Chapter 6

Contours

Edges must be linked into a representation for a region boundary. This representation is called a contour. The contour can be open or closed. Closed contours correspond to region boundaries, and the pixels in the region may be found by a filling algorithm. An open contour may be part of a region boundary. Gaps can occur in a region boundary because the contrast between regions may not be enough to allow the edges along the boundary to be found by an edge detector. The edge detection threshold may have been set too high, or the contrast along some portion of the boundary may be so weak relative to other areas of the image that no single threshold works everywhere in the image. Open contours also occur when line fragments are linked together—for example, when line fragments are linked along a stroke in a drawing or sample of handwriting.

A contour may be represented as an ordered list of edges or by a curve. A curve is a mathematical model for a contour. Examples of curves include line segments and cubic splines. There are several criteria for a good contour representation:

Efficiency: The contour should be a simple, compact representation.

Accuracy: The contour should accurately fit the image features.

Effectiveness: The contour should be suitable for the operations to be performed in later stages of the application.

The accuracy of the contour representation is determined by the form of curve used to model the contour, by the performance of the curve fit-

ting algorithm, and by the accuracy of the estimates of edge location. The simplest representation of a contour is an ordered list of its edges. This representation is as accurate as the location estimates for the edges, but is the least compact representation and may not provide an effective representation for subsequent image analysis. Fitting the appropriate curve model to the edges increases accuracy, since errors in edge location are reduced through averaging, and it increases efficiency by providing a more appropriate and more compact representation for subsequent operations. For example, a set of edges that lie along a line can be represented most efficiently by fitting a line to the edges. This representation simplifies later calculations, such as determining the orientation or length of the line, and increases the accuracy, since the mean squared error between the estimated line and the true line will be smaller than the error between the true line and any of the edges.

The first section in this chapter presents the elementary differential geometry of curves in the plane. The second section gives a collection of techniques for calculating contour properties such as length, tangent, and curvature from the list of edges, without fitting a curve model to the edges. The remaining sections cover curve models and techniques for fitting the models to contours.

Before proceeding, some terms must be defined. A curve *interpolates* a list of points if the curve passes through the points. *Approximation* is fitting a curve to a list of points with the curve passing close to the points, but not necessarily passing exactly through the points. In the following sections, we will begin by assuming that the edges provided by an edge detection algorithm are exact and will fit curves to the edge points using interpolation methods. The edges provided by edge detection applied to real images will not be exact. There will be some error in the estimated location of the edge. Later sections will present methods for curve approximation.

Definition 6.1 *An edge list is an ordered set of edge points or fragments.*

Definition 6.2 *A contour is an edge list or the curve that has been used to represent the edge list.*

Definition 6.3 *A boundary is the closed contour that surrounds a region.*

In this chapter, the term *edges* will usually refer to edge points. The edge orientation is not used by most curve fitting algorithms. In the few cases where the algorithm does use the edge orientation, it will be clear from the context that the term *edges* refers to edge fragments.

6.1 Geometry of Curves

Planar curves can be represented in three different ways: the explicit form $y = f(x)$, the implicit form $f(x, y) = 0$, or the parametric form $(x(u), y(u))$ for some parameter u. The explicit form is rarely used in machine vision since a curve in the x–y plane can twist around in such a way that there can be more than one point on the curve for a given x.

The parametric form of a curve uses two functions, $x(u)$ and $y(u)$, of a parameter u to specify the point along the curve from the starting point of the curve at $\mathbf{p}_1 = (x(u_1), y(u_1))$ to the end point $\mathbf{p}_2 = (x(u_2), y(u_2))$. The length of a curve is given by the arc length:

$$\int_{u_1}^{u_2} \sqrt{\left(\frac{dx}{du}\right)^2 + \left(\frac{dy}{du}\right)^2}\, du. \tag{6.1}$$

The unit tangent vector is

$$\mathbf{t}(u) = \frac{\mathbf{p}'(u)}{|\mathbf{p}'(u)|}, \tag{6.2}$$

where $\mathbf{p}(u) = (x(u), y(u))$. The curvature of the curve is the derivative of the tangent: $\mathbf{n}(u) = \mathbf{p}''(u)$.

