L. Horowitz, "A Graph-Theoretic Approach to Picture Processing," *Computer Graphics and Image Processing*, Vol. 7, No. 2, pp. 282-291, Apr. 1978.

[Hueckel71] M.H. Hueckel, "An Operator which Locates Edges in Digitized Pictures," *J. of the ACM*, Vol. 18, No. 1, pp. 113-125, Jan. 1971.

[Hueckel73] M.H. Hueckel, "A Local Visual Operator which Recognizes Edges and Lines," *J. of the ACM*, Vol. 20, No. 4, 634-647, Oct. 1973.

[Hummel78] R.A. Hummel and A. Rosenfeld, "Relaxation Processes for Scene Labeling," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 10, pp. 765-768, Oct. 1978.

[Hunter79] G.M. Hunter and K. Steiglitz, "Operations on Images using Quad Trees," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-1, No. 2, pp. 145-153, Apr. 1979.

[Jain80] A.K. Jain, S.P. Smith, and E. Backer, "Segmentation of Muscle Cell Pictures: A Preliminary Study," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 3, pp. 232-242, May 1980.

[Jarvis75] R.A. Jarvis, "Image Segmentation by Interactively Combining Line, Region, and Semantic Structure," *Proc. Conf. on Computer Graphics, Pattern Recognition, and Data Structures*, Los Angeles, pp. 279-288, May 1975.

[Kanade80] T. Kanade, "Region Segmentation: Signal vs Semantics," *Computer Graphics and Image Processing*, Vol. 13, pp. 279-297, 1980. Reprinted in Y.H. Pao and G.W. Ernst (eds.), *Context-Directed Pattern Recognition and Machine Intelligence Techniques for Information Processing*, pp. 518-533, 1982.

[Kasvand74] T. Kasvand, "Segmentation of Single Gray Level Pictures of General 3D Scenes," *Proc. 2nd Int. Jnt. Conf. on Pattern Recognition*, Copenhagen, pp. 372-373, Aug. 1974.

[Kelly70] M.D. Kelly, *Visual Identification of People by Computer*, Report AIM-130, Computer Science Thesis, Stanford Univ., July 1970.

[Kelly71] M.D. Kelly, "Edge Detection in Pictures by Computer using Planning," in B. Meltzer and D. Michie (eds.), *Machine Intelligence*, Vol. 6, Edinburgh Univ. Press, pp. 397-409, 1971.

[Kender76] J.R. Kender, *Saturation, Hue, and Normalized Color: Calculation, Digitization Effects, and Use*, Dept. of Computer Science, Carnegie-Mellon Univ., Nov. 1976.

[Kender77] J.R. Kender, "Instabilities in Color Transformations," *IEEE Conf. on Pattern Rec. and Image Processing*, Troy, NY, pp. 266-274, June 1977.

[Keng77a] J. Keng, "Image Segmentation and Object Detection by a Syntactic Method," *Proc. Image Understanding Workshop*, Minneapolis, pp. 38-43, Apr. 1977.

[Keng77b] J. Keng, *Syntactic Algorithms for Image Segmentation and a Special Computer Architecture for Image Processing*, Ph.D. Thesis, Purdue Univ., West Lafayette, Indiana, Dec. 1977.

[Kettig76] R.K. Kettig and D.A. Landgrebe, "Classification of Multispectral Image Data by Extraction and Classification of Homogeneous Objects," *IEEE Trans. on Geosci. Electron.*, Vol. GE-14, pp. 19-25, Jan. 1976.

[Kirby79] R. Kirby, "A Product Rule Relaxation Method," *Computer Graphics and Image Processing*, Vol. 10, pp. 235-245, 1979.

[Kirsch71] R.A. Kirsch, "Computer Determination of the Constituent Structure of Biological Images," *Computers and Biomedical Research*, Vol. 4, No. 3, pp. 315-328, June 1971.

[Klein77] G.M. Klein and S.A. Doudani, "Locating Man-Made Objects in Low-Resolution Outdoor Scenes," *Applic. of Digital Image Processing*, SPIE Vol. 119, pp. 276-283, 1977.

[Klinger76] A. Klinger and C.R. Dyer, "Experiments on Picture Representation using Regular Decomposition," *Computer Graphics and Image Processing*, Vol. 5, pp. 68-105, 1976.

