Chapter 10

Image Segmentation

The term *image segmentation* refers to the partition of an image into a set of regions that cover it. The goal in many tasks is for the regions to represent meaningful areas of the image, such as the crops, urban areas, and forests of a satellite image. In other analysis tasks, the regions might be sets of border pixels grouped into such structures as line segments and circular arc segments in images of 3D industrial objects. Regions may also be defined as groups of pixels having both a border and a particular shape such as a circle or ellipse or polygon. When the interesting regions do not cover the whole image, we can still talk about segmentation, into foreground regions of interest and background regions to be ignored.

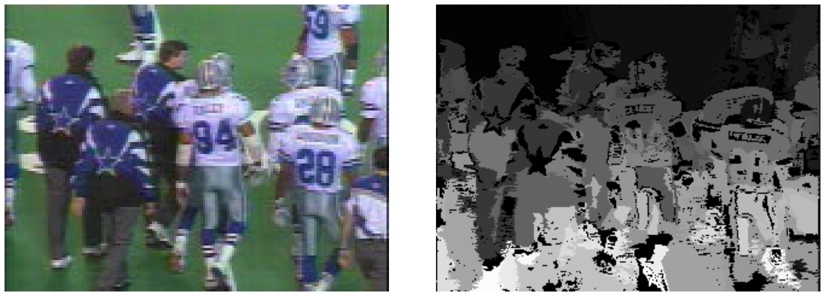


Figure 10.1: Football image (left) and segmentation into regions (right). Each region is a set of connected pixels that are similar in color.

Segmentation has two objectives. The first objective is to decompose the image into parts for further analysis. In simple cases, the environment might be well enough controlled so that the segmentation process reliably extracts only the parts that need to be analyzed further. For example, in the chapter on color, an algorithm was presented for segmenting a human face from a color video image. The segmentation is reliable, provided that the person's clothing or room background does not have the same color components as a human face. In complex cases, such as extracting a complete road network from a greyscale aerial image, the segmentation problem can be very difficult and might require application of a

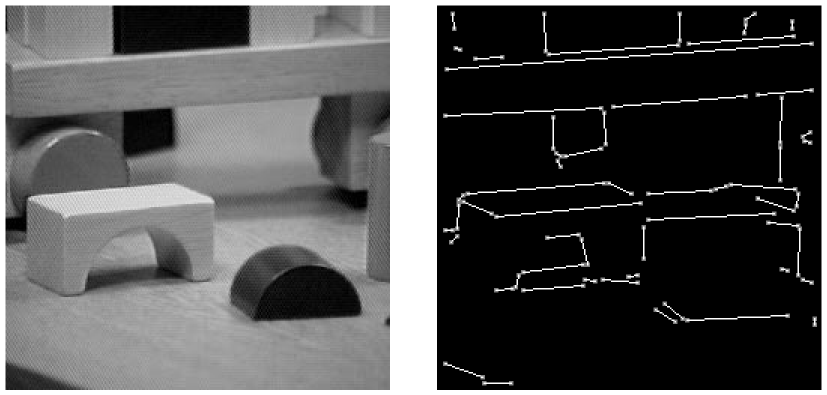


Figure 10.2: Blocks image (left) and extracted set of straight line segments (right). The line segments were extracted by the ORT (Object Recognition Toolkit) package.

great deal of domaina building knowledge.

The second objective of segmentation is to perform a change of representation. The pixels of the image must be organized into higher-level units that are either more meaningful or more efficient for further analysis (or both). A critical issue is whether or not segmentation can be performed for many different domains using general bottom-up methods that do not use any special domain knowledge. This chapter presents segmentation methods that have potential use in many different domains. Both region-based and curve-based units are discussed in the following sections. The prospects of having a single segmentation system work well for all problems appear to be dim. Experience has shown that an implementor of machine vision applications must be able to choose from a toolset of methods and perhaps tailor a solution using knowledge of the application.

This chapter discusses several different kinds of segmentation algorithms including the classical region growers, clustering algorithms, and line and circular arc detectors. Figure 10.1 illustrates the segmentation of a colored image of a football game into regions of near-constant color. Figure 10.2 shows the line segments extracted from an image of toy blocks. In both cases, note that the results are far from perfect by human standards. However, these segmentations might provide useful input for higher-level automated processing, for example, identifying players by number or recognizing a part to be assembled.

10.1 Identifying Regions

- Regions of an image segmentation should be uniform and homogeneous with respect to some characteristic, such as gray level, color, or texture
- Region interiors should be simple and without many small holes.
- Adjacent regions of a segmentation should have significantly different values with respect to the characteristic on which they are uniform.

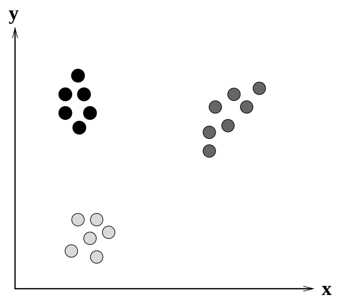


Figure 10.3: Set of points in a Euclidean measurement space that can be separated into three clusters of points. Each cluster consists of points that are in some sense close to one another. Clusters are designated by the fill patterns inside the circles.

- Boundaries of each segment should be smooth, not ragged, and should be spatially accurate.

Achieving all these desired properties is difficult because strictly uniform and homogeneous regions are typically full of small holes and have ragged boundaries. Insisting that adjacent regions have large differences in values can cause regions to merge and boundaries to be lost. In addition, the regions that humans see as homogeneous may not be homogeneous in terms of the low-level features available to the segmentation system, so higher-level knowledge may have to be used. The goal of this chapter is to develop algorithms that will apply to a variety of images and serve a variety of higher-level analyses.

10.1.1 Clustering Methods

Clustering in pattern recognition is the process of partitioning a set of pattern vectors into subsets called *clusters*. For example, if the pattern vectors are pairs of real numbers as illustrated by the point plot of Figure 10.3, clustering consists of finding subsets of points that are "close" to each other in Euclidean two-space.

The general term *clustering* refers to a number of different methods. We will look at several different types of clustering algorithms that have been found useful in image segmentation. These include classical clustering algorithms, simple histogram-based methods, Ohlander's recursive histogram-based technique, and Shi's graph-partitioning technique.

Classical Clustering Algorithms

The general problem in clustering is to partition a set of vectors into groups having similar values. In image analysis, the vectors represent pixels or sometimes small neighborhoods around pixels. The components of these vectors can include:

1. intensity values
2. RGB values and color properties derived from them

3. calculated properties
4. texture measurements

Any feature that can be associated with a pixel can be used to group pixels. Once pixels have been grouped into clusters based on these *measurement-space* values, it is easy to find connected regions using connected components labeling.

In traditional clustering, there are K clusters $C_1, C_2, \ldots, C_K$ with means $m_1, m_2, \ldots, m_K$. A *least squares error measure* can be defined as

$$D = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - m_k\|^2.$$

which measures how close the data are to their assigned clusters. A least-squares clustering procedure could consider *all* possible partitions into K clusters and select the one that minimizes D. Since this is computationally infeasible, the popular methods are approximations. One important issue is whether or not K is known in advance. Many algorithms expect K as a parameter from the user. Others attempt to find the best K according to some criterion, such as keeping the variance of each cluster less than a specified value.

Iterative K-Means Clustering The K-means algorithm is a simple, iterative hill-climbing method. It can be expressed as follows.

```
Form K-means clusters from a set of n-dimensional vectors.

  1. Set $ic$ (iteration count) to 1.
  2. Choose randomly a set of $K$ means $m_1(1), m_2(1), \ldots, m_K(1)$.
  3. For each vector $x_i$ compute $D(x_i, m_k(ic))$ for each $k = 1, \ldots, K$ and assign $x_i$ to the
     cluster $C_j$ with the nearest mean.
  4. Increment $ic$ by 1 and update the means to get a new set $m_1(ic), m_2(ic), \ldots, m_K(ic)$.
  5. Repeat steps 3 and 4 until $C_k(ic) = C_k(ic+1)$ for all $k$.
```

Algorithm 1: K-Means Clustering

This algorithm is guaranteed to terminate, but it may not find the global optimum in the least squares sense. Step 2 may be modified to partition the set of vectors into K random clusters and then compute their means. Step 5 may be modified to stop after the percentage of vectors that change clusters in a given iteration is small. Figure 10.4 illustrates the application of the K-means clustering algorithm in RGB space to the original football image of Figure 10.1.

Isodata Clustering *Isodata clustering* is another iterative algorithm that uses a split-and-merge technique. Again assume that there are K clusters $C_1, C_2, \ldots, C_K$ with means

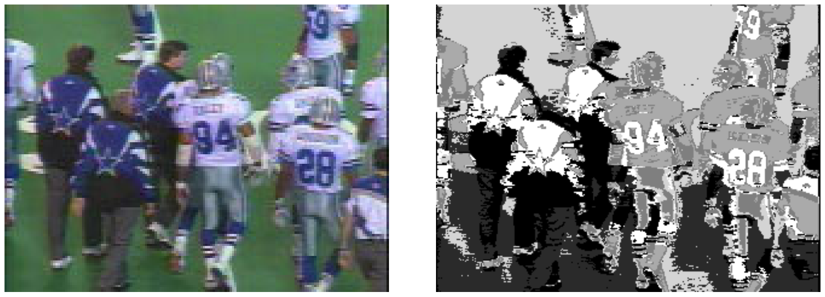


Figure 10.4: Football image (left) and K=6 clusters resulting from a K-means clustering procedure (right) shown as distinct gray tones. The six clusters correspond to the six main colors in the original image: dark green, medium green, dark blue, white, silver, and black.

$m_1, m_2, \ldots, m_K$, and let Σ_k be the covariance matrix of cluster k (as defined below). If the x_i's are vectors of the form

$$x_i = [v_1, v_2, \ldots, v_n]$$

then each mean m_k is a vector

$$m_k = [m_{1k}, m_{2k}, \ldots, m_{nk}]$$

and Σ_k is defined by

$$\Sigma_k = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \ldots & \sigma_{1n} \\ \sigma_{12} & \sigma_{22} & \ldots & \sigma_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{1n} & \sigma_{2n} & \ldots & \sigma_{nn} \end{bmatrix} \tag{10.1}$$

where $\sigma_{ii} = \sigma_i^2$ is the variance of the ith component v_i of the vectors and $\sigma_{ij} = \rho_{ij}\sigma_i\sigma_j$ is the covariance between the ith and jth components of the vectors. (ρ_{ij} is the correlation coefficient between the ith and jth components, σ_i is the standard deviation of the ith component, and σ_j is the standard deviation of the jth component.)

Figure 10.5 illustrates the application of the isodata clustering algorithm in RGB space to the original football image of Figure 10.1. This cluster image was the input to a connected components procedure which produced the segmentation shown in Figure 10.1. The threshold τ_v for the isodata clustering was set to a size of 10% of the side of the RGB color-space cube.

Simple Histogram-based Methods

Iterative partition rearrangement schemes have to go through the image data set many times. Because they require only one pass through the data, histogram methods probably involve the least computation time of the measurement–space clustering techniques.

```
Form isodata clusters from a set of n-dimensional vectors.

  1. assign $x_i$ to the cluster $l$ that minimizes
         $D_\Sigma = [x_i - m_l]' \Sigma_l^{-1} [x_i - m_l].$
  2. Merge clusters $i$ and $j$ if
         $| m_i - m_j | < \tau_v$
     where $\tau_v$ is a variance threshold.
  3. Split cluster $k$ if the maximum eigenvalue of $\sigma_k$ is larger than $\tau_v$.
  4. Stop when
         $| m_i(t) - m_i(t+1) | < \epsilon$
     for every cluster $i$ or when the maximum number of iterations has been reached.
```

Algorithm 2: Isodata Clustering

Exercise 1 Isodata vs. K-means clustering

The isodata algorithm gave better results than the K-means algorithm on the football images in that it correctly grouped the dark green areas at the top of the image with those near the bottom. Why do you think the isodata algorithm was able to perform better than the K-means algorithm?

Histogram mode seeking is a measurement–space clustering process in which it is assumed that homogeneous objects in the image manifest themselves as the clusters in measurement–space, i.e. on the histogram. Image segmentation is accomplished by mapping the clusters back to the image domain where the maximal connected components of the cluster labels constitute the image segments. For gray-tone images, the measurement–space clustering can be accomplished by determining the valleys in the histogram and declaring the clusters to be the interval of values between valleys. A pixel whose value is in the ith interval is labeled with index i and the segment to which it belongs is one of the connected components of all pixels whose label is i. The automatic thresholding technique discussed in Chapter 3 is an example of histogram mode seeking with a bimodal histogram.

Exercise 2 Histogram mode seeking

Write a program that finds the modes of a multimodal histogram by first breaking it into two parts, as does the Otsu method in Chapter 3, and then recursively trying to break each part into two more parts, if possible. Test it on gray-tone and color images.

In general, a gray-tone image will have a multimodal histogram, so that any automatic thresholding technique will have to look for significant peaks in the image and the valleys that separate them. This task is easier said than done. Figure 10.6 shows the histogram of the gray-tone blocks image. A naive valley-seeking algorithm might judge it to be bimodal and place the single threshold somewhere between 39 and 79. Trial-and-error threshold

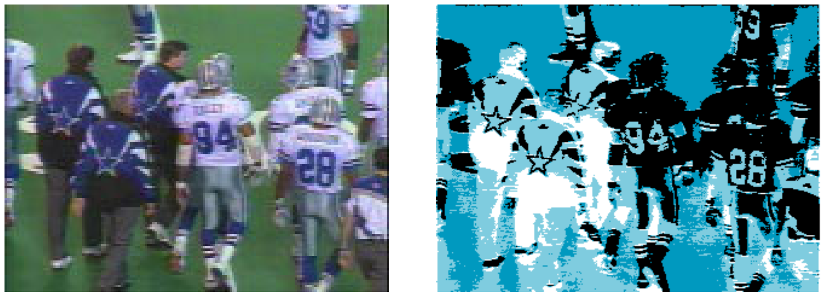


Figure 10.5: Football image (left) and K=5 clusters resulting from an isodata clustering procedure (right) shown as distinct gray tones. The five clusters correspond to five color groups: green, dark blue, white, silver, and black.

selection, however, produced three thresholds yielding the four thresholded images of Figure 10.7, which show some meaningful regions of the image. This motivates the need for *knowledge-directed thresholding* techiques where the thresholds chosen depend on both the histogram and the quality/usefulness of the resulting regions.