Consider three points along the curve: $\mathbf{p}(u + \Delta)$, $\mathbf{p}(u)$, and $\mathbf{p}(u - \Delta)$. Imagine a circle passing through these three points, which uniquely determine the circle. In the limit as $\Delta \to 0$, this circle is the osculating circle. The osculating circle touches the curve at $\mathbf{p}(u)$, and the center of the circle lies along the line containing the normal to the curve. The curvature is the inverse of the radius of the osculating circle.

6.2 Digital Curves

In this section, we present a set of algorithms for computing the elements of curve geometry, such as contour length, tangent orientation, and curvature, from the list of edge points. Slope and curvature are difficult to compute precisely in the digital domain, since the angle between neighboring pixels is quantized to 45° increments.

The basic idea is to estimate the tangent orientation using edge points that are not adjacent in the edge list. This allows a larger set of possible

tangent orientations. Let $\mathbf{p}_i = (x_i, y_i)$ be the coordinates of edge i in the edge list. The k-slope is the (angle) direction vector between points that are k edges apart. The left k-slope is the direction from $\mathbf{p}_{i-k}$ to $\mathbf{p}_i$, and the right k-slope is the direction from $\mathbf{p}_i$ to $\mathbf{p}_{i+k}$. The k-curvature is the difference between the left and right k-slopes.

Suppose that there are n edge points $(x_1, y_1), \ldots, (x_n, y_n)$ in the edge list. The length of a digital curve can be approximated by adding the lengths of the individual segments between pixels:

$$S = \sum_{i=2}^{n} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}. \tag{6.3}$$

A good approximation is obtained by traversing the edge list and adding 2 along sides and 3 along diagonals, and dividing the final sum by 2. The distance between end points of a contour is

$$D = \sqrt{(y_n - y_1)^2 + (x_n - x_1)^2}. \tag{6.4}$$

6.2.1 Chain Codes

Chain codes are a notation for recording the list of edge points along a contour. The chain code specifies the direction of a contour at each edge in the edge list. Directions are quantized into one of eight directions, as shown in Figure 6.1. Starting at the first edge in the list and going clockwise around the contour, the direction to the next edge is specified using one of the eight chain codes. The direction is the chain code for the 8-neighbor of the edge. The chain code represents an edge list by the coordinates of the first edge and the list of chain codes leading to subsequent edges. A curve and its chain code are shown in Figure 6.2.

The chain code has some attractive properties. Rotation of an object by 45° can be easily implemented. If an object is rotated by $n \times 45°$, then the code for the rotated object is obtained by adding $n \bmod 8$ to the original code. The derivative of the chain code, also called *difference code*, obtained by using first difference, is a rotation-invariant boundary description. Some other characteristics of a region, such as area and corners, may be directly computed using the chain code. The limitation of this representation is

2	3	4
1	·	5
8	7	6

Figure 6.1: The chain codes for representing the directions between linked edge points.