[Kohler81] R. Kohler, "A Segmentation System Based on Thresholding," *Computer Graphics and Image Processing*, Vol. 15, pp. 319-338, 1981.

[Laws80] K.I. Laws, *Textured Image Segmentation*, USC Image Processing Inst., Los Angeles, CA, Report USCIPI 940, Jan. 1980.

[Laws83] K.I. Laws, *The GHOUGH Generalized Hough Transform Package: Description and Evaluation*," Technical Note, Artificial Intelligence Center, SRI International, June 1983.

[Lee82] C.H. Lee, "Iterative Region Segmentation," *IEEE Conf. on Pattern Rec. and Image Processing*, Las Vegas, pp. 557-559, June 1982.

[Lemkin79] P. Lemkin, "An Approach to Region Splitting," *Computer Graphics and Image Processing*, Vol. 10, No. 3, pp. 281-288, July 1979.

[Levine76] M.D. Levine and J. Leemet, "A Method for Non-Purposive Picture Segmentation," *Proc. 3rd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 494-498, Nov. 1976.

[Levine77] M.D. Levine, *A Knowledge-Based Computer Vision System*, Report R-77-3, Dept. of Electrical Engineering, McGill Univ., Oct. 1977. See also the same title in *Adv. papers for the Workshop on Computer Vision Systems, Vol. III*, Univ. of Mass., pp. 1-20, June 1977.

[MacVicar-Whelan81] P.J. MacVicar-Whelan and T.O. Binford, "Line Finding with Subpixel Precision," *Proc. Image Understanding Workshop*, Washington, D.C., pp. 26-31, Apr. 1981.

[Martelli76] A. Martelli, "An Application of Heuristic Search Methods to Edge and Contour Detection," *Comm. of the ACM*, Vol. 19, No. 2, pp. 73-83, Feb. 1976.

[McCormick75] B.H. McCormick and S.N. Jayaramamurthy, "A Decision Theory Method for the Analysis of Texture," *Int. J. of Computer and Info. Sciences*, Vol. 4, No. 1, pp. 1-38, Mar. 1975.

[McKeown81] D. McKeown and J. Denlinger, *Grinnell Display Software Support*, IUS Document CMU002, Second Ed., Computer Science Dept., Carnegie-Mellon Univ., June 1981.

[Milgram77] D.L. Milgram, "Region Extraction using Convergent Evidence," *Proc. Image Understanding Workshop*, Minneapolis, pp. 58-64, Apr. 1977. Also Report TR-674, U. of Maryland Computer Science Center, June 1978, and in *Computer Graphics and Image Processing*, Vol. 11, pp. 1-13, 1979.

[Milgram78] D.L. Milgram, "Edge Point Linking using Convergent Evidence," *Proc. Image Understanding Workshop*, pp. 85-91, Nov. 1978.

[Milgram79] D.L. Milgram and D.J. Kahl, "Recursive Region Extraction," *Computer Graphics and Image Processing*, Vol. 9, pp. 82-88, 1979.

[Milgram80] D.L. Milgram and M. Herman, "Clustering Edge Values for Threshold Selection," *Computer Graphics and Image Processing*, Vol. 10, pp. 272-280, 1980.

[Mitchell78] O.R. Mitchell and S.G. Carlton, "Image Segmentation using a Local Extrema Texture Measure," *Pattern Recognition*, Vol. 10, pp. 205-210, 1978.

[Mitchell79] O.R. Mitchell, S.P. Lutton, and S.P. Su, "Texture Image Segmentation using Local Extrema," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Chicago, pp. 508-511, Aug. 1979.

[Mitiche80] A. Mitiche and L. Davis, "Theoretical Analysis of Edge Detection in Textures," *Proc. 5th Int. Int. Conf. on Pattern Recognition*, Miami Beach, FL, pp. 540-547, Dec. 1980.

[Montanari71] U. Montanari, "On the Optimal Detection of Curves in Noisy Pictures," *Comm. of the ACM*, Vol. 14, No. 5, pp. 335-345, May 1971.

[Muerle68] J.L. Muerle and D.C. Allen, "Experimental Evaluation of Techniques for Automatic Segmentation of Objects in a Complex Scene," in G.C. Cheng *et al.* (eds.), *Pictorial Pattern Recognition*, Thompson, Washington, DC, pp. 3-13, 1968.