Ohlander's Recursive Histogram-Based Technique

Ohlander *et al* (1978) refine the histogram-based clustering idea in a recursive way. The idea is to perform histogram mode seeking first on the whole image and then on each of the regions obtained from the resultant clusters, until regions are obtained that can be decomposed no further. They begin by defining a mask selecting all pixels in the image. Given any mask, a histogram of the masked portion of the image is computed. Measurement–space clustering is applied to this histogram, producing a set of clusters. Pixels in the image are then identified with the cluster to which they belong. If there is only one measurement–space cluster, then the mask is terminated. If there is more than one cluster, then the connected components operator is applied to each cluster, producing a set of connected regions for each cluster label. Each connected component is then used to generate a new mask which is placed on a mask stack. The masks on the stack represent regions that are candidates for further segmentation. During successive iterations, the next mask in the stack selects pixels in the histogram computation process. Clustering is repeated for each new mask until the stack is empty. Figure 10.8 illustrates this process which we call recursive histogram–directed spatial clustering.

For ordinary color images, Ohta, Kanade, and Sakai (1980) suggested that histograms not be computed individually on the red, green, and blue (RGB) color variables, but on a set of variables closer to what the Karhunen–Loeve (principal components) transform would suggest: $(R+G+B)/3$, $(R-B)/2$, and $(2G-R-B)/4$.

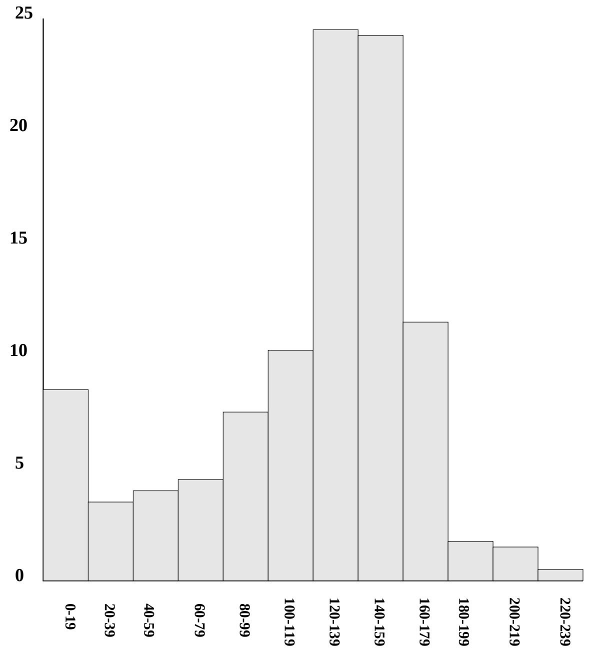


Figure 10.6: Histogram of the blocks image of Figure 10.2.

*Shi's Graph-Partitioning Technique

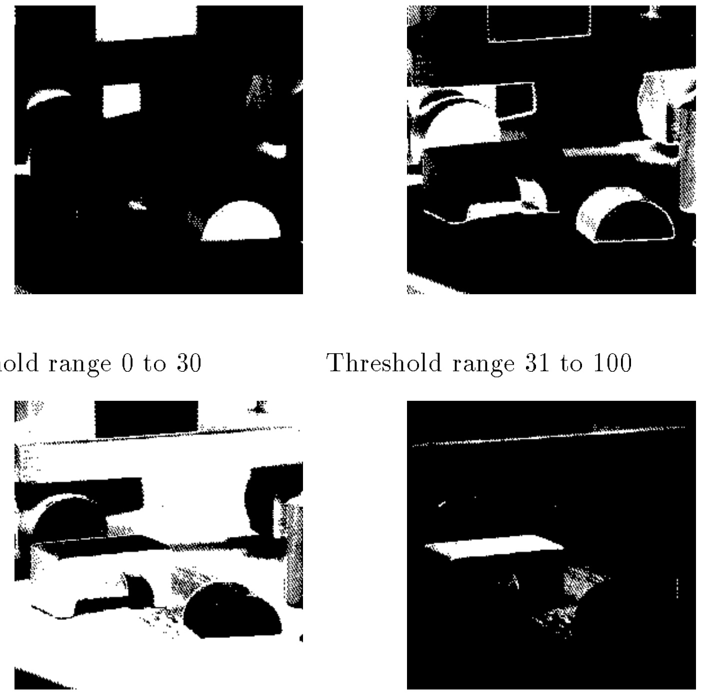
The Ohlander/Ohta algorithms work well on reasonably simple color scenes with man-made objects and single-color regions, but do not extend well to complex images of natural scenes, where they give lots of tiny regions in textured areas. Shi developed a method that can use color, texture, or any combination of these and other attributes. He formulated the segmentation problem as a graph partitioning problem and developed a new graph-partitioning method that reduced to solving an eigenvector/eigenvalue problem as follows.

Let $G = (V, E)$ be a graph whose nodes are points in measurement space and whose edges each have a weight $w(i, j)$ representing the similarity between nodes i and j. The goal in segmentation is to find a partition of the vertices into disjoint sets $V_1, V_2, \ldots, V_m$ so that the similarity within the sets is high and across different sets is low.

A graph $G = (V, E)$ can be partitioned into two disjoint graphs with node sets A and B by removing any edges that connect nodes in A with nodes in B. The degree of dissimilarity between the two sets A and B can be computed as the sum of the weights of the edges that have been removed; this total weight is called a *cut*.

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v) \tag{10.2}$$

One way of formulating the segmentation problem is to look for the *minimum cut* in



Threshold range 0 to 30 Threshold range 31 to 100

Threshold range 101 to 179 Threshold range 180 to 239

Figure 10.7: Four images resulting from three thresholds hand-chosen from the histogram of the blocks image.

the graph, and to do so recursively until the regions are uniform enough. The minimum cut criterion, however, favors cutting small sets of isolated nodes, which is not useful in finding large uniform color/texture regions. Shi proposed the *normalized cut* (Ncut) defined in terms of $cut(A,B)$ and the *association* of A and the full vertex set V defined by:

$$asso(A,V) = \sum_{u \in A, t \in V} w(u,t) \tag{10.3}$$

The definition of normalized cut is then

$$Ncut(A,B) = \frac{cut(A,B)}{asso(A,V)} + \frac{cut(A,B)}{asso(B,V)} \tag{10.4}$$

With this definition, the cut that partitions out small isolated point sets will not have small $Ncut$ values, and the partitions that do produce small $Ncut$ values are more likely to be useful in image segmentation. Furthermore, the related measure for total normalized *association* given by

$$Nasso(A,B) = \frac{asso(A,A)}{asso(A,V)} + \frac{asso(B,B)}{asso(B,V)} \tag{10.5}$$

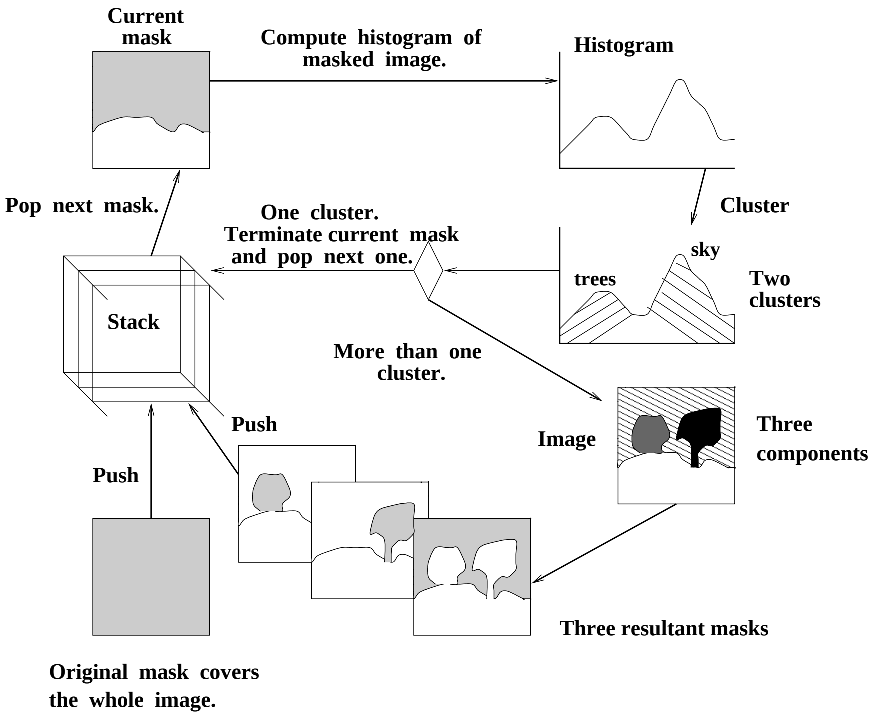


Figure 10.8: Recursive histogram-directed spatial-clustering scheme. The original image has four regions: grass, sky, and two trees. The current mask (shown at upper left) identifies the region containing the sky and the trees. Clustering its histogram leads to two clusters in color space, one for the sky and one for the trees. The sky cluster yields one connected component, while the tree cluster yields two. Each of the three connected components become masks that are pushed onto the mask stack for possible further segmentation.

measures how tightly the nodes within a given set are connected to one another. It is related to the $Ncut$ by the relationship:

$$Ncut(A, B) = 2 - Nasso(A, B) \tag{10.6}$$

so that either of them can be used, as convenient, by a partitioning procedure.

Given the definitions for $Ncut$ and $Nasso$, we need a procedure that segments an image by partitioning the pixel set. Shi's segmentation procedure is as follows.

Shi ran the above algorithm to segment images based on brightness, color, or texture information. The edge weights $w(i, j)$ were defined by

$$w(i,j) = e^{\frac{-\|F(i)-F(j)\|_2}{\sigma_I}} * \begin{cases} e^{\frac{-\|X(i)-X(j)\|_2}{\sigma_X}} & \text{if } \|X(i) - X(j)\|_2 < r \\ 0 & \text{otherwise} \end{cases} \tag{10.11}$$

where

- $X(i)$ is the spatial location of node i.
- $F(i)$ is the feature vector based on intensity, color, or texture information and is defined by
  - $F(i) = I(i)$, the intensity value, for segmenting intensity images.
  - $F(i) = [v, v \cdot s \cdot sin(h), v \cdot s \cdot cos(h)](i)$, where h, s, and v are the HSV values, for color segmentation.
  - $F(i) = [|I * f_1|, \ldots, |I * f_n|](i)$, where the f_i are DOOG (difference of difference of Gaussian) filters at various scales and orientations, for texture segmentation.

Note that the weight $w(i, j)$ is set to 0 for any pair of nodes i and j that are more than a prespecified number r of pixels apart.

The above algorithm leads to very good segmentations of images via color and texture. Figure 10.9 illustrates the performance of the algorithm on some sample images of natural scenes. While the segmentations are very good, the complexity of the algorithm makes it unsuitable for use in a real-time system.

10.1.2 Region Growing

Instead of partitioning the image, a *region grower* begins at one position in the image (often the top left corner) and attempts to grow each region until the pixels being compared are too dissimilar to the region to add them. Usually a statistical test is performed to decide if this is the case. Haralick proposed the following region-growing technique, which assumes that a region is a set of connected pixels with the same population mean and variance.

Let R be a region of N pixels neighboring a pixel with gray tone intensity y. Define the region mean $\overline{X}$ and scatter S^2 by

$$\overline{X} = \frac{1}{N} \sum_{[r,c] \in R} I[r, c] \tag{10.12}$$

Perform a graph-theoretic clustering on a graph whose nodes are pixels and whose edges represent the similarities between pairs of pixels.

1. Set up a weighted graph $G = (V, E)$ whose nodeset V is the set of pixels of the image and whose edgeset E is a set of weighted edges with $w(i, j)$, the weight on the edge between node i and j, computed as the similarity between the measurement-space vector of i and the measurement-space vector of j. Let N be the size of nodeset V. Define the vector d with $d(i)$ given by

$$d(i) = \sum_j w(i, j) \tag{10.7}$$

so that $d(i)$ represents the total connection from node i to all other nodes. Let D be an $N \times N$ diagnonal matrix with d on its diagonal. Let W be an $N \times N$ symmetrical matrix with $W(i, j) = w(i, j)$.

2. Let x be a vector whose components are defined by

$$x_i = \begin{cases} 1 & \text{if node } i \text{ is in } A \\ -1 & \text{otherwise} \end{cases} \tag{10.8}$$

and let y be the continuous approximation to x defined by

$$y = (1 + x) - \frac{\sum_{x_i>0} d_i}{\sum_{x_i<0} d_i}(1 - x). \tag{10.9}$$

Solve the system of equations

$$(D - W)y = \lambda D y \tag{10.10}$$

for the eigenvectors y and eigenvalues λ.

3. Use the eigenvector with the second smallest eigenvalue to bipartition the graph to find the splitting point such that $Ncut$ is minimized.[1]

4. Decide if the current partition should be subdivided further by checking the stability of the cut and making sure that $Ncut$ is below a pre-specified threshold value.

5. Recursively repartition the segmented parts if necessary.

Algorithm 3: Shi's Clustering Procedure

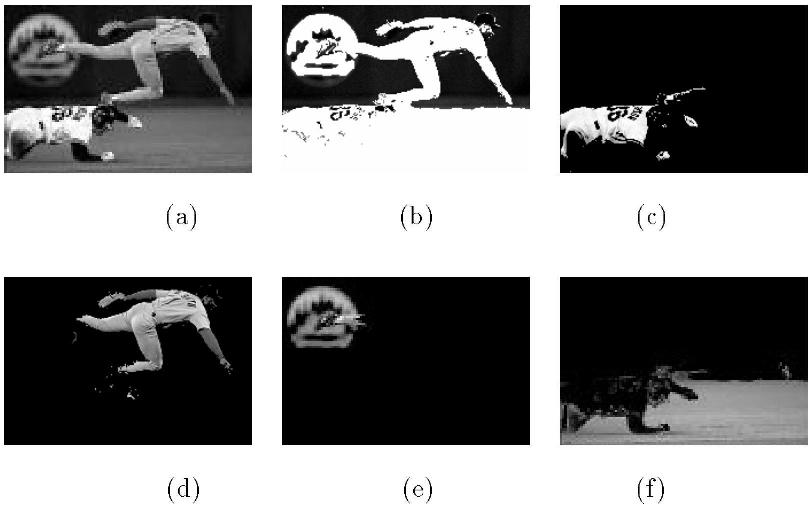


Figure 10.9: Original gray-tone image (a) and regions produced by the Shi segmentation method (b)-(f). In result image (b), the selected region is the dark background region , and it is shown in black. In all other results, the selected region is shown in its original gray tones, with the remainder of the image shown in black. (Courtesy of Jianbo Shi.)

and

$$S^2 = \sum_{[r,c] \in R} (I[r,c] - \overline{X})^2. \tag{10.13}$$

Under the assumption that all the pixels in R and the test pixel y are independent and identically distributed normals, the statistic

$$T = \left[\frac{(N-1)N}{(N+1)} (y - \overline{X})^2 / S^2 \right]^{\frac{1}{2}} \tag{10.14}$$

has a T_{N-1} distribution. If T is small enough y is added to region R and the mean and scatter are updated using y. The new mean and scatter are given by

$$\overline{X}_{\text{new}} \leftarrow (N\overline{X}_{\text{old}} + y)/(N+1) \tag{10.15}$$

and

$$S^2_{\text{new}} \leftarrow S^2_{\text{old}} + (y - \overline{X}_{\text{new}})^2 + N(\overline{X}_{\text{new}} - \overline{X}_{\text{old}})^2. \tag{10.16}$$

If T is too high the value y is not likely to have arisen from the population of pixels in R. If y is different from all of its neighboring regions then it begins its own region. A slightly stricter linking criterion can require that not only must y be close enough to the mean of the neighboring region, but that a neighboring pixel in that region must have a close enough value to y.