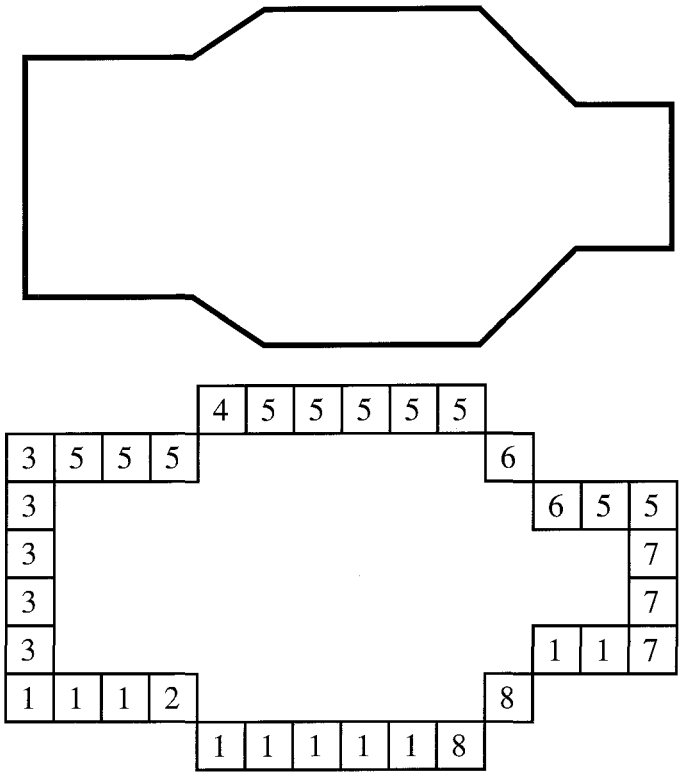

Figure 6.2: A curve and its chain code.

the limited set of directions used to represent the tangent at a point. This limitation can be removed by using one of the curve representations presented in the following sections. Once a curve has been fitted to the list of edges, any of the geometric quantities presented in Section 6.1 can be computed from the mathematical formula for the curve.

6.2.2 Slope Representation

The slope representation of a contour, also called the Ψ-s plot, is like a continuous version of the chain code. We want to represent a contour using arbitrary tangent directions, rather than the limited set of tangent directions allowed by the chain code. Suppose that we start at the beginning of the edge list and compute the tangent and arc length using the formulas presented for digital curves. We may plot the tangent Ψ versus arc length s to obtain a representation for the contour in the Ψ-s space. The Ψ-s plot is a representation of the shape of the contour. For example, a contour that consists of line segments and circular arcs will look like a sequence of line segments in the Ψ-s plot. Horizontal line segments in the Ψ-s plot correspond to line segments in the contour; line segments at other orientations in the Ψ-s plot correspond to circular arcs. Portions of the Ψ-s plot that are not straight lines correspond to other curve primitives.

The contour may be split into straight lines and circular arcs by segmenting the Ψ-s plot into straight lines. This method has been used by many researchers, and there are several versions of this approach for splitting a contour into segments.

One may use the Ψ-s plot as a compact description of the shape of the original contour. In Figure 6.3, we show a contour and its Ψ-s plot. For a closed contour, the Ψ-s plot is periodic.

6.2.3 Slope Density Function

The slope density function is the histogram of the slopes (tangent angles) along a contour. This can be a useful descriptor for recognition. Correlating the slope density function of a model contour with the slope density function for a contour extracted from an image allows the orientation of the object to be determined. This also provides a means for object recognition.