[Mui78] J.K. Mui, J.W. Bacus, and K.S. Fu, "A Scene Segmentation Technique for Microscopic Cell Images," *Proc. Symp. Computer Aided Diagnosis of Medical Images*, San Diego, CA, IEEE CH1170-0C, pp. 99-106, 1976.

[Nagin77] P.A. Nagin, A.R. Hanson, and E.M. Riseman, *Region Extraction and Description through Planning*, Computer and Information Science Report 77-8, Univ. of Mass. at Amherst, May 1977.

[Nagin78] P.A. Nagin, *Segmentation using Spatial Context and Feature Space Cluster Labels*, Computer and Information Science Report 78-8, Univ. of Mass. at Amherst, May 1978.

[Nagin79] P. Nagin, R. Kohler, A. Hanson, and E. Riseman, "Segmentation, Evaluation, and Natural Scenes," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Chicago, pp. 515-522, Aug. 1979.

[Nahi77] N.E. Nahi and M.H. Jahanshahi, "Image Boundary Estimation," *IEEE Trans. on Computers*, Vol. C-26, No. 8, pp. 772-781, Aug. 1977. Reprinted in H.C. Andrews (ed.), *Tutorial and selected papers in Digital Image Processing*, IEEE Computer Society, pp. 546-555, 1978.

[Nahi78] N.E. Nahi and S. Lopez-Mora, "Estimation-Detection of Object Boundaries in Noisy Images," *IEEE Trans. on Automatic Control*, Vol. AC-23, No. 5, pp. 834-846, Oct. 1978.

[Narendra77] P.M. Narendra and M. Goldberg, "A Graph-Theoretic Approach to Image Segmentation," *Proc.*IEEE Troy, NY, pp. 248-256, June 1977.

[Narendra80] P.M. Narendra and M. Goldberg, "Image Segmentation with Directed Trees," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 2, pp. 185-191, Mar. 1980.

[Nevatia78] R. Nevatia, "Locating Object Boundaries in Textured Environments," *IEEE Trans. on Computers*, Vol. C-25, No. 11, pp. 1170-1175, Nov. 1976.

[Nevatia77a] R. Nevatia and K. Price, "A Comparison of some Segmentation Techniques", *Proc. Image Understanding Workshop*, pp. 55-57, Apr. 1977.

[Nevatia77b] R. Nevatia, "A Color Edge Detector and Its Use in Scene Segmentation," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-7, No. 11, pp. 820-826, Nov. 1977.

[Nevatia82] R. Nevatia, *Machine Perception*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Ohlander75] R. Ohlander, *Analysis of Natural Scenes*, PhD. Thesis, Dept. of Computer Science, Carnegie-Mellon U., Pittsburg, Apr. 1975.

[Ohlander78] R. Ohlander, K. Price, and D.R. Reddy, "Picture Segmentation using a Recursive Region Splitting Method," *Computer Graphics and Image Processing*, Vol. 8, No. 3, pp. 313-333, Dec. 1978.

[Ohta80a] Y. Ohta, T. Kanade, and T. Sakai, "Color Information for Region Segmentation," *Computer Graphics and Image Processing*, Vol. 13, pp. 222-241, 1980.

[Ohta80b] Y. Ohta, *A Region-Oriented Image-Analysis System by Computer*, Ph.D. Thesis, Dept. of Info. Sciences, Kyoto Univ., Japan, Mar. 1980.

[O'Rourke81] J. O'Rourke, "Dynamically Quantized Spaces for Focusing the Hough Transform," *Proc. 7th Int. Jnt. Conf. on Artificial Intelligence*, Vol. 2, pp. 737-739, 1981.

[Panda78] D.P. Panda and A. Rosenfeld, "Image Segmentation by Pixel Classification in (Grey Level, Edge Value) Space," *IEEE Trans. on Computers*, Vol. C-27, No. 9, pp. 875-879, Sep. 1978.

[Pavlidis72] T. Pavlidis, "Segmentation of Pictures and Maps through Functional Approximation," *Computer Graphics and Image Processing*, Vol. 1, pp. 360-372, 1972.