To give a precise meaning to the notion of too high a difference, we can use an α level statistical significance test. The fraction α represents the probability that a T statistic with

$N-1$ degrees of freedom will exceed the value $t_{N-1}(\alpha)$. If the observed T is larger than $t_{N-1}(\alpha)$, then we declare the difference to be significant. If the pixel and the segment really come from the same population, the probability that the test provides an incorrect answer is α.

The significance level α is a user-provided parameter. The value of $t_{N-1}(\alpha)$ is higher for small degrees of freedom and lower for larger degrees of freedom. Thus, region scatters considered to be equal, the larger a region is, the closer a pixel's value has to be to the region's mean in order to merge into the region. This behavior tends to prevent already large regions from attracting to it many other additional pixels and tends to prevent the drift of the region mean as the region gets larger. Figure 10.10 illustrates the operation of the Haralick region-growing procedure.

Exercise 3 Region Growing

Implement the Haralick region-growing operator as a program and use it to segment gray-tone images.

10.2 Representing Regions

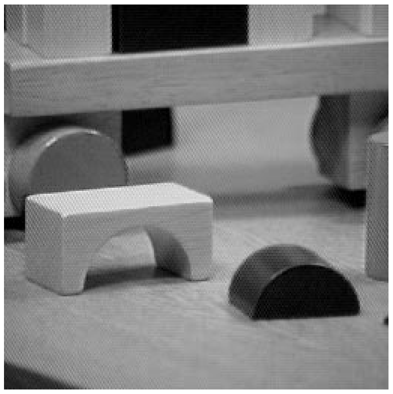
Each algorithm that produces a set of image regions has to have a way to store them for future use. There are several possibilities including overlays on the original images, labeled images, boundary encodings, quad-tree data structures, and property tables. Labeled images are the most commonly used representation. We describe each representation below.

10.2.1 Overlays

An overlay is a method of showing the regions computed from an image by overlaying some color or colors on top of the original image. Many image processing systems provide this operation as part of their image-output procedures. Usually, the original image is a gray-tone image and the overlay color is something that stands out well on the gray tones, such as red or white. To show a region segmentation, one could convert the pixels of the region borders to white and display the transformed gray tone image. Sometimes more than one pixel in width is used to make the region borders stand out. Figure 10.11a shows the borders of selected dark regions, including dark blue referees' jackets and players' numbers, overlayed on a gray-tone football image. Another use for overlays is to highlight certain features of an image. Figure 10.11b reprints an industrial part image from Chapter 1 in which projections of the recognized object models are overlayed on the original gray-tone image.

10.2.2 Labeled Images

Labeled images are good intermediate representations for regions that can also be used in further processing. The idea is to assign each detected region a unique identifier (usually an integer) and create an image where all the pixels of a region will have its unique identifier as their pixel value. Most connected components operators (see Chapter 3) produce this kind of output. A labeled image can be used as a kind of mask to identify pixels of a region in some operation that computes region properties, such as area or length of major axis of



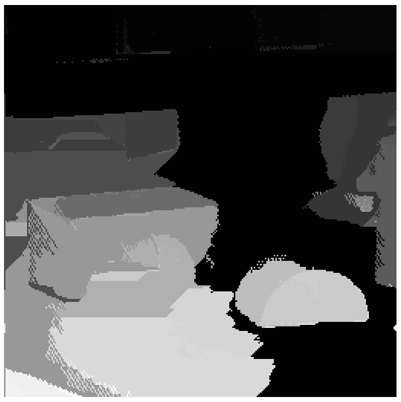
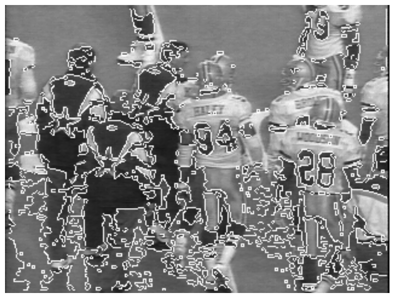
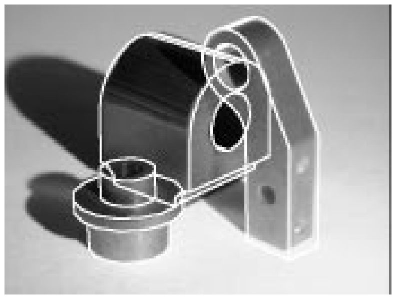


Figure 10.10: The blocks image (left) and a segmentation (right) resulting from application of the Haralick region-growing procedure.



a) region-border overlay



b) wire-frame-model overlay

Figure 10.11: Examples of overlays. a) overlay of selected region borders onto a football image. b) overlay of wire-frame 3D object models onto a industrial parts image.

best-fitting ellipse. It can also be displayed in gray-tone or pseudo-color. If the integer labels are small integers that would all look black in gray tone, the labeled image can be stretched or histogram-equalized to get a better distribution of gray tones. The segmentations of football images earlier in this chapter are labeled images shown in gray tone.

10.2.3 Boundary Coding

Regions can also be represented by their boundaries in a data structure instead of an image. The simplest form is just a linear list of the border pixels of each region. (See the *border* procedure later in this chapter, which extracts the region borders from a labeled image.) A variation of the list of points is the Freeman *chain code*, which encodes the information from the list of points at any desired quantization and uses less space than the original point list. Conceptually, a boundary to be encoded is overlaid on a square grid whose side length determines the resolution of the encoding. Starting at the beginning of the curve, the grid intersection points that come closest to it are used to define small line segments that join each grid point to one of its neighbors. The directions of these line segments are then encoded as small integers from zero to the number of neighbors used in the encoding. Figure 10.12 illustrates this encoding with an eight-neighbor chain code. The line segments are encoded as 0 for a $0°$ segment, 1 for a $45°$ segment, up to 7 for a $315°$ segment. In the figure, a hexagon symbol marks the beginning of the closed curve, and the rest of the grid intersection points are shown with diamonds. The coordinates of the beginning point plus the chain code are enough to reproduce the curve at the resolution of the selected grid. The chain code not only saves space, but can also be used in subseqent operations on the curve itself, such as shape-based object recognition. When a region has not only an external boundary, but also one or more hole boundaries, it can be represented by the chain codes for each of them.

When the boundary does not have to be exact, the boundary pixels can approximated by straight line segments, forming a *polygonal approximation* to the boundary, as shown at the bottom right of Figure 10.12. This representation can save space and simplify algorithms that process the boundary.

10.2.4 Quad Trees

The *quad tree* is another space-saving region representation that encodes the whole region, not just its border. In general, each region of interest would be represented by a quad tree structure. Each node of a quad-tree represents a square region in the image and can have one of three labels: full, *empty*, or *mixed.* If the node is labeled *full*, then every pixel of the square region it represents is a pixel of the region of interest. If the node is labeled *empty*, then there is no intersection between the square region it represents and the region of interest. If the node is labeled *mixed*, then some of the pixels of the square region are pixels of the region of interest and some are not. Only the mixed nodes in a quad tree have children. The full nodes and empty nodes are leaf nodes. Figure 10.13 illustrates a quad-tree representation of an image region. The region looks blocky, because the resolution of the image is only 8×8, which leads to a four-level quad tree. Many more levels would be required to produce a reasonably smoothly curved boundary. Quad trees have been used to represent map regions in geographic information systems.

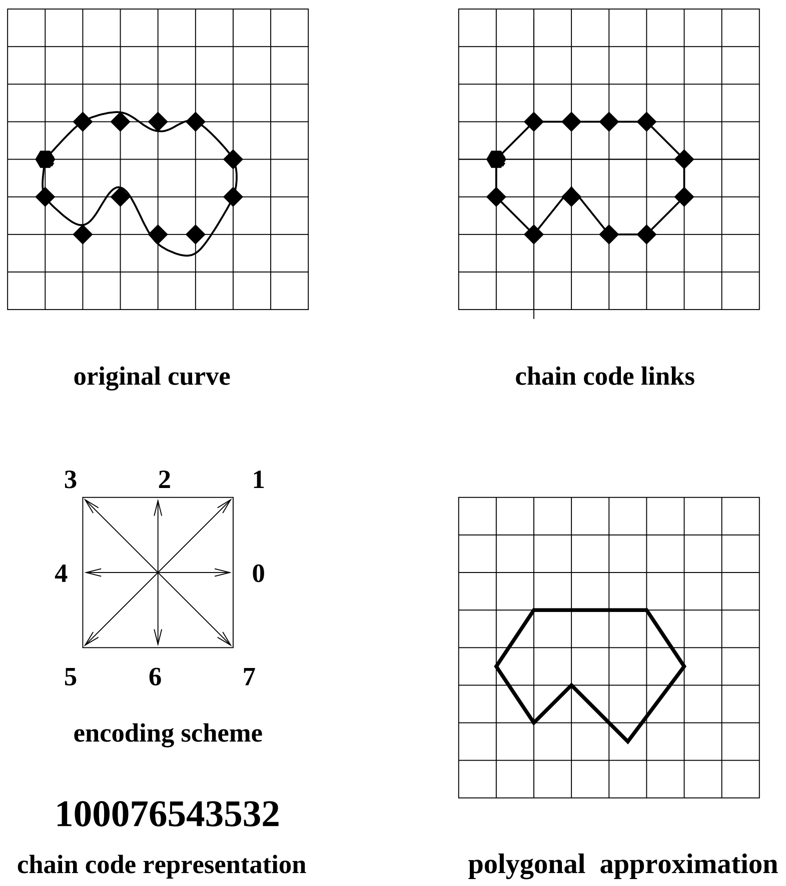


Figure 10.12: Two boundary encodings: chain-code and polygonal approximation. The chain-code boundary encoding uses an 8-symbol code to represent 8 possible angles of line segments that approximate the curve on a square grid. The polygonal approximation uses line segments fit to the original curve; the end points of the line segments have real-valued coordinates and are not constrainted to the original grid points.

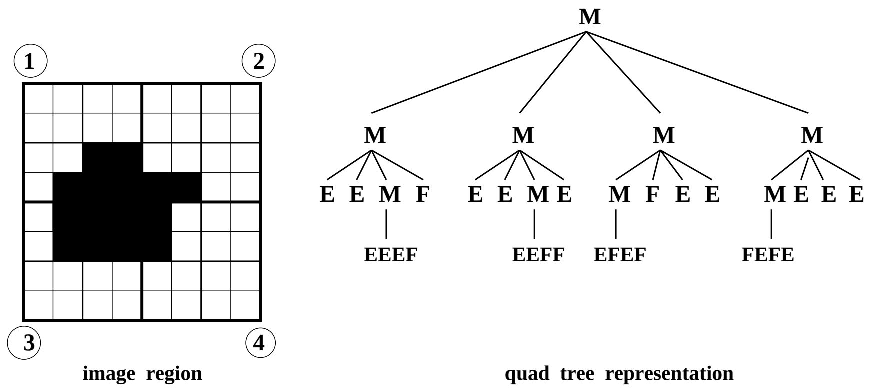


Figure 10.13: A quad tree representation of an image region. The four children of each node correspond to the upper left, upper right, lower left, and lower right quadrants, as illustrated by the numbers in circles for the first level of the tree. M = mixed; E = empty; F = full.

10.2.5 Property Tables

Sometimes we want to represent a region by its extracted properties rather than by its pixels. In this case, the representation is called a *property table*. It is a table in the relational database sense that has a row for each region in the image and a column for each property of interest. Properties can represent the size, shape, intensity, color, or texture of the region. The features described in Chapters 3, 6, and 7 are all possibilities. For example, in a content-based image retrieval system, regions might be described by area, ratio of minor-to-major axis of the best-fitting ellipse, two main colors, and one or more texture measures. Property tables can be augmented to include or to point to the chain-code encoding or quad-tree representation of the region.

Exercise 4 Computing area and perimeter

Consider an image region represented by (a) a labeled image and (b) a chain code representation.

1. Give algorithms for computing the area and the perimeter of the region.
2. Give the running times of your algorithms.

10.3 Identifying Contours

While some image analysis applications work directly with regions, others need the borders of these regions or various other structures, such as line and circular arc segments. This section discusses the extraction of these structures from images.

Exercise 5 Testing for pixels in a region

Consider an image region represented by (a) a labeled image and (b) a polygonal approximation to the boundary.

1. In each case, give an algorithm for testing if an arbitrary pixel [r,c] is in that region.
2. Give the running times of your algorithms in terms of the appropriate parameters, ie. number of pixels in the region or number of segments in the polygonal approximation.

10.3.1 Tracking Existing Region Boundaries

Once a set of regions has been determined by a procedure such as segmentation or connected components, the boundary of each region may be extracted. *Boundary extraction* can be done simply for small-sized images. Scan through the image and make a list of the first border pixel for each connected component. Then for each region, begin at its first border pixel and follow the border of the connected component around in a clockwise direction until the tracking returns to the first border pixel. For large sized images that do not reside in memory, this simple border tracking algorithm results in excessive I/O to the mass storage device on which the image resides.