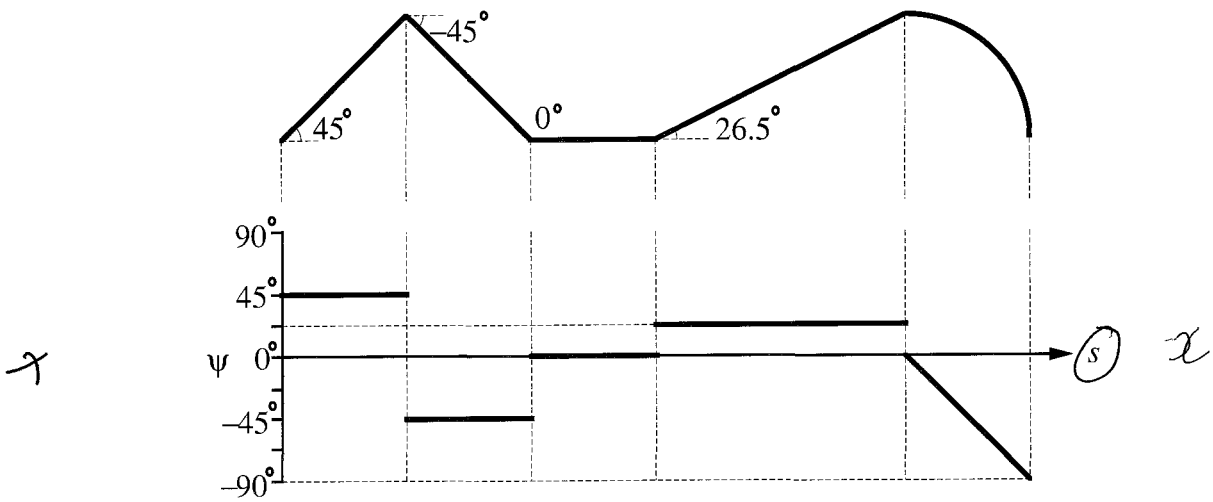

Figure 6.3: Slope representations of a contour.

6.3 Curve Fitting

The rest of this chapter will cover four curve models and the methods for fitting the models to edge points. The models include:

- Line segments
- Circular arcs
- Conic sections
- Cubic splines

Any fitting algorithm must address two questions:

1. What method is used to fit the curve to the edges?
2. How is the closeness of the fit measured?

Sections 6.4 through 6.7 will cover techniques for fitting curve models to edges with the assumption that the edge locations are sufficiently accurate that selected edge points can be used to determine the fit. Section 6.8 will present successively more powerful methods that can handle errors in the edge locations.

Let d_i be the distance of edge point i from a line. There are several measures of the goodness of fit of a curve to the candidate edge points. All of them depend on the error between the fitted curve and the candidate points forming the curve. Some commonly used methods follow.

Maximum absolute error measures how much the points deviate from the curve in the worst case:

$$\text{MAE} = \max_i |d_i| \tag{6.5}$$

Mean squared error gives an overall measure of the deviation of the curve from the edge points:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} d_i^2 \tag{6.6}$$

Normalized maximum error is the ratio of the maximum absolute error to the length of the curve:

$$\varepsilon = \frac{\max_i |d_i|}{S} \tag{6.7}$$

Number of sign changes in the error is a good indicator of the appropriateness of the curve as a model for the edges in the contour.

Ratio of curve length to end point distance is a good measure of the complexity of the curve.

The normalized maximum error provides a unitless measure of error independent of the length of the curve. In other words, a given amount of deviation from a curve may be equally significant, in some applications, as twice as much deviation from a curve that is twice as long. If the curve model is a line segment, then it is not necessary to compute the arc length; the distance D between the end points can be used:

$$D = \sqrt{(y_n - y_1)^2 + (x_n - x_1)^2}. \tag{6.8}$$

Sign changes are a very useful indication of goodness of fit. Fit a list of edge points with a straight line and examine the number of sign changes. One sign change indicates that the list of edges may be modeled by a line segment, two sign changes indicate that the edges should be modeled by a quadratic curve, three sign changes indicate a cubic curve, and so on. Numerous sign

changes indicate that a small increase in the complexity of the curve will not improve the fit significantly. A good fit has a random pattern to the sign changes. Runs of errors of the same sign indicate a systematic error in fitting; possibly due to the wrong curve model.

In the following sections, we will use simple curve fitting methods to illustrate the use of the polyline, circular arc, conic section, and cubic spline models. Section 6.8 will present more powerful curve fitting methods, using polylines as the primary example; but, in principle, any of the models presented in the following sections could be used with any of the curve fitting methods presented in Section 6.8.

The choice of curve fitting model must be guided by the application. The use of straight line segments (polylines) is appropriate if the scene consists of straight lines and is the starting point for fitting other models. Circular arcs are a useful representation for estimating curvature, since the curve is segmented into sections with piecewise constant curvature. Conic sections provide a convenient way to represent sequences of line segments and circular arcs, as well as elliptic and hyperbolic arcs, and explicitly represent inflection points. Cubic splines are good for modeling smooth curves and do not force estimates of tangent vectors and curvature to be piecewise constant.