[Pavlidis75] T. Pavlidis and S. Tanimoto, "Texture Identification by a Directed Split-and-Merge Procedure," *Proc. Conf. on Computer Graphics, Pattern Recognition, and Data Structures*, Los Angeles, pp. 201-203, May 1975.

[Pavlidis78] T. Pavlidis, "A Review of Algorithms for Shape Analysis," *Computer Graphics and Image Processing*, Vol. 7, pp. 243-258, 1978.

[Peleg78] S. Peleg, "Iterative Histogram Modification, 2", *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 7, pp. 555-556, July 1978.

[Peleg80] S. Peleg, "A New Probabilistic Relaxation Scheme," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 4, pp. 362-369, July 1980.

[Perkins80] W.A. Perkins, "Area Segmentation of Images Using Edge Points," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 1, pp. 8-15, Jan. 1980.

[Pietikäinen82] M. Pietikäinen, A. Rosenfeld, and I. Walter, "Split-and-Link Algorithms for Image Segmentation," *Pattern Recognition*, Vol. 15, No. 4, pp. 287-298, 1982.

[Pingle 71] K. Pingle and J. Tenenbaum, "An Accommodating Edge Follower," *Proc. 2nd Int. Jnt. Conf. on Artificial Intelligence*, London, pp. 1-7, Sep. 1971.

[Postaire81] J.G. Postaire and C.P.A. Vasseur, "An Approximate Solution to Normal Mixture Identification with Application to Unsupervised Pattern Classification," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-3, No. 2, pp. 163-179, Mar. 1981.

[Prager80] J.M. Prager, "Extracting and Labeling Boundary Segments in Natural Scenes," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 1, pp. 16-27, Jan. 1980.

[Pratt78] W.K. Pratt, *Digital Image Processing*. New York: Wiley-Interscience, 1978.

[Preparata72] F.P. Preparata and S.R. Ray, "An Approach to Artificial Non-Symbolic Cognition," *Inf. Sci.*, Vol. 4, pp. 65-86, 1972.

[Prewitt66] J.M.S. Prewitt and M.L. Mendhelson, "The Analysis of Cell Images," *Ann. N.Y. Acad. of Sci.*, Vol. 128, pp. 1035-1053, 1966.

[Prewitt70] J.M.S. Prewitt, "Object Enhancement and Extraction." In B. Lipkin and A. Rosenfeld (eds.), *Picture Processing and Psychopictorics*, Academic Press, New York, pp. 75-149, 1970.

[Price78] K.E. Price, *Change Detection and Analysis in Multi-Spectral Images*, Ph.D. Thesis, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA, Dec. 1976.

[Price78a] K.E. Price, "Matching Segments of Images," *Semiannual Technical Report*, Report 800, Image Processing Inst., Univ. of Southern California, pp. 2-22, Mar. 1978.

[Price78b] K.E. Price, "Symbolic Matching and Analysis with Substantial Changes in Orientation," *Semiannual Technical Report*, Report 800, Image Processing Inst., Univ. of Southern California, pp. 22-41, Mar. 1978.

[Price79] K.E. Price, "Segmentation," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Chicago, pp. 512-514, Aug. 1979.

[Price81] K.E. Price, "Relaxation Matching applied to Aerial Images," *Proc. Image Understanding Workshop*, pp. 22-25, Apr. 1981.

[Ranade80] S. Ranade and J.M.S. Prewitt, "A Comparison of Some Segmentation Algorithms for Cytology," *Proc. 5th Int. Jnt. Conf. on Pattern Recognition*, Miami Beach, FL, pp. 561-564, Dec. 1980.

[Ratkovic79a] J.A. Ratkovic, *Performance Considerations for Image Matching Systems*, The Rand Corp., Santa Monica, CA, Report RAND/N-1217-AF, December 1979, 79 pp.

[Ratkovic79b] J.A. Ratkovic, *Structuring the Components of the Image Matching Problem*, The Rand Corp., Santa Monica, CA, Report RAND/N-1216-AF, December 1979, 35 pp.

[Ratkovic79c] J.A. Ratkovic, "Hybrid Correlation Algorithms – A Bridge between Feature Matching and Image Correlation," *Proc. 18th IEEE Conf. on Decision and Control*, Fort Lauderdale, FL, Vol. 2, pp. 1046-52, December 1979.