We will describe an algorithm called *border* which can extract the boundaries for all regions in one left-right, top-bottom scan through the image. *Border* inputs a labeled image and outputs, for each region, a clockwise ordered list of the coordinates of its border pixels. The algorithm is flexible in that it can be easily modified to select the borders of specified regions.

The input to *border* is a labeled image whose pixel values denote region labels. It is assumed that there is one background label that designates those pixels in part of a possibly disconnected background region whose borders do not have to be found. Rather than tracing all around the border of a single region and then moving on to the next region, the border algorithm moves in a left-right, top-bottom scan down the image collecting chains of border pixels that form connected sections of the borders of regions. At any given time during execution of the algorithm, there is a set of *current regions* whose borders have been partially scanned, but not yet output, a set of *past regions* that have been completely scanned and their borders output, and a set of *future regions* that have not yet been reached by the scan.

The data structures contain the chains of border pixels of the current regions. Since there may be a huge number of region labels in the image, but only at most $2 * number\_of\_columns$ may be active at once, a hash table can be used as the device to allow rapid access to the chains of a region given the label of the region. ($2 * number\_of\_columns$ is a safe upper bound; the actual number of regions will be smaller.) When a region is completed and output, it is removed from the hash table. When a new region is encountered during the scan, it is added to the hash table. The hash table entry for a region points to a linked list of chains that have been formed so far for that region. Each chain is a linked list of pixel positions that can be grown from the beginning or the end.

Find the borders of every region of a labeled image S.
S[**R**, **C**] is the input labeled image.
NLINES is the number of rows in the image.
NPIXELS is the number of pixels per row.
NEWCHAIN is a flag that is true when a pixel starts a new chain
and false when a new pixel is added to an existant chain.

```
procedure border(S);
{
 for R:= 1 to NLINES
   {
     for C:= 1 to NPIXELS
       {
       LABEL:= S[R,C];
       if new_region(LABEL) then add(CURRENT,LABEL);
       NEIGHB:= neighbors(R,C,LABEL);
       T:= pixeltype(R,C,NEIGHB);
       if T == 'border'
       then for each pixel N in NEIGHB
         {
         CHAINSET:= chainlist(LABEL);
         NEWCHAIN:= true;
         for each chain X in CHAINSET while NEWCHAIN
           if N==rear(X)
           then { add(X,[R,C]); NEWCHAIN:= false }
         if NEWCHAIN
         then make_new_chain(CHAINSET,[R,C],LABEL);
         }
       }
     for each region REG in CURRENT
       if complete(REG)
       then { connect_chains(REG); output(REG); free(REG) }
   }
}
```

Algorithm 4: Finding the Borders of Labeled Regions

The tracking algorithm examines three rows of the labeled image at a time: the current row being processed; the row above it; and the row below it. Two dummy rows of background pixels are appended to the image, one on top and one on the bottom, so that all rows can be treated alike. The algorithm for an NLINES by NPIXELS labeled image S is as follows.

In this procedure, S is the name of the labeled image; thus S[R,C] is the value (LABEL) of the current pixel being scanned. If this is a new label, it is added to the set CURRENT of current region labels. NEIGHB is the list of only neighbors of pixel [R,C] which have label LABEL. The function *pixeltype* looks at the values of [R,C] and its neighbors to decide if [R,C] is a nonbackground border pixel. If so, the procedure searches for a chain of the region with label LABEL that has a neighbor of [R,C] at its rear, and, if it finds one, appends [R,C] to the end of the chain by the procedure *add* whose first argument is a chain and whose second argument is [R,C]. If no neighbor of [R,C] is at the rear of a chain of this region, then a new chain is created containing [R,C] as its only element by the procedure *make_new_chain* whose first argument is the set of chains in which a new chain is being added whose sole element is the location [R,C] which is its second argument. Its third argument is the label LABEL to be associated with the chain.

After each row R is scanned, the chains of those current regions whose borders are now complete are merged into a single border chain which is output. The hash table entrees and list elements associated with those regions are then freed. Figure 10.14 shows a labeled image and its output from the border procedure.

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	0	2	2	0
3	0	1	1	1	2	2	0
4	0	1	1	1	2	2	0
5	0	1	1	1	2	2	0
6	0	0	0	0	2	2	0
7	0	0	0	0	0	0	0

(a) A labeled image with two regions.

Region	Length	List
1	8	(3,2)(3,3)(3,4)(4,4)(5,4)(5,3)(5,2)(4,2)
2	10	(2,5)(2,6)(3,6)(4,6)(5,6)(6,6)(6,5)(5,5) (4,5)(3,5)

(b). The output of the border procedure for the labeled image.

Figure 10.14: Action of the border procedure on a labeled image.

Exercise 6 Limitations of the border tracking algorithm.

The border tracking algorithm makes certain assumptions about the regions that it is tracking. Under what conditions could it fail to properly identify the border of a region?

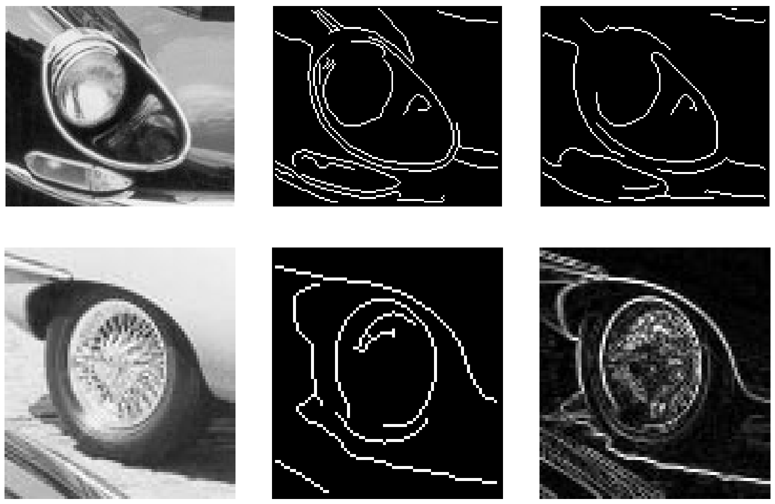


Figure 10.15: (Top left) Image of headlight of a black car; (top center) results of Canny operator with $\sigma = 1$; (top right) results of Canny operator with $\sigma = 4$; (bottom left) image of car wheel; (bottom center) results of Canny operator with $\sigma = 1$; (bottom right) results of Roberts operator. Note in the top row how specular reflection at the top left distracts the edge detector from the boundary of the chrome headlight rim. In the bottom row, note how the shadow of the car connects to the tire which connects to the fender: neither the tire nor the spokes are detected well.

10.3.2 The Canny Edge Detector and Linker

The Canny edge detector and linker extracts boundary segments of an intensity image. It was briefly introduced in Chapter 5 along with other edge detectors. The Canny operator is often used and recent work comparing edge operators justifies its popularity. Examples of its were provided in Chapter 5: Figure 10.15 shows two examples of images of car parts taken from a larger image shown in Chapter 2. Both of these show well-known problems with all edge detection and boundary following algorithms: the contour segments from actual object parts erroneously merge with contour segments from illumination or reflectance boundaries. Such contours are difficult to analyze in a bottom-up manner by a general object recognition system. However, top-down matching of such representations to models of specific objects can be done successfully, as we shall see in subsequent chapters. Thus, the quality of these edge representations of images depends upon their use in the overall machine vision system.

The Canny edge detection algorithm defined in Algorithm 5 produces thin fragments of image contours and is controlled by the single smoothing parameter σ. The image is first smoothed with a Gaussian filter of spread σ and then gradient magnitude and direction are computed at each pixel of the smoothed image. Gradient direction is used to thin edges by

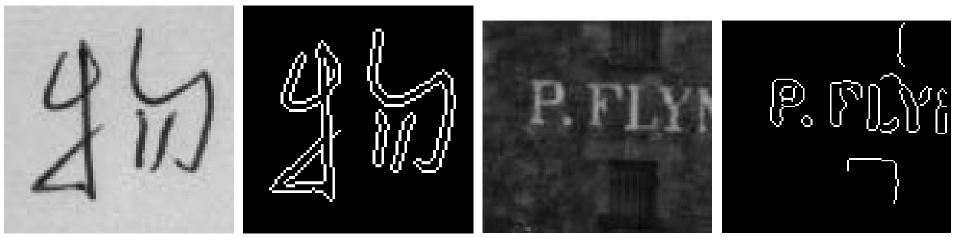


Figure 10.16: Identifying regions corresponding to symbols on surfaces is often easy because they are created with good contrast. These results were obtained by applying only the Canny operator: (left set) character carefully written with ink on paper; (right set) weathered signage on a brick wall.

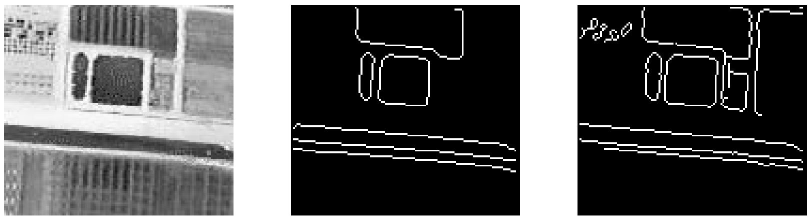


Figure 10.17: Contours from an aerial image of farm fields defined using the Canny operator with $\sigma = 2$ and $\sigma = 1$ respectively. Note that five major structures are well represented – three fields and two straight horizontal bands in the bottom of the image (a canal and a road alongside it).

suppressing any pixel response that is not higher than the two neighboring pixels on either side of it along the direction of the gradient. This is called *nonmaximum suppression*, a good operation to use with any edge operator when thin boundaries are wanted. The two 8-neighbors of a pixel $[x, y]$ that are to be compared are found by rounding off the computed gradient direction to yield one neighbor on each side of the center pixel. Once the gradient magnitudes are thinned, high magnitude contours are tracked. In the final aggregation phase, continuous contour segments are sequentially followed. Contour following is initiated only on edge pixels where the gradient magnitude meets a high threshold; however, once started, a contour may be followed through pixels whose gradient magnitude meet a lower threshold, usually about half of the higher starting threshold.

Image regions can sometimes be detected when boundary segments close on themselves. Examples of this are shown in Figures 10.16 and 10.17. Such segments can be further analyzed by segmenting the set of boundary pixels into straight or circular sides, etc. For example, the boundary of a rectangular building might result in four straight line segments. Straight line segments can be identified by the Hough transform or by direct fitting of a parameteric line model.

10.3.3 Aggregating Consistent Neighboring Edgels into Curves

The border-tracking algorithm in Section 10.3.1 required as input a labeled image denoting a set of regions. It tracked along the border of each region as it scanned the image, row by

```
Compute thin connected edge segments in the input image.

I[x, y] : input intensity image; σ : spread used in Gaussian smoothing;
E[x, y] : output binary image;
IS[x, y] : smoothed intensity image;
Mag[x, y] : gradient magnitude; Dir[x, y] : gradient direction;
$T_{low}$ is low intensity threshold; $T_{high}$ is high intensity threshold;


procedure Canny( I[], σ);
{
      IS[] = image I[] smoothed by convolution with Gaussian $G_\sigma(x, y)$;
      use Roberts operator to compute Mag[x, y] and Dir[x, y] from IS[];
      Suppress_Nonmaxima( Mag[], Dir[], $T_{low}$, $T_{high}$ );
      Edge_Detect( Mag[], $T_{low}$, $T_{high}$, E[] );
}
 procedure Suppress_Nonmaxima( Mag[], Dir[] );
{
      define +Del[4] =  (1,0), (1,1), (0,1) (-1,1);
      define -Del[4] =  (-1,0), (-1-,1), (0,-1) (1,-1);
        for x := 0 to MaxX-1;
        for y := 0 to MaxY-1;
        {
          direction := ( Dir[x, y] + π/8 ) modulo π/4;
          if ( $\mathbf{Mag}[x, y] \leq \mathbf{Mag}[(x, y) + Del[direction]]$) then Mag[x, y] := 0;
          if ( $\mathbf{Mag}[x, y] \leq \mathbf{Mag}[(x, y) + -Del[direction]]$) then Mag[x, y] := 0;
        }
} procedure Edge_Detect( Mag[], $T_{low}$, $T_{high}$, E[] );
{
      for x := 0 to MaxX - 1;
      for y := 0 to MaxY - 1;
        {
          if ($\mathbf{Mag}[x, y] \geq T_{high}$) then Follow_Edge( x, y, Mag[], $T_{low}$, $T_{high}$, E[] );
        } ;
}
 procedure Follow_Edge( x, y, Mag[], $T_{low}$, $T_{high}$, E[] );
{
      E[x, y] := 1;
      while $\mathbf{Mag}[u, v] > T_{low}$ for some 8-neighbor [u, v] of [x, y]
        {
          E[u, v] := 1;
          [x, y] := [u, v];
        } ;
}
```

Algorithm 5: Canny Edge Detector

Exercise 7

Consider the contour following phase of the Canny edge detection algorithm. When following an image contour by tracing pixels of high gradient magnitude, would it be a good idea to select the next possible pixel only from the two neighbors that are perpendicular to the gradient direction? Why or why not? Show specific cases to support your answer.