6.4 Polyline Representation

A polyline is a sequence of line segments joined end to end. The polyline representation for a contour fits the edge list with a sequence of line segments. The polyline interpolates a selected subset of the edge points in the edge list. The ends of each line segment are edge points in the original edge list. Each line segment models the run of contiguous edges between its end points. The points where line segments are joined are called vertices. Note that polylines are two-dimensional curves in the image plane, as are all of the curves discussed in this chapter, and the vertices are points in the image plane.

The polyline algorithm takes as input an ordered list of edge points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$. The edge point coordinates may be computed to subpixel resolution (see Section 5.7). Since line segments are fit between two edge points selected as vertices, only the coordinates of the edges that are selected as vertices need to be computed precisely.

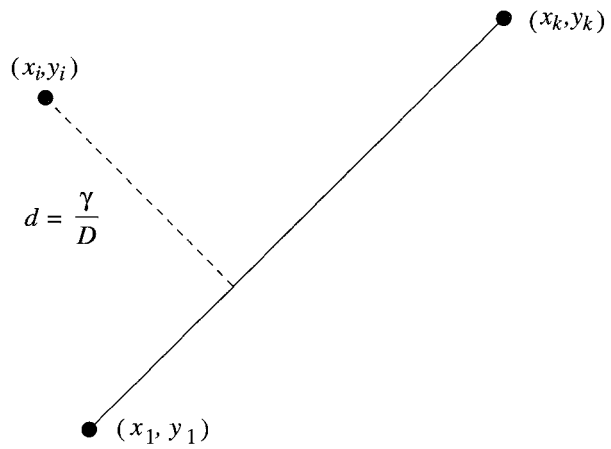

Figure 6.4: Diagram showing the perpendicular distance of a point from a line segment. The value γ is computed by plugging the coordinates (x_i, y_i) of the point into the equation for the line segment.

The formula for a line segment that approximates a list of edge points and joins the first and last edge points (x_1, y_1) and (x_k, y_k) can be derived by noting that the slope of the line between the end points is the same as the slope of the line between the first point and an arbitrary point along the line:

$$\frac{y - y_1}{x - x_1} = \frac{y_k - y_1}{x_k - x_1}. \tag{6.9}$$

Multiplying out and rearranging terms gives the implicit form for a line segment, parameterized by the coordinates of the end points:

$$x(y_1 - y_k) + y(x_k - x_1) + y_k x_1 - y_1 x_k = 0. \tag{6.10}$$

The distance of any point (x_i, y_i) from the line is $d = r/D$, where r is computed by plugging the coordinates of the point into the equation for the line segment,

$$r = x_i(y_1 - y_k) + y_i(x_k - x_1) + y_k x_1 - y_1 x_k, \tag{6.11}$$

and D is the distance between the end points. (Refer to Figure 6.4.) The sign of r can be used to compute the number of sign changes C. The normalized

distance is d/D. The normalized maximum absolute error is

$$\varepsilon = \frac{\max_i |d_i|}{D}, \tag{6.12}$$

where d_i is the distance between the line and the position of the ith edge in the edge list. The normalized maximum error is frequently used as the measure for the goodness of fit of a line segment to a set of edges. All of these formulas assume that the perpendicular projection of a point onto a line is within the line segment; that is, both on the line and between the end points of the line segment. This is the case for the situations throughout this chapter, but in other cases the formulas may need to be modified to compute the distance of the point from the nearest end point of the line segment.

There are two approaches to fitting polylines: top-down splitting and bottom-up merging.

6.4.1 Polyline Splitting

The top-down splitting algorithm recursively adds vertices, starting with an initial curve. Consider the curve shown in Figure 6.5. The initial curve is the line segment between the first and last edge points, labeled A and B. The point in the edge list that is farthest from the straight line is found. If the normalized maximum error is above a threshold, then a vertex is inserted at the edge point farthest from the line segment, labeled as point C in Figure 6.5. The splitting algorithm is recursively applied to the two new line segments and the edge list. The edge list is partitioned into two lists corresponding to the two line segments. The edge points in the list that are farthest from each segment are found, and new vertices are introduced if the points are too far from the line segments. The polyline splitting algorithm terminates when the normalized maximum error, for all edge points along the polyline, is below the threshold. This recursive procedure is very efficient. Segment splitting is also called *recursive subdivision.*