[Read72] J.S. Read and S.N. Jayaramamurthy, "Automatic Generation of Texture Feature Detectors," *IEEE Trans. on Computers*, Vol. C-21, No. 7, pp. 803-812, July 1972.

[Riseman77] E.M. Riseman and M.A. Arbib, "Computational Techniques in the Visual Segmentation of Static Scenes," *Computer Graphics and Image Processing*, Vol. 6, No. 3, pp. 221-276, June 1977.

[Robertson73] T.V. Robertson, P.H. Swain, and K.S. Fu, *Multispectral Image Partitioning*, TR-EE 73-26 (LARS Information Note 071373), School of Electrical Engineering, Purdue Univ., Aug. 1973.

[Robinson77] G.S. Robinson, "Color Edge Detection," *Optical Engineering*, Sep./Oct. 1977. Also *Proc. SPIE Symp. on Advances in Image Transmission Techniques*, San Diego, Vol. 87, Aug. 1976.

[Rosenfeld76a] A. Rosenfeld and A.C. Kak, *Digital Image Processing*, Academic Press, NY, 1976.

[Rosenfeld76b] A. Rosenfeld, R.A. Hummel, and S.W. Zucker, "Scene Labeling by Relaxation Operations," *IEEE Systems, Man, and Cybernetics*, Vol. SMC-6, No. 6, pp. 420-433, June 1976.

[Rosenfeld78] A. Rosenfeld and L.S. Davis, "Iterative Histogram Modification," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 8, pp. 300-302, 1978.

[Rosenfeld79] A. Rosenfeld and L.S. Davis, "Image Segmentation and Image Models," *Proc. IEEE*, Vol. 67, No. 5, pp. 764-772, May 1979.

[Rosenfeld80] A. Rosenfeld, C.Y. Wang, and A.Y. Wu, *Multispectral Texture*, Report TR-988 (and AFOSR-77-3271), Computer Science Ctr., Univ. of Maryland, College Park, Dec. 1980.

[Rubin80] S.M. Rubin, "Natural Scene Recognition Using LOCUS Search," *Computer Graphics and Image Processing*, Vol. 13, pp. 298-333, 1980.

[Rutkowski81] W.S. Rutkowski, S. Peleg, and A. Rosenfeld, "Shape Segmentation Using Relaxation," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-3, No. 4, pp. 368-375, July 1981.

[Sakai76] T. Sakai, T. Kanade, and Y. Ohta, "Model Based Interpretation of Outdoor Scenes," *Proc. 3rd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 581-585, Nov. 1976.

[Samet79] H. Samet, "Quadtree Structures for Region Processing," *Proc. Image Understanding Workshop*, Los Angeles, pp. 36-41, Nov. 1979.

[Sarabi81] A. Sarabi and J.K. Aggarwal, "Segmentation of Chromatic Images," *Pattern Recognition*, Vol. 13, No. 6, pp. 417-427, 1981.

[Schachter75] B.J. Schachter, L.S. Davis, and A. Rosenfeld, *Scene Segmentation by Cluster Detection in Color Spaces*, Report TR-424, Univ. of Maryland Computer Science Dept., 22 pp., Nov. 1975.

[Schachter77] B.J. Schachter, L.S. Davis, and A. Rosenfeld, *Some Experiments in Image Segmentation by Clustering of Local Feature Values*, Report TR-510, Univ. of Maryland Computer Science Dept., Mar. 1977.

[Schachter79] B.J. Schachter, L.S. Davis, and A. Rosenfeld, "Some Experiments in Image Segmentation by Clustering of Local Feature Values," *Pattern Recognition*, Vol. 11, No. 1, pp. 19-28, 1979.

[Shafer80] S.A. Shafer, *MOOSE: User's Manual, Implementation Guide, Evaluation*, IFI-HH B-70/80, Fachbereich Informatik, Univ. of Hamburg, W. Germany, Apr. 1980.

[Shafer82] S. Shafer and T. Kanade, *Recursive Region Segmentation by Analysis of Histograms*, Proc. Int. Conf. on Acoustics, Speech, and Signal Processing, Paris, pp. 1166-1171, May 1982.

[Serreyn78] D. Serreyn and R. Larson, "Adaptive Threshold for an Image Recognition System - Background Results and Conclusions," *Proc. Image Understanding Workshop*, pp. 133-138, May 1978.