Exercise 8 Measuring across Canny edges

Perform the following experiment. Obtain a program for the Canny edge detector or some image tool that contains it. Obtain some flat objects with precise parallel edges, such as razor blades, and some rounded objects, such as the shanks of drill bits. Image several of these objects in several different orientations: use high resolution scanning, if possible. Apply the Canny edge detector and study the quality of edges obtained, including the repeatability of the distance across parallel edges. Is there any difference between measuring the razor blades, which have "sharp edges" and measuring the drill bits, which may have "soft edges" in the image.

row. Because of the assumption that each border bounded a closed region, there was never any point at which a border could be split into two or more segments. When the input is instead a labeled edge image with a value of 1 for edge pixels and 0 for non-edge pixels, the problem of tracking edge segments is more complex. Here it is not necessary for edge pixels to bound closed regions and the segments consist of connected edge pixels which go from end point, corner, or junction to endpoint, corner, or junction with no intermediate junctions or corners. Figure 10.18 illustrates such a labeled edge image. Pixel (3,3) of the image is a *junction* pixel where three different edge (line) segments meet. Pixel (5,3) is a corner pixel and may be considered a segment end point as well, if the application requires ending segments at corners. An algorithm that tracks segments like these has to be concerned with the following tasks:

1. starting a new segment
2. adding an interior pixel to a segment
3. ending a segment
4. finding a junction
5. finding a corner

As in border tracking, efficient data structure manipulation is needed to manage the information at each step of the procedure. The data structures used are very similar to those used in the border algorithm. Instead of past, current, and future regions, there are past, current, and future segments. Segments are lists of edge points that represent straight or curved lines on the image. Current segments are kept in internal memory and accessed by a hash table. Finished segments are written out to a disk file and their space in the hash table freed. The main difference is the detection of junction points and the segments entering them from above or the left and the segments leaving them from below or the right. We will assume an extended neighborhood operator called *pixeltype* that determines if a pixel is an isolated point, the starting point of a new segment, an interior pixel of an old segment, an ending point of an old segment, a junction or a corner. If the pixel is an interior or end

	1	2	3	4	5
1	**1**	0	0	0	**1**
2	0	**1**	0	**1**	0
3	0	0	**1**	0	0
4	0	0	**1**	0	0
5	0	0	**1**	**1**	**1**

Figure 10.18: Labeled edge image containing a junction of three line segments at pixel (3,3) and a potential corner at pixel (5,3).

point of an old segment, the segment id of the old segment is also returned. If the pixel is a junction or a corner point, then a list (INLIST) of segment ids of incoming segments plus a list (OUTLIST) of pixels representing outgoing segments are returned. A procedure for tracking edges in a labeled image is given below. Figure 10.19 gives the results of its application on the labeled image of Figure 10.18.

Segment ID	Length	List
1	3	(1,1)(2,2)(3,3)
2	3	(1,5)(2,4)(3,3)
3	3	(3,3)(4,3)(5,3)
4	3	(5,3)(5,4)(5,5)

Figure 10.19: Output of the *edge_track* procedure on the image of Fig. 10.18, assuming the point (5,3) is judged to be a corner point. If corner points are not used to terminate segments, then segement 3 would have length 5 and list ((3,3)(4,3)(5,3)(5,4)(5,5)).

The details of keeping track of segment ids entering and leaving segments at a junction have been supressed. This part of the procedure can be very simple and assume every pixel adjacent to a junction pixel is part of a different segment. In this case, if the segments are more than one-pixel wide, the algorithm will detect a large number of small segments that are really not new line segments at all. This can be avoided by applying a connected shrink operator to the edge image. Another alternative would be to make the *pixeltype* operator even smarter. It can look at a larger neighborhood and use heuristics to decide if this is just a thick part of the current segment, or a new segment is starting. Often the application will dictate what these heuristics should be.

10.3.4 Hough Transform for Lines and Circular Arcs

The *Hough transform* is a method for detecting straight lines and curves in gray-tone (or color) images. The method is given the family of curves being sought and produces the set of curves from that family that appear on the image. In this section we describe the Hough transform technique, and show how to apply it to finding straight line segments and circular arcs in images.

Find the line segments of binary edge image S.
S[R, C] is the input labeled image.
NLINES is the number of rows in the image.
NPIXELS is the number of pixels per row.
IDNEW is the ID of the newest segment.
INLIST is the list of incoming segment IDs returned by pixeltype.
OUTLIST is the list of outgoing segment IDs returned by pixeltype.

```
procedure edge_track(S);
{
IDNEW := 0;
for R:= 1 to NLINES
  for C := 1 to NPIXELS
    if S[R,C] $\neq$ background pixel
    {
    NAME := address[R,C]; NEIGHB := neighbors[R,C];
    T := pixeltype(R,C,NEIGHB,ID,INLIST,OUTLIST);
    case
      T = isolated point : next;
      T = start point of new segment: {
        IDNEW := IDNEW + 1;
        make_new_segment(IDNEW,NAME); } ;
      T = interior point of old segment : add(ID,NAME);
      T = end point of old segment : {
        add(ID,NAME);
        output(ID); free(ID) } ;
      T = junction or corner point:
        for each ID in INLIST {
          add(ID,NAME);
          output(ID); free(ID); } ;
        for each pixel in OUTLIST {
          IDNEW := IDNEW + 1;
          make_new_segment(IDNEW,NAME); } ;
    } }
```

Algorithm 6: Tracking Edges of a Binary Edge Image

Exercise 9 Determining the type of a pixel

Give the code for the operator *pixeltype*, using a 3×3 neighborhood about a pixel to classify it as one of the types: isolated, start or end, interior, junction, and corner.

The Hough Transform Technique

The Hough transform algorithm requires an accumulator array whose dimension corresponds to the number of unknown parameters in the equation of the family of curves being sought. For example, finding line segments using the equation $y = mx + b$ requires finding two parameters for each segment: m and b. The two dimensions of the accumulator array for this family would correspond to quantized values for m and quantized values for b. The accumulator array accumlates evidence for the existence of the line $y = mx + b$ in bin $A[M, B]$ where M and B are quantizations of m and b, respectively.

Using an accumulator array A, the Hough procedure examines each pixel and its neighborhood in the image. It determines if there is enough evidence of an edge at that pixel, and if so calculates the parameters of the specified curve that passes through this pixel. In the straight line example with equation $y = mx + b$, it would estimate the m and the b of the line passing through the pixel being considered if the measure of edge strength (such as gradient) at that pixel were high enough. Once the parameters at a given pixel are estimated, they are quantized to corresponding values M and B and the accumulator $A[M, B]$ is incremented. Some schemes increment by one and some by the strength of the gradient at the pixel being processed. After all pixels have been processed, the accumulator array is searched for peaks. The peaks indicate the parameters of the most likely lines in the image.

Although the accumulator array tells us the parameters of the infinite lines (or curves), it does not tell us where the actual segments begin and end. In order to have this information, we can add a parallel structure called *PTLIST*. *PTLIST*$[M, B]$ contains a list of all the pixel positions that contributed to the sum in accumulator $A[M, B]$. From these lists the actual segments can be determined.

The above description of the Hough method is general; it leaves out the details needed for an implementation. We will now discuss algorithms for straight line and circle finding in detail.

Finding Straight Line Segments

The equation $y = mx + b$ for straight lines does not work for vertical lines. A better model is the equation $d = x \cos\theta + y \sin\theta$ where d is the perpendicular distance from the line to the origin and θ is the angle the perpendicular makes with the x-axis. We will use this form of the equation but convert to row (r) and column (c) coordinates. Since the column coordinate c corresponds to x and the row coordinate r corresponds to $-y$, our equation becomes

$$d = c \cos\theta - r \sin\theta \qquad (10.17)$$

where d is the perpendicular distance from the line to the origin of the image (assumed to be at upper left), and θ is the angle this perpendicular makes with the c (column) axis. Figure 10.20 illustrates the parameters of the line segment. Suppose the point where the perpendicular from the origin intersects the line is (50,50) and that $\theta = 315°$. Then we have

$$d = 50cos(315) - 50sin(315) = 50(.707) - 50(-.707) \approx 70$$

The accumulator A has subscripts that represent quantized values of d and θ. O'Gorman and Clowes quantized the values of d by 3's and θ by 10° increments in their experiments

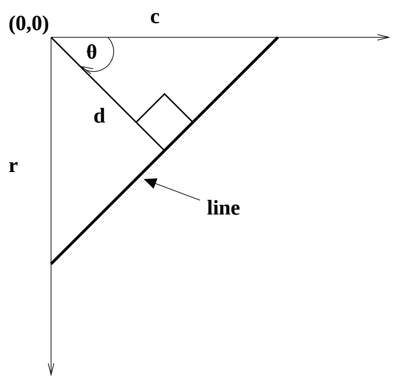


Figure 10.20: illustrates the parameters d and θ used in the equation $d = -r \sin\theta + c \cos\theta$ of a straight line.

on gray level images of puppet objects. An accumulator array quantized in this fashion is illustrated in Fig. 10.21. The O'Gorman and Clowes algorithm for filling the accumulator A and parallel list array $PTLIST$ is given in procedure *accumulate_lines* below.

The algorithm is expressed in (row,column) space. The functions *row_gradient* and *column_gradient* are neighborhood functions that estimate the row and column components of the gradient, while the function *gradient* combines the two to get the magnitude. The function *atan2* is the standard scientific library function that returns the angle in the correct quadrant given the row and column components of the gradient. We assume here that *atan2* returns a value between $0°$ and $359°$. Many implementations return the angle in radians which would have to be converted to degrees. If the distance d comes out negative (i.e. for $\theta = 135°$), its absolute value gives the distance to the line. The actions of the procedure are illustrated in Fig. 10.22. Notice that with a 3×3 gradient operator, the lines are two pixels wide. Notice also that counts appear in other accumulators than the two correct ones.

Procedure *accumulate_lines* is O'Gorman and Clowes' version of the Hough method. Once the accumulator and list arrays are filled, though, there is no standard method for extracting the line segments. Their ad hoc procedure, which illustrates some of the problems that come up in this phase of the line segment extraction process, is expressed as follows:

The function *pick_greatest_bin* returns the value in the largest accumulator while setting its last two parameters, DQ and THETAQ, to the quantized d and θ values for that bin. The *reorder function* orders the list of points in a bin by column coordinate for $\theta < 45$ or $\theta > 135$ and by row coordinate for $45 \leq \theta \leq 135$. The arrays D and THETA are expected to hold the quantized D and THETA values for a pixel that were computed during the accumulation. Similarly the array GRADIENT is expected to contain the computed gradient magnitude. These can be saved as intermediate images. The *merge* procedure merges the list of points from a neighbor of a pixel in with the list of points for that pixel, keeping the spatial

Accumulate the straight line segments in gray-tone image S to accumulator A.
S[**R**, **C**] is the input gray-tone image.
NLINES is the number of rows in the image.
NPIXELS is the number of pixels per row.
A[**DQ**, **THETAQ**] is the accumulator array.
DQ is the quantized distance from a line to the origin.
THETAQ is the quantized angle of the normal to the line.

```
procedure accumulate_lines(S,A);
{
A := 0;
PTLIST := NIL;
for R := 1 to NLINES
  for C := 1 to NPIXELS
    {
    DR := row_gradient(S,R,C);
    DC := col_gradient(S,R,C);
    GMAG := gradient(DR,DC);
    if GMAG > gradient_threshold
      {
      THETA := atan2(DR,DC);
      THETAQ := quantize_angle(THETA);
      D := abs(C*cos(THETAQ) - R*sin(THETAQ));
      DQ := quantize_distance(D);
      A[DQ,THETAQ] := A[DQ,THETAQ]+GMAG;
      PTLIST(DQ,THETAQ) := append(PTLIST(DQ,THETAQ),[R,C])
      }
    }
}
```

Algorithm 7: Hough Transform for Finding Straight Lines

Find the point lists corresponding to separate line segments.
A[DQ, THETAQ] is the accumulator array from accumulate_lines.
DQ is the quantized distance from a line to the origin.
THETAQ is the quantized angle of the normal to the line.

```
procedure find_lines;
{
V := pick_greatest_bin(A,DQ,THETAQ);
while V > value_threshold
  {
  list_of_points := reorder(PTLIST(DQ,THETAQ));
  for each point [R,C] in list_of_points
    for each neighbor (R',C') of [R,C] not in list_of_points
      {
      DPRIME := D(R',C');
      THETAPRIME := THETA(R',C');
      GRADPRIME := GRADIENT(R',C');
      if GRADPRIME > gradient_threshold
        and abs(THETAPRIME–THETA)≤ 10
      then {
            merge(PTLIST(DQ,THETAQ),PTLIST(DPRIME,
              THETAPRIME));
            set_to_zero(A,DPRIME,THETAPRIME);
          }
      }
  final_list_of_points := PTLIST(DQ,THETAQ);
  create_segments(final_list_of_points);
  set_to_zero(A,DQ,THETAQ);
  V := pick_greatest_bin(A,DQ,THETAQ);
  }
}
```

Algorithm 8: O'Gorman/Clowes Method for Extracting Straight Lines

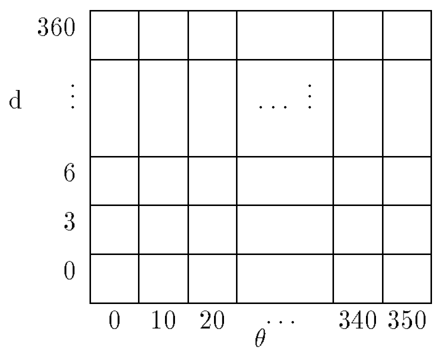


Figure 10.21: illustrates the accumulator array for finding straight line segments in images of size 256×256.

ordering. The *set_to_zero* procedure zeroes out an accumulator so that it will not be reused. Finally, the procedure *create_segments* goes through the final ordered set of points searching for gaps longer than one pixel. It creates and saves a set of line segments terminating at gaps. For better accuracy, a least squares procedure can be used to fit lists of points to line segments. It is important to mention that the Hough procedure can gather strong evidence from broken or virtual lines such as a row of stones or a road broken by overhanging trees.

Exercise 10

This exercise follows the work of R. Kasturi: the problem is to apply the Hough transform to identify lines of text. Apply existing programs or tools and write new ones as needed to perform the following experiment. (a) Type or print a few lines of text in various directions and binarize the image. Add some other objects, such as blobs or curves. (b) Apply connected components processing and output the centroids of all objects whose bounding boxes are appropriate for characters. (c) Input the set of all selected centroids to a Hough line detection procedure and report on how well the text lines can be detected.