6.4.2 Segment Merging

In segment merging, edge points are added to line segments as the edge list is traversed. New segments are started when the edge points deviate too

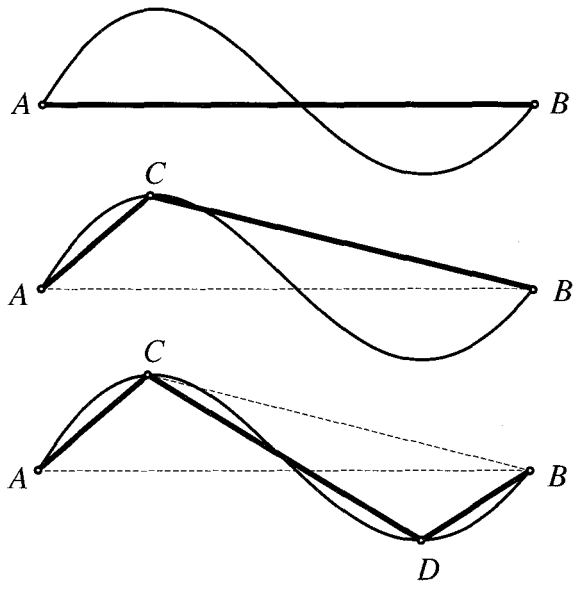

Figure 6.5: Splitting method for polylines.

far from the line segment. The merge approach is also called the bottom-up approach to polyline fitting.

There are several measures that can be used to determine if an edge point is too far from the line segment that is being formed. One method is to use sequential least-squares, which performs a least-squares fit of the line segment to the edge points and updates the parameters of the line segment incrementally as each new edge point is processed. The fitting algorithm calculates the squared residual between the line segment and the edge points. When the error exceeds a threshold, a vertex is introduced and a new segment is started from the end point of the last segment.

The tolerance band algorithm uses a different method for determining the placement of vertices. Two line segments that are parallel to the line segment approximating the edge points at a distance ϵ from the center line segment are computed. (See Figure 6.6.) The value of ϵ represents the absolute amount of deviation from the fitted line that is tolerated. Edges are added to the current line segment as long as the new edges are inside the tolerance band. The parameters of the line segment may be recomputed as new edges are added to the segment. The approximating line segment does

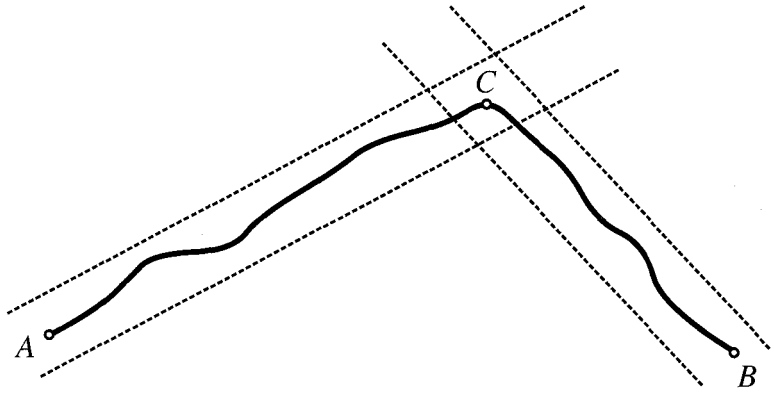

Figure 6.6: Tolerance band for fitting line segments.

not have to remain parallel to the sides of the tolerance band. The vertex at the end of the segment is the starting point for the next segment. This approach usually results in too many segments. Corner locations and angles are not accurately estimated since a vertex is not created until the algorithm has processed edges up to the boundary of the tolerance band.

6.4.3 Split and Merge

The top-down method of recursive subdivision and the bottom-up method of merging can be combined as the split and merge algorithm. Splitting and merging methods are only partially successful when used by themselves, but the accuracy of line segment approximations to a list of edges can be improved by interleaving merge and split operations. Figure 6.7 shows an example where a split followed by a merge can repair a badly placed vertex.