[Smith80] D.R. Smith, *CMU Image Format and Paging System*, IUS Document CMU003, Computer Science Dept., Carnegie-Mellon Univ., Nov. 1980.

[Sloan75] K.R. Sloan and R. Bajcsy, "A Computational Structure for Color Perception," *Proc. ACM '75*, Minneapolis, 1975.

[Sloan81] K.R. Sloan, Jr., "Dynamically Quantized Pyramids," *Proc. 7th Int. Jnt. Conf. on Artificial Intelligence*, Vol. 2, pp. 734-736, 1981.

[Somerville78] C. Somerville and J.L. Mundy, "One Pass Contouring of Images Through Planar Approximation," *Proc. 3rd Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 745-748, Nov. 1976.

[Tanimoto75] S.L. Tanimoto and T. Pavlidis, "A Hierarchical Data Structure for Picture Processing," *Computer Graphics and Image Processing*, Vol. 4, pp. 104-113, 1975.

[Tanimoto77] S.L. Tanimoto and T. Pavlidis, "The Editing of Picture Segmentations using Local Analysis of Graphs," *Comm. of the ACM*, Vol. 20, No. 4, pp. 223-229, Apr. 1977.

[Tanimoto78] S.L. Tanimoto, "An Optimal Algorithm for Computing Fourier Texture Descriptors," *IEEE Trans. on Computers*, Vol. C-27, pp. 81-84, January 1978.

[Tenenbaum73] J.M. Tenenbaum, *On Locating Objects by Their Distinguishing Features in Multisensory* Images," TN 84, Artificial Intelligence Center, SRI International, Sep. 1973.

[Tenenbaum74] J.M. Tenenbaum, T.D. Garvey, S. Weyl, and H.C. Wolf, *An Interactive Facility for Scene Analysis Research*, Technical Note 87, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, CA, Jan. 1974.

[Tenenbaum75] J.M. Tenenbaum and S. Weyl, "A Region-Analysis Subsystem for Interactive Scene Analysis," *Proc. 4th Int. Conf. on Artificial Intelligence*, Tbilisi, USSR, pp. 682-687, Sep. 1975.

[Tenenbaum76a] J.M. Tenenbaum and H.G. Barrow, "Experiments in Interpretation-Guided Segmentation," *Jnt. Conf. on Pattern Recognition and Artificial Intelligence*, June 1976. Also in *Artificial Intelligence*, Vol. 8, pp. 241-274, 1976.

[Tenenbaum76b] J.M. Tenenbaum and H.G. Barrow, "IGS: A Paradigm for Integrating Image Segmentation and Interpretation," *Proc. 3rd Int. Jnt. Conf. on Pattern Recognition*, Coronado, CA, pp. 504-513, Nov. 1976.

[Tenenbaum80] J.M. Tenenbaum, *Application of Image Understanding to Cartography: A Survey* of Relevant Research and Opportunities," Technical Report, Artificial Intelligence Center, SRI International, Dec. 1980.

[Thompson77] W.B. Thompson, "Textural Boundary Analysis," *IEEE Trans. on Computers*, pp. 272-275, Mar. 1977.

[Thompson80] W.B. Thompson, "Combining Motion and Contrast for Segmentation," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-2, No. 6, pp. 543-549, Nov. 1980.

[Tisdale79] G.E. Tisdale, A.R. Helland, and J. Shipley, *Automatic Target Cueing (AUTO-Q) System Engineering Model Design*, Westinghouse Systems Development Division, Baltimore, MD, Report 80-0488, Sep. 1979.

[Tomita73] F. Tomita, M. Yachida, and S. Tsuji, "Detection of Homogeneous Regions by Structural Analysis," *Proc. 3rd Int. Jnt. Conf. on Artificial Intelligence*, pp. 564-571, Aug. 1973.

[Tomita82] F. Tomita, Y. Shirai, and S. Tsuji, "Description of Textures by a Structural Analysis," *IEEE Trans. on Pattern Anal. and Machine Intelligence*, Vol. PAMI-4, No. 2, pp. 183-191, Mar. 1982.

[Troy73] E.B. Troy, E.S. Deutsch, and A. Rosenfeld, "Gray-Level Manipulation Experiments for Texture Analysis," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-3, No. 1, pp. 91-98, Jan. 1973.