Finding Circles

The Hough transform technique can be extended to circles and other parametrized curves. The standard equation of a circle has three parameters. If a point $[R, C]$ lies on a circle then the gradient at $[R, C]$ points to the center of that circle as shown in Fig. 10.23. So if a point $[R, C]$ is given, a radius d is selected, and the direction of the vector from $[R, C]$ to the center is computed, the coordinates of the center can be found. The radius, d, the row-coordinate of the center, r_o, and the column-coordinate of the center, c_o, are the three parameters used to vote for circles in the Hough algorithm. In row-column coordinates, circles are represented by the equations

$$r = r_o + d \sin\theta \tag{10.18}$$

$$c = c_o - d \cos\theta \tag{10.19}$$

	1	2	3	4	5
1	0	0	0	100	100
2	0	0	0	100	100
3	0	0	0	100	100
4	100	100	100	100	100
5	100	100	100	100	100

gray level image

	1	2	3	4	5
1	0	0	300	300	0
2	0	0	300	300	0
3	0	0	200	200	0
4	0	0	0	100	0
5	0	0	0	0	0

column gradient

	1	2	3	4	5
1	0	0	0	0	0
2	0	0	0	0	0
3	300	300	200	100	0
4	300	300	200	100	0
5	0	0	0	0	0

row gradient

	1	2	3	4	5
1	-	-	0	0	-
2	-	-	0	0	-
3	90	90	45	26.6	-
4	90	90	90	45	-
5	-	-	-	-	-

THETA

	1	2	3	4	5
1	-	-	0	0	-
2	-	-	0	0	-
3	90	90	40	20	-
4	90	90	90	40	-
5	-	-	-	-	-

THETAQ

	1	2	3	4	5
1			3	4	
2			3	4	
3	3	3	4.2	4.8	
4	4	4	4	5.6	
5					

D

	1	2	3	4	5
1			3	3	
2			3	3	
3	3	3	3	3	
4	3	3	3	3	
5					

DQ

DQ							
360							
⋮							
6							
3	4		1		2		5
0							
	0	10	20	30	40	⋯	90

THETAQ
accumulator A

DQ							
360							
⋮							
6							
3	♣		♢		♠		♡
0							
	0	10	20	30	40	⋯	90

THETAQ
PTLIST

♣ (1,3)(1,4)(2,3)(2,4)
♢ (3,4)
♠ (3,3)(4,4)
♡ (3,1)(3,2)(4,1)(4,2)(4,3)

Figure 10.22: The results of the operation of procedure *accumulate* on a simple gray level image using Prewitt masks. For this small example, the evidence for correct detections is not much larger than that for incorrect ones, but in real images with long segments, the difference will be much more pronounced.

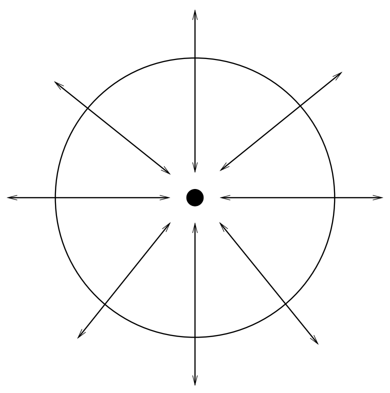


Figure 10.23: illustrates the direction of the gradient at the boundary points of a circle. The inward pointing gradients are the ones that will accumulate evidence for the center of the circle.

(10.20)

With these equations, the accumulate algorithm for circles becomes algorithm accumulate_circles on the next page.

This procedure can easily be modified to take into account the gradient magnitude as did the procedure for line segments. The results of applying it to a technical document image are shown in Figure 10.24.

Extensions

The Hough transform method can be extended to any curve with analytic equation of the form $f(\mathbf{x}, \mathbf{a}) = 0$ where $\mathbf{x}$ denotes an image point and $\mathbf{a}$ is a vector of parameters. The procedure is as follows:

1. Initialize accumulator array $A[\mathbf{a}]$ to zero.
2. For each edge pixel $\mathbf{x}$ determine each $\mathbf{a}$ such that $f(\mathbf{x}, \mathbf{a}) = 0$ and set A[$\mathbf{a}$] := A[$\mathbf{a}$]+1.
3. Local maxima in A correspond to curves of f in the image.

If there are m parameters in $\mathbf{a}$, each having M discrete values, then the time complexity is $O(M^{m-2})$. The Hough transform method has been further generalized to arbitrary shapes specified by a sequence of boundary points (Ballard, 1981). This is known as the *generalized Hough transform.*

The Burns Line Finder

A number of hybrid techniques exist that use some of the principles of the Hough transform. The Burns line finder (Burns et al, 1986) was developed to find straight lines in complex images of outdoor scenes. The Burns method can be summarized as follows:

Accumulate the circles in gray-tone image S to accumulator A.
S[**R**, **C**] is the input gray-tone image.
NLINES is the number of rows in the image.
NPIXELS is the number of pixels per row.
A[**R**, **C**, **RAD**] is the accumulator array.
R is the row index of the circle center.
C is the column index of the circle center.
RAD is the radius of the circle.

```
procedure accumulate_circles(S,A);
{
A := 0;
PTLIST := 0;
for R := 1 to NLINES
  for C := 1 to NPIXELS
    for each possible value RAD of radius
      {
      THETA := compute_theta(S,R,C,RAD);
      R0 := R - RAD*cos(THETA);
      C0 := C + RAD*sin(THETA);
      A[R0,C0,RAD] := A[R0,C0,RAD]+1;
      PTLIST(R0,C0,RAD) := append(PTLIST(R0,C0,RAD),[R,C])
      }
}
```

Algorithm 9: Hough Transform for Finding Circles

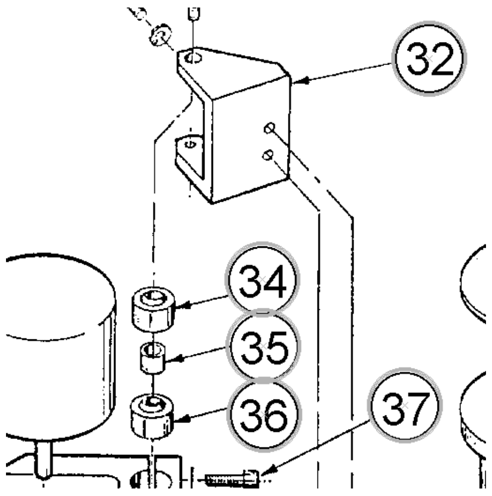


Figure 10.24: Circles detected by the Hough transform on a section of an technical drawing, shown by overlaying an extra circle of slightly larger radius on each detected circle.

1. Compute the gradient magnitude and direction at each pixel.

2. For points with high enough gradient magnitude, assign two labels representing two different quantizations of the gradient direction. (For example, for eight bins, if the first quantization is 0 to 44, 45 to 90, 91 to 134, etc., then the second can be -22 to 22, 23 to 67, 68 to 112, etc.) The result is two symbolic images.

3. Find the connected components of each symbolic image and compute line length for each component.

   - Each pixel is a member of two components, one from each symbolic image.
   - Each pixel votes for its *longer* component.
   - Each component receives a count of pixels that voted for it.
   - The components (line segments) that receive the majority support are selected.

The Burns line finder takes advantage of two powerful algorithms: the Hough transform and the connected components algorithm. It attempts to get rid of the quantization problems that forced O'Gorman and Clowes to search neighboring bins by the use of two separate quantizations. In practice, it suffers from a problem that will affect any line finder that estimates angle based on a small neighborhood around a pixel. Digital lines are not straight. Diagonal lines are really a sequence of horizontal and vertical steps. If the angle detection technique uses too small a neighborhood, it will end up finding a lot of tiny horizontal and vertical segments instead of a long diagonal line. Thus in practice, the Burns line finder and any other angle-based line finder can break up lines that a human would like to detect as a connected whole.

Exercise 11 Burns compared to Hough

Implement both the Hough transform and the Burns operator for line finding and compare the results on real-world images having a good supply of straight lines.

Exercise 12 Line Detection

Implement the following approach to detecting lines in a gray-tone image I.

```
for all image pixels I[R,C]
  {
  compute the gradient $G_{mag}$ and $G_{dir}$
  if $G_{mag}$ > threshold
  then output [$G_{mag}$,$G_{dir}$] to set $H$
  }
detect clusters in the set $H$;
```

The significant clusters will correspond to the significant line segments in I.

10.4 Fitting Models to Segments

Mathematical models that fit data not only reveal important structure in the data, but also can provide efficient representations for further analysis. A straight line model might be used for the edge pixels of a building or a planar model might apply to surface data from the face of a building. Convenient mathematical models exist for circles, cylinders, and many other shapes.

Below, we present the *method of least squares* for determining the parameters of the *best* mathematical model fitting observed data. This data might be obtained using one of the region or boundary segmentation methods described previously; for example, we have already mentioned that we could fit a straight line model to all the pixels $[r, c]$ voting for a particular line hypothesis $A[THETAQ, DQ]$ in the Hough accumulator array. In order to apply least squares, there must be some way of establishing which model or models should be tried out of an infinite number of possibilities. Once a model is fit and its parameters determined, it is possible to determine whether or not the model fits the data acceptably. A good fit might mean that an object of some specified shape has been detected; or, it might just provide a more compact representation of the data for further analysis.

Fitting a Straight Line

We introduce the least-squares theory by way of a simply example. One straight line model is a function with two parameters: $y = f(x) = c_1 x + c_0$. Suppose we want to test whether or not a set of observed points $\{(x_j, y_j), j = 1, n\}$ form a line. To do this, we determine the best parameters c_1 and c_0 of the linear function and then examine how close the observed points are to the function. Different criteria can be used to quantify how close the observations are to the model. Figure 10.25 shows the fitting of a line to six data points. Surely, we could move the line slightly and we would have a different line that still fits well.

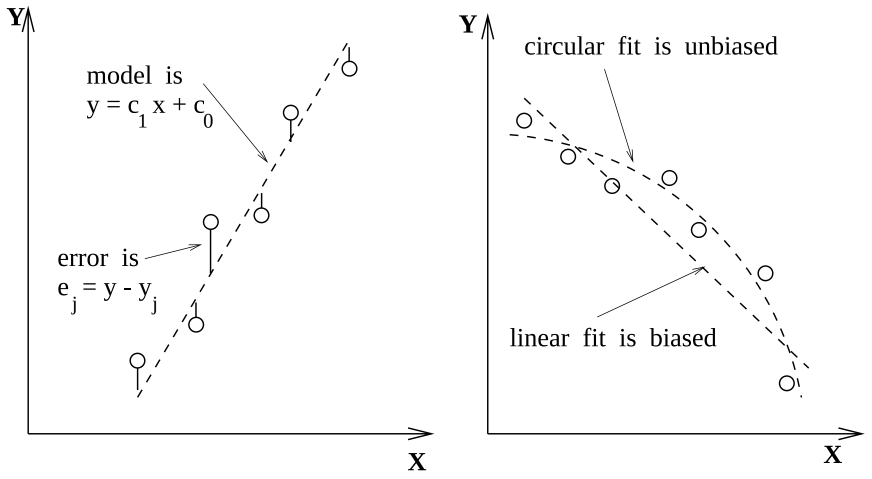


Figure 10.25: (Left) Fit of model $y = f(x)$ to six data points; (right) competing straight line and circular models: the signs of the residual errors show that the line fit is biased and the circular fit is unbiased.

The *least-squares criteria* defines a best line according to the definition below.

1 Definition **Least-Squares Error Criteria:** *The measure of how well a model $y = f(x)$ fits a set of n observations $\{(x_j, y_j), j = 1, n\}$ is*

$$LSE \;=\; \sum_{j=1}^{n} (f(x_j) - y_j)^2$$

The best model $y = f(x)$ is the model with the parameters minimizing this criteria.

2 Definition *The* **root-mean-square error, or RMSE**, *is the average difference of observations from the model:*

$$RMSE \;=\; [\, \sum_{j=1}^{n} (f(x_j) - y_j)^2)/n \,]^{1/2}$$

Note that for the straight line fit, this difference is not the Euclidean distance of the observed point from the line, but the distance measured parallel to the y-axis as Figure 10.25.

3 Definition **Max-Error Criteria:** *The measure of how well a model $y = f(x)$ fits a set of n observations $\{(x_j, y_j), j = 1, n\}$ is*

$$MAXE \;=\; max(\{\mid (f(x_j) - y_j) \mid\}_{j=1,n})$$

Note that this measure depends only on the worst fit point, whereas the RMS error depends on the fit of all of the points.

Table 10.1: Least-squares fit of data generated using $y = 3x - 7$ plus noise gives fitted model $y = 2.971x - 6.962$.

Data Pts (x_j, y_j)	(0.0, -6.8)	(1.0, -4.1)	(2.0, -1.1)	(3.0, 1.8)	(4.0, 5.1)	(5.0, 7.9)
Residuals $y - y_j$:	-0.162	0.110	0.081	0.152	-0.176	-0.005

Closed Form Solutions for Parameters

The least-squares criteria is popular for two reasons; first, it is a logical choice when a Gaussian noise model holds, second, derivation of a closed form solution for the parameters of the best model is easy. We first derive the closed form solution for the parameters of the best fitting straight line. Development of other models follows the same pattern. The least-squares error for the straight line model can be explicitly written as follows: note that the observed data x_j, y_j are regarded as constants in this formula.

$$LSE \;=\; \varepsilon(c_1, c_0) \;=\; \sum_{j=1}^{n} (c_1 x_j + c_0 \;- y_j)^2 \tag{10.21}$$

The error function ε is a smooth non-negative function of the two parameters c_1 and c_0 and will have a global minimum at the point (c_1, c_0) where $\partial\varepsilon/\partial c_1 = 0$ and $\partial\varepsilon/\partial c_0 = 0$. Writing out these derivatives from the formula in Equation 10.21 and using the fact that the derivative of a sum is the sum of the derviatives yields the following derivation.

$$\begin{aligned}
\partial\varepsilon/\partial c_1 &= \sum_{j=1}^{n} 2(c_1 x_j + c_0 \;- y_j) x_j = 0 && (10.22)\\
&= 2(\sum_{j=1}^{n} x_j^2) c_1 \;+\; 2(\sum_{j=1}^{n} x_j) c_0 \;- 2\sum_{j=1}^{n} x_j y_j && (10.23)\\
\partial\varepsilon/\partial c_0 &= \sum_{j=1}^{n} 2(c_1 x_j + c_0 \;- y_j) = 0 && (10.24)\\
&= 2(\sum_{j=1}^{n} x_j) c_1 \;+\; 2\sum_{j=1}^{n} c_0 \;- 2\sum_{j=1}^{n} y_j && (10.25)
\end{aligned}$$

These equations are nicely represented in matrix form. The parameters of the best line are found by solving the equations. The general case representing an arbitrary polynomial fit results in a highly patterned set of equations called the *normal equations.*

$$\begin{bmatrix} \sum_{j=1}^{n} x_j^2 & \sum_{j=1}^{n} x_j \\ \sum_{j=1}^{n} x_j & \sum_{j=1}^{n} 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{n} x_j y_j \\ \sum_{j=1}^{n} y_j \end{bmatrix} \tag{10.26}$$

Exercise 13 Fitting a line to 3 points

Using Equation 10.26, compute the parameters c_1 and c_0 of the best line through the points (0, -7), (2, -1) and (4, 5).

Exercise 14 normal equations

(a) Derive the matrix form for the equations constraining the 4 parameters for fitting a cubic polynomial $c_3x^3 + c_2x^2 + c_1x + c_0$ to observed data $(x_j, y_j), j = 1, n$. (b) From the pattern of the matrix elements, predict the matrix form for fitting a polynomial of degree four.

Empirical Interpretation of the Error

Empirical interpetation of the error and individual errors is often straightforward in machine vision problems. For example, we might accept the fit if all observed points are within a pixel or two of the model. In a controlled 2D imaging environment, one could image many straight-sided objects and study the variation of detected edge points off the ideal line. If individual points are far from the fitted line (these are called *outliers*), they could indicate feature detection error, an actual defect in the object, or that a different object or model exists. In these cases, it is appropriate to delete the outliers from the set of observations and repeat the fit so that the model is not pulled off by points which it should not model. All the original points can still be interpreted relative to the updated model. If the model fitting is being used for curve segmentation, it is typically the extreme points that are deleted, because they are actually part of a different shaped object or part.

*Statistical Interpretation of the Error

Error can be interpreted relative to a formal statistical hypothesis. The common assumption is that the observed value of y_j is just the model value $f(x_j)$ plus (Gaussian) noise from the normal distribution $N(0, \sigma)$, where σ is known from the analysis of measurement error, which could be done empirically as above. It is also assumed that the noise in any individual observation j is independent of the noise in any other observation k. It follows that the variable $S_{sq} = \sum_{j=1}^{n}((f(x_j) - y_j)^2)/\sigma^2)$ is χ^2 distributed, so its likelihood can be determined by formula or table lookup. The number of degrees of freedom is $n - 2$ for the straight line fit since two parameters are estimated from the n observations. If 95% of the χ^2 distribution is below our observed S_{sq}, then perhaps we should reject the hypothesis that this model fits the data. Other confidence levels can be used. The χ^2 test is not only useful for accepting/rejecting a given hypothsis, but it is also useful for selecting the most likely model from a set of competing alternatives. For example, a parabolic model may compete with the straight line model. Note that in this case, the parabolic model $y = c_2x^2 + c_1x + c_0$ has three parameters so the χ^2 distribution would have $n - 3$ degrees of freedom.

Intuitively, we should not be too comfortable in assuming that error in observation j is independent of the error in observations $j - 1$ or $j + 1$. For example, an errant manufacturing process might distort an entire neighborhood of points from the ideal model. The independence hypothesis can be tested using a *run-of-signs* test, which can detect systematic bias in the error, which in turn indicates that a different shape model will fit better. If the noise is truly random, then the signs of the error should be random and hence fluctuate frequently. Figure 10.25 (right) shows a biased linear fit competing with an unbiased circular fit. The signs of the errors indicate that the linear fit is biased. Consult the references at the end of the chapter for more reading on statistical hypothesis-testing for evaluating fit quality.

Exercise 15 fitting a planar equation to 3D points

(a) Solve for the parameters a, b, c of the model $z = f(x, y) = ax + by + c$ of the least squares plane through the five surface points (20, 10, 130), (25, 20, 130), (30, 15, 145), (25, 10, 140), (30, 20, 140). (b) Repeat part (a) after adding a random variation to each of the three coordinates of each of the five points. Flip a coin to obtain the variation: if the coin shows heads, then add 1, if it shows tails then subtract 1.

Exercise 16 Prewitt operator is "optimal"

Show that the Prewitt gradient operator from Chapter 5 can be obtained by fitting the least squares plane through the 3x3 neighborhood of the intensity function. To compute the gradient at $I[x, y]$, fit the nine points $(x + \Delta x, y + \Delta y, I[x + \Delta x, y + \Delta y])$ where Δx and Δy range through -1, 0, +1. Having the planar model $z = ax + by + c$ that best fits the intensity surface, show that using the two Prewitt masks actually compute a and b.

Problems in Fitting

It is important to consider several kinds of problems in fitting.

outliers Since every observation affects the RMS error, a large number of outliers may render the fit worthless: the initial fit may be so far off the ideal that it is impossible to identify and delete the real outliers. Methods of *robust statistics* can be applied in such cases: consult the Boyer *et al* (1994) reference cited at the end of the chapter.

error definition The mathematical definition of error as a difference along the y-axis is not a true geometric distance; thus the least squares fit does not necessarily yield a curve or surface that best approximates the data in geometric space. The rightmost data point at the right of Figure 10.25 illustrates this problem – that point is geometrically very close to the circle, but the functional difference along the y-axis is rather large. This effect is even more pronounced when complex surfaces are fit to 3D points. While geometric distance is usually more meaningful than functional difference, it is not always easy to compute. In the case of fitting a straight line, when the line gets near to vertical, it is better to use the best axis computation given in Chapter 3 rather than the least squares method presented here. The best axis computation is formulated based on minimizing the geometric distances between line and points.

nonlinear optimization Sometimes, a closed form solution to the model parameters is not available. The error criteria can still be optimized, however, by using a technique that **searches parameter space** for the best parameters. Hill-climbing, gradient-based search, or even exhaustive search can be used for optimization. See the works by Chen and Medioni and Ponce *et al*, which address this and the previous issue.

high dimensionality When the dimensionality of the data and/or the number of model parameters is high, both empirical and statistical interpretation of a fit can be difficult. Moreover, if a search technique is used to find parameters it may not even be known whether or not these parameters are optimal or just result from a local minima of the error criteria.

fit constraints Sometimes, the model being fit must satisfy additional constraints. For example, we may need to find the best line through observations that is also perpendicular to another line. Techniques for constrained optimization can be found in the references.

Segmenting Curves via Fitting

The model fitting method and theory presented above assumes that both a model hypothesis and set of observations are given. Boundary tracking can be done in order to obtain long strings of boundary points which can then be segmented as follows. First, high curvature points or cusps can be detected in the boundary sequence in order to segment it. Then, model hypotheses can be tested against the segments between breakpoints. The result of this process is a set of curve segments and the mathematical model and parameters which characterize the shape of each segment. An alternative method is to use the model-fitting process in order to segment the original boundary curve. In the first stage, each subsequence of k consecutive points is fit with each model. The χ^2 value of the fit is stored in a set with each acceptable fit. The second stage attempts to extend acceptable fits by repetitively adding another endpoint to the subsequence. Fitted segments are grown until addition of an endpoint decreases the χ^2 value of the fit. The result of this process is a set of possibly overlapping subsequences, each with a model and the χ^2 value of the model fit. This set is then passed to higher level processes which can construct domain objects from the available detected parts. This process is similar in spirit to the region-grower described in Section 10.1.2, which has been successfully used to grow line segments using the direction estimate at each edge pixel as the critical property instead of the grey tone property used in growing regions.

10.5 Identifying Higher-level Structure

Analysis of images often requires the combination of segments. For example, quadrilateral regions and straight edge segments might be combined as evidence of a building or intersecting edge segments might accurately define the corner of a building, or a green region inside a blue region might provide evidence of an island. The methods for combining segments are limitless. Below, we look at just two general cases of combining edge segments to form more informative structures: these are the *ribbon* and the *corner*.

10.5.1 Ribbons

A very general type of image segment is the *ribbon*. Ribbons are commonly produced by imaging elongated objects in 2D or in 3D; for example, by imaging a conduction path on a printed circuit board, the door of a house, a pen on a table, or a road through fields. In these examples, the sides of the ribbons are approximately parallel to each other, but not necessarily straight. Although we limit ourselves to straight-sided ribbons below, ribbons can have more general shape, such as that of a wine bottle or ornate lampost, where the shape of the silhouette is some complex curve with reflective symmetry relative to the axis of the ribbon. An electric cord, a rope, a meandering stream or road each produce a ribbon in an image, as will the shadow of a rope or lampost. Chapter 14 discusses 3D object parts called *generalized cylinders*, which produce ribbons when viewed. At the right in Figure 10.16 is a symbol that is well represented by four ribbons, two of which are highly curved. We leave extraction of general ribbons to future study and concentrate on those with straight sides.

4 DEFINITION *A ribbon is an elongated region that is approximately symmetrical about its major axis. Often, but not always, the edges of a ribbon contrast symmetrically with its background.*

Figure 10.26 shows how the Hough Transform can be extended slightly to encode the gradient direction across an edge in addition to its orientation and location. As was shown in Chapter 5, and earlier in this chapter, gradient direction θ at a pixel $[r, c]$ that has significant gradient magnitude can be computed in the interval $[0, 2\Pi)$ using operators such as the Sobel operator. The vector from the image origin to the pixel is $[r, c]$: we project this vector onto the unit vector in the direction θ to obtain a signed distance d.

$$d \; = \; [r, c] \circ [-sin\theta, cos\theta] \; = \; -r \; sin\theta \; + \; c \; cos\theta \tag{10.27}$$

A positive value for d is the same as that obtained in the usual polar coordinate representation for pixel $[r, c]$. However, a negative d results when the direction from the origin to the edge is opposite to the gradient direction: this will result in two separate clusters for every "line" on a checkboard for example. Figure 10.26 illustrates this idea. Consider the edge P_1P_2 in the figure. Pixels along this edge should all have gradient direction approximately 315 degrees. The perpendicular direction from the origin to P_1P_2 is in the same direction, so pixels along P_1P_2 will transform to (approximately) $[d_1, 315°]$ in the Hough parameter space. Pixels along line segment P_2P_3 however, have a gradient direction of 135 degrees, which is opposite to the direction of the perpendicular from the origin to P_2P_3. Thus, pixels along segment P_2P_3 will transform to (approximately) $[-d_1, 135°]$.

Exercise 17

Figure 10.27 shows a dark ring on a light background centered at the image origin. Sketch the parameter space that would result when such an image is transformed using the Hough Transform as shown in Figure 10.26

Detecting Straight Ribbons

By using the Hough parameters along with the point lists obtained by Algorithm *accumulate_lines*, more complex image structure can be detected. Two edges whose directions differ by 180° provide evidence of a possible ribbon. If in addition, the point lists are located near each other, then there is evidence of a larger linear feature that reverses gradient, such as a road through farm fields as in Figure 10.17.

Figure 10.28 shows an image of part of a white house containing a downspout. The image was taken in strong sunlight and this resulted in hard shadows. By using a gradient operator and then collecting pixels on edge segments using *accumulate_lines*, there is strong evidence of a bright ribbon on a dark background corresponding to the downspout (sides are AB and ED). The shadow of the downspout s also creates evidence of a dark ribbon on a bright background.

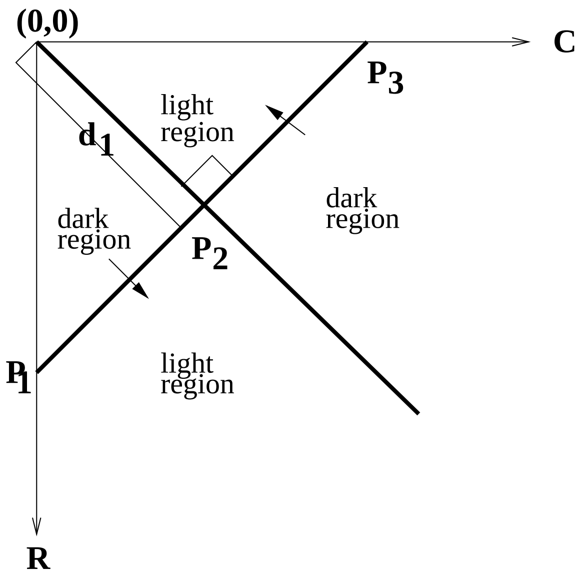


Figure 10.26: The Hough transform can encode the location and orientation of an edge and its gradient direction. The transition from a dark region to a lighter one gives the opposite gradient direction from the transition from the lighter region to the darker one, along the same image line.

Exercise 18

Write a computer program to study the use of the Hough Transform to detect ribbons. (a) Use the Sobel operator to extract the gradient magnitude and direction at all pixels. Then transform only pixels of high magnitude. (b) Detect any clusters in $[d, \theta]$-space. (c) Detect pairs of clusters, $([d_1, \theta_1], [d_2, \theta_2])$, where θ_1 and θ_2 are π apart. (d) Delete pairs that are not approximately symmetrical across an axis between them.

10.5.2 Detecting Corners

Significant region corners can be detected by finding pairs of detected edge segments E_1 and E_2 in the following relationship.

1. Lines fit to edge point sets E_1 and E_2 intersect at point $[u, v]$ in the real image coordinate space.
2. Point $[u, v]$ is close to extreme points of both sets E_1 and E_2.
3. The gradient directions of E_1 and E_2 are symmetric about their axis of symmetry.

This definition models only corners of type 'L': constraint (2) rules outs those of type 'T', 'X' and 'Y'. The computed intersection $[u, v]$ will have *subpixel accuracy*. Figure 10.29 sketches the geometry of a corner structure. Edge segments can be identified intitially by using the Hough transform or by boundary following and line fitting or by any other appropriate algorithm. For each pair $([d_1, \theta_1], [d_2, \theta_2])$ satisfying the above criteria, add the quad $([d_1, \theta_1], [d_2, \theta_2], [u, v], \alpha)$ to a set of candidate corners. The angle α is formed at the corner.

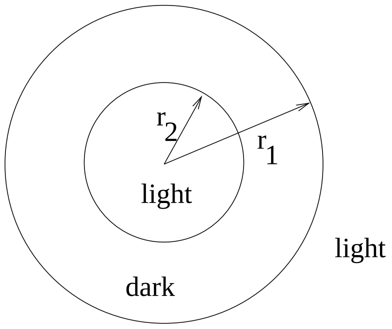


Figure 10.27: Dark ring centered at the origin on light background. Dark region is outside the smaller circle and inside the larger one.

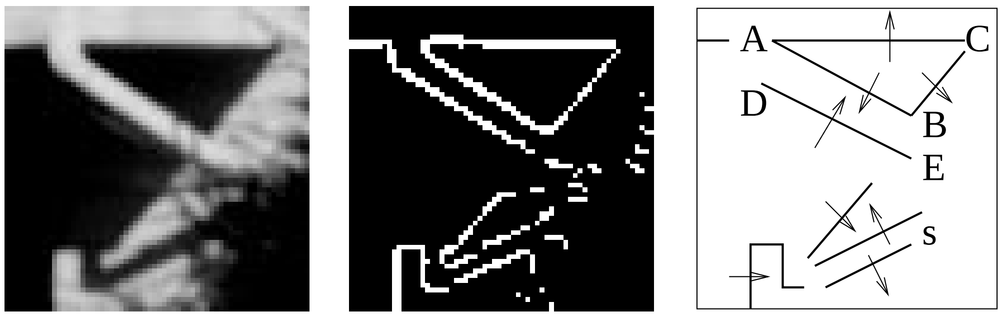


Figure 10.28: (Left) Region of an image of a house showing a downspout and strong shadows; (center) the highest 10% gradient magnitudes computed by the Prewitt 3x3 operator; (right) sketch of ribbons and corners evident.