[Tsuji73] S. Tsuji and F. Tomita, "A Structural Analyzer for a Class of Textures," *Computer Graphics and Image Processing*, Vol. 2, No. 3/4, pp. 216-231, Dec. 1973.

[Uhr72] L. Uhr, "Layered Recognition Cone Networks that Preprocess, Classify and Describe," *IEEE Trans. on Computers*, Vol. 21, pp. 758-768, 1972.

[Underwood77] S.A. Underwood and J.K. Aggarwal, "Interactive Computer Analysis of Aerial Color Infrared Photographs," *Computer Graphics and Image Processing*, Vol. 6, pp. 1-24, 1977.

[Wacker69] A.G. Wacker, *A Cluster Approach to Finding Spatial Boundaries in Multispectral Imagery*, LARS Information Note 122969, Purdue Univ., West Lafayette, Indiana, 1969.

[Weszka74] J.S. Weszka, R.N. Nagel, and A. Rosenfeld, "A Threshold Selection Technique," *IEEE Trans. on Computers*, Vol. C-23, No. 12, pp. 1322-1326, Dec. 1974.

[Weszka78] J.S. Weszka and A. Rosenfeld, "Threshold Evaluation Techniques," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 8, pp. 622-629, Aug. 1978.

[Winkler78] G. Winkler and K. Vattrodt, "Measures for Conspicuousness," *Computer Graphics and Image Processing*, Vol. 8, pp. 355-368, 1978.

[Wolfe70] J.H. Wolfe, "Pattern Clustering by Multivariate Mixture Analysis," *Behavioral Research*, Vol. 5, pp. 329-350, 1970.

[Yakimovsky73a] Y. Yakimovsky, *Scene Analysis using a Semantic Base for Region Growing*, Report STAN-CS-73-380, Stanford U. Computer Science Dept., June 1973.

[Yakimovsky73b] Y. Yakimovsky and J.A. Feldman, "A Semantic Based Decision Theory Region Analyzer," *Proc. 3rd Int. Jnt. Conf. on Artificial Intelligence*, pp. 580-588, Aug. 1973.

[Yakimovsky76] Y. Yakimovsky, "Boundary and Object Detection in Real World Images," *J. of the ACM*, Vol. 23, No. 4, pp. 599-618, Oct. 1976.

[Yan77] J.K. Yan and D.J. Sakrison, "Encoding of Images Based on a Two-Component Source Model," *IEEE Trans. on Communications*, Vol. COM-25, No. 11, pp. 1315-1322, Nov. 1977.

[Yasnoff77] W.A. Yasnoff, J.K. Mui, and J.W. Bacus, "Error Measures for Scene Segmentations," *Pattern Recognition*, Vol. 9, pp. 217-231, 1977.

[Yoo78] J.R. Yoo and T.S. Huang, *Image Segmentation by Unsupervised Clustering and its Applications*, Report TR-EE-78-19, Purdue Univ., West Lafayette, Indiana, 1978.

[Zucker75] S.W. Zucker, A. Rosenfeld, and L.S. Davis, "Picture Segmentation by Texture Discrimination," *IEEE Trans. on Computers*, Vol. C-24, No. 12, pp. 1228-1233, Dec. 1975.

[Zucker76a] S.W. Zucker, "Region Growing: Childhood and Adolescence," *Computer Graphics and Image Processing*, Vol. 5, No. 3, pp. 382-399, Sep. 1976.

[Zucker76b] S.W. Zucker, "Relaxation Labelling and the Reduction of Local Ambiguities," *Proc. 3rd Int. Int. Conf. on Pattern Recognition*, Coronado, CA, pp. 852-861, Nov. 1976.

[Zucker77] S.W. Zucker, R.A. Hummel, and A. Rosenfeld, "Application of Relaxation Labeling to Line and Curve Enhancement," *IEEE Trans. on Computers*, Vol. C-26, No. 4, pp. 394-403, Apr. 1977. Correction in Vol. C-26, No. 9, pp. 922-929, Sep. 1977.

[Zucker78] S. W. Zucker and J.L. Mohammed, "Analysis of Probabilistic Relaxation Labelling Processes," *Proc. IEEE Conf. on Pattern Rec. and Image Processing*, Chicago, pp. 307-312, 1978.