This set of corner features can be used for building higher level descriptions, or can be used directly in image matching or warping methods as shown in Chapter 11.

Several corners can easily be extracted from the blocks image in Figure 10.2; however, most of them are due to viewpoint dependent occlusions of one object crease by another and not by the joining of two actual 3D object creases. Four actual corners are evident for the top of the arch. The corners of triangle ABC in Figure 10.28 are all artifacts of the lighting and viewpoint. We conclude this discussion by making the point that although representations using edge segments are often used in specific problem domains they may be highly ambiguous in when used in general. Usually, problem-specific knowledge is needed in order to interpret higher-level structure.

10.6 Segmentation using Motion Coherence

As we have seen, motion is important for determining scene content and action. Chapter 9 presented methods for detecting change in a scene and for tracking motion over many frames.

Exercise 19

Describe how to change the ribbon detection algorithm so that (a) it only detects ribbons that are nearly vertical, (b) it detects ribbons that are no wider than W.

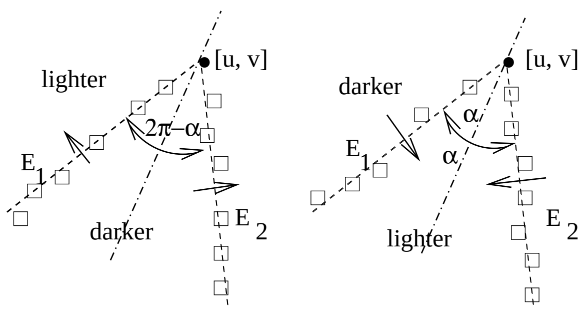


Figure 10.29: Corners are detected as pairs of detected edge segments appropriately related.

10.6.1 Boundaries in Space-Time

The contours of moving objects can be identified by using both spatial and temporal contrast. Our previous examples have used only spatial contrast of some property such as intensity or texture in a single image. Spatial and temporal gradients can be computed and combined if we have two images $\mathbf{I}[x, y, t]$ and $\mathbf{I}[x, y, t + \delta t]$ of the scene. We can define a *spatio-temporal gradient magnitude* as the product of the spatial gradient magnitude and the temporal gradient magnitude as in Equation 10.28. Once an image **STG**[] is computed, it is amenable to all the contour extraction methods already discussed. The contours that are extracted will be the boundaries of moving objects and not static ones, however.

$$\mathbf{STG}[x, y, t] \;=\; \mathbf{Mag}[x, y, t]\;(|\mathbf{I}[x, y, t] - \mathbf{I}[x, y, t + \delta t]|) \qquad (10.28)$$

10.6.2 Aggregrating Motion Trajectories

Assume that motion vectors are computed across two frames of an image sequence. This can be done using special interest points or regions as described in Chapter 9. Region segmentation can be performed on the motion vectors by clustering according to image position, speed, and direction as shown in Figure 10.30. Clustering should be very tight for a translating object, because points of the object should have the same velocity. Through more complex analysis, objects that rotate and translate can also be detected.

Figure 10.31 shows processing from an application where the purpose of motion is communication: the goal is to input to a machine via American Sign Language (ASL). The figure shows only a sample of frames from a sequence representing about two seconds of gesturing by the human signer. The results shown in Figure 10.31 were produced using both color segmentation within a single frame and motion segmentation across pairs of frames. A sketch of steps of an algorithm is given in Algorithm 10: for details of the actual algorithm producing Figure 10.31, consult the reference by Yang and Ahuja (1999). The

Exercise 20

Given two lines parameterized by $([d_1, \theta_1], [d_2, \theta_2])$, (a) derive a formula for their intersection point $[x, y]$ and (b) derive a formula for their axis of symmetry $[d_a, \theta_a]$.

Exercise 21

Obtain two successive images of a scene with moving objects and compute a spatio-temporal image from them using Equation 10.28. (Two frames from a Motion JPEG video would be good. Or, one could digitize some dark cutouts on a flatbed scanner, moving them slightly for the second image.)

first several steps of the algorithm can be generally applied to many different kinds of sequences. Color segmentation is done for each image and the segments are matched across frames. These matches are used to guide the computation of a dense motion field for each pair of consecutive images, which is then segmented to derive motion trajectories at the individual pixel level. This motion field is then segmented into regions of uniform motion. Only at this point do we add domain knowledge to identify hands and face: a skin color model, as seen in Chapter 6, identifies skin regions, the largest of which is taken to be the face. The center of each hand region is then tracked over all frames: the two trajectories can then be used to recognize the sign made. Yang and Ahuja reported recognition rates of over 90% in experiments with many instances of a set of 40 American Sign Language signs.

Exercise 22

Describe how Algorithm 10 can be modified to make it simpler and faster by specializing all of its steps for the ASL application.

10.7 References

Segmentation is one of the oldest, and still unsolved, areas of computer vision. The 1985 survey by Haralick and Shapiro gives a good overview of the early work, most of which worked with gray-tone images. The first useful segmentation work with natural color images was done by Ohlander, Price, and Reddy in 1978. Only in recent years has the area become fruitful again. The work of Shi and Malik on normalized cuts, starting in 1997, can be credited with being the catalyst for newer work in which segmentations of arbitrary color images from large image collections is being undertaken. In line-drawing analysis, Freeman first proposed his chain code in the 1960's; his 1974 article discusses its use. While the Hough Transform was published only as a patent, it was popularized and expanded by Duda and Hart, and its use is nicely illustrated for line segments in O'Gorman and Clowes' 1976 paper and for circles in Kimme, Ballard, and Sklansky's 1975 work. The Burns line finder, published ten years later is an improvement to the technique to make it more robust and reliable. Samet's 1990 book on spatial data structures is an excellent reference on quad trees.

Bowyer *et al* (1994) show how to use robust statistics to fit models to data for segmentation. Any anticipated model can be fit to all of the image: robust fitting can eliminate a huge number of outliers, resulting in a segment of the image where the particular model fits

Exercise 23

Suppose we have two motion trajectories $\mathbf{P}_j, j = 1, N$ and $\mathbf{Q}_k, k = 1, M$, where $\mathbf{P}_j$ and $\mathbf{Q}_k$ are points in 2D in proper time sequence. Devise an algorithm for matching two such trajectories, such that 1.0 is output when both trajectories are identical and 0.0 is output when they are very different. Note the M and N may not be equal.

```
Input a video sequence of a person signing in ASL.
Output motion trajectories of the two palms.

 1. Segment each frame $I_t$ of the sequence into regions using color.
 2. Match the regions of each pair of images $(I_t, I_{t+1})$ by color and neighborhood.
 3. Compute the affine transformation matching each region of $I_t$ to the corresponding
    region of $I_{t+1}$.
 4. Use the transformation matching regions to guide the computation of motion vectors
    for individual pixels.
 5. Segment the motion field derived above using motion coherence and image location.
 6. Identify two hand regions and the face region using a skin color model.
 7. Merge adjacent skin colored regions that were fragmented previously.
 8. Find an ellipse approximating each hand and the face.
 9. Create motion trajectories by tracking each ellipse center over the entire sequence.
10. (Recognize the gesture using the trajectories of the two hands.)
```

Algorithm 10: Algorithm using color and motion to track ASL gestures (Motivated by Yang and Ahuja (1999)).

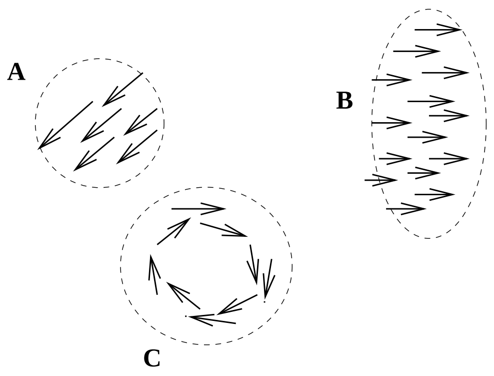


Figure 10.30: Aggregation of vectors from the motion field via compatible location, velocity and direction: translating objects (A,B) are easier to detect than rotating objects (C).

well. An image can be said to be segmented when all anticipated models have been fitted: the segments are comprised of the points that have been fitted.

1. K. Boyer, K. Mirza and G. Ganguly (1994), " The Robust Sequential Estimator: A General Approach and its Application to Surface Organization in Range Data", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 16, No. 10 (Oct 1994), pp. 987-1001.

2. J. R. Burns, A. R. Hanson, and E. M. Riseman, "Extracting Straight Lines," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, 1986, pp. 425-455.

3. Y. Chen and G. Medioni, *Surface Description of Complex Object from Multiple Range Images,* **Proceedings of IEEE Conference on Computer Vision and Pattern Recognition**, Seattle, WA (June 1994)513-518.

4. R. O. Duda and P. E. Hart, "Use of the Hough Transform to Detect Lines and Curves in Pictures," *Communications of the ACM*, Vol. 15, 1972, pp. 11-15.

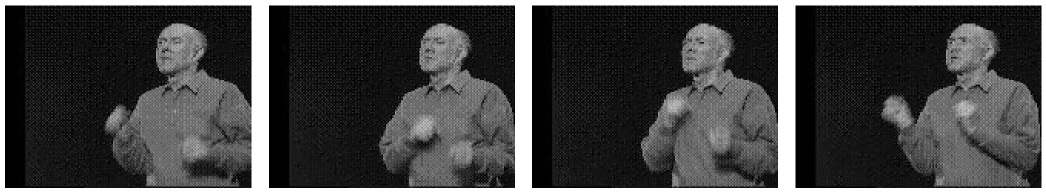
5. H. Freeman, "Computer Processing of Line-Drawing Images," *Computing Surveys*, Vol. 6, 1974, pp. 57-97.

6. R. M. Haralick and L. G. Shapiro, "Image Segmentation Techniques," *Computer Vision, Graphics, and Image Processing,* Vol. 29, No. 1, January 1985, pp. 100-132.

7. C. Kimme, D. Ballard, and J. Sklansky, "Finding Circles by an Array of Accumulators," *Communications of the ACM,* VOl. 18, 1975, pp. 120-122.

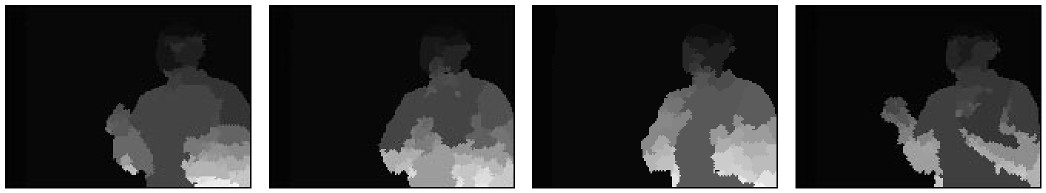
8. F. O'Gorman and M. B. Clowes, "Finding Pciture Edges through Collinearity of Feature Points, " *IEEE Transactions on Computers*, VOl. C-25, 1976, pp. 449-454.

9. R. Ohlander, K. Price, and D. R. Reddy, "Picture Segmentation Using a Recursive Region Splitting Method," *Computer Graphics and Image Processing*, Vol. 8, 1978, pp. 313-333.



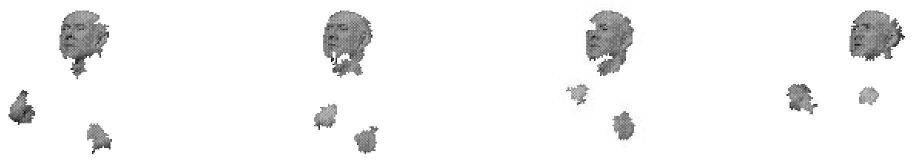
(a) frame 14 (b) frame 16 (c) frame 19 (d) frame 22

(I) Four video frames of a 55-frame sequence of ASL sign "cheerleader".



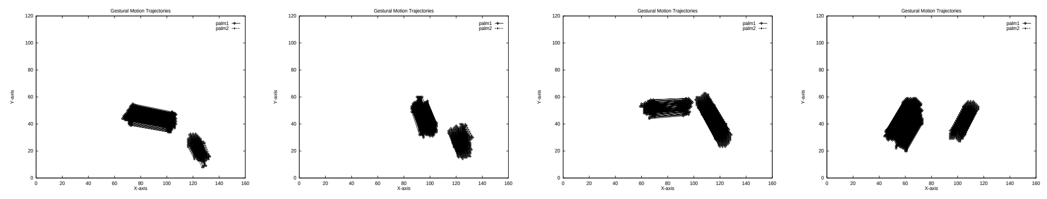
(e) frame 14 (f) frame 16 (g) frame 19 (h) frame 22

(II) Motion segmentation of the image sequence "cheerleader".
(pixels of the same motion region are displayed with same gray level and different regions are displayed with different gray levels)



(i) frame 14 (j) frame 16 (k) frame 19 (l) frame 22

(III) Extracted head and palm regions from image sequence "cheerleader".



(m) #14-#16 (n) #16-#19 (o) #19-#22 (p) #22-#25

(IV) Extracted gestural motion trajectories from segments of ASL sign "cheerleader" (since all pixel trajectories are shown, they form a thick blob)

Figure 10.31: Extraction of motion trajectories from image sequence. (I) Sample frames from video sequence; (II) frames segmented using motion; (III) head and palm regions extracted by color and size; (IV) motion trajectories for points on the palms. (Figures courtesy of Ming-Hsuan Yang and Narendra Ahuja.)

10. K. Rao, PhD Thesis

11. H. Samet, *Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

12. J. Shi and J. Malik (1997), *Normalized Cuts and Image Segmentation*, *IEEE Conference on Computer Vision and Pattern Recognition*, 1997, pp. 731-737.

13. S. Sullivan, L. Sandford and J. Ponce (1994), *Using Geometric Distance for 3D Object Modeling and Recognition*, **IEEE Trans. on Pattern Analysis and Machine Intelligence**, Vol. 16, No. 12 (Dec 1994)1183-1196.

14. M.-H. Yang and N. Ahuja (1999), *Recognizing Hand Gesture Using Motion Trajectories*, **Proceedings IEEE Conf. on Computer Vison and Pattern Recognition 1999**, Ft. Collins, CO (23-25 June 99)466-472.