The basic idea is to interleave split and merge passes. After recursive subdivision, allow adjacent segments to be replaced by a single segment between the first and last end points if the new segment fits the edges with less normalized error. Note that it is necessary to use normalized error since multiple line segments will always fit a list of edges with less error than for a single line segment. After segment merging, the new segment may be split at a different point. Alternate applications of split and merge continue until no segments are merged or split.

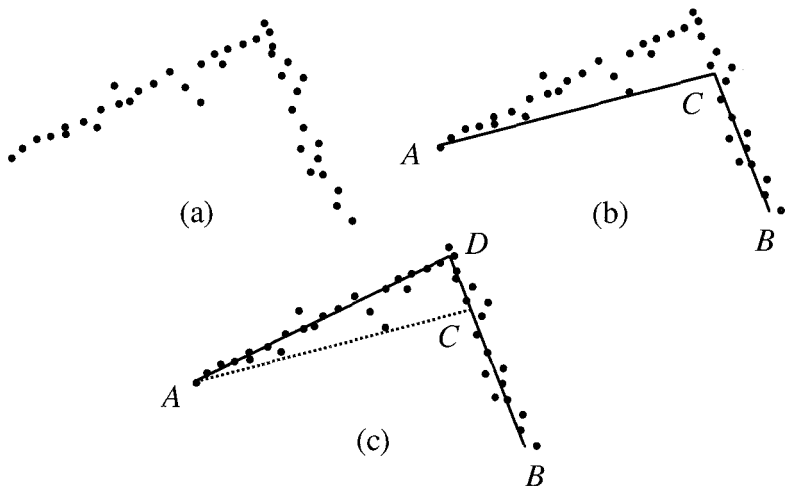

Figure 6.7: A bad corner estimate produced by a bottom-up edge merge that missed the true corner location can be repaired by split and merge passes that split the first segment at a point closer to the true corner and then merge the two segments into a single line segment.

6.4.4 Hop-Along Algorithm

The hop-along algorithm approximates a contour by a sequence of line segments, like the split and merge method described above, but works on short sublists of edges. The algorithm starts at one end of a list of edge points, grabs some fixed number of edges, and fits a line segment between the first and last edge points. If the fit is bad, the algorithm does a split at the point of maximum error and repeats with the segment closest to the beginning of the run. In other words, the algorithm falls back until it finds a good line segment approximation to some initial sequence of edges. The algorithm makes the current segment the previous segment, and continues with the remaining edge points. The algorithm also checks to see if the current segment can be merged with the previous segment. The algorithm is like a split and merge algorithm, but it does not start with the entire list of edges and does not waste time doing lots of splits. The algorithm hops along, working on modest-sized runs of edges. The algorithm is given as Algorithm 6.1.

Algorithm 6.1 Hop-Along Algorithm for Polyline Fitting

1. *Start with the first k edges from the list.*
2. *Fit a line segment between the first and last edges in the sublist.*
3. *If the normalized maximum error is too large, shorten the sublist to the point of maximum error. Return to step 2.*
4. *If the line fit succeeds, compare the orientation of the current line segment with that of the previous line segment. If the lines have similar orientations, replace the two line segments with a single line segment.*
5. *Make the current line segment the previous line segment and advance the window of edges so that there are k edges in the sublist. Return to step 2.*

The algorithm hops along, advancing the window of edges by a constant k. If the fit of a line segment to the edges is not good enough, the algorithm falls back to the point of maximum error. Since the algorithm considers only a short run of edges, it is more efficient than pure recursive subdivision or the split and merge algorithm, which would start with the entire list of edges and waste a lot of time splitting the edge list into manageable pieces.

6.5 Circular Arcs

After a list of edges is approximated by line segments, subsequences of the line segments can be replaced by circular arcs if desired. Replacing line segments by circular arcs involves fitting circular arcs through the end points of two or more line segments. In other words, circular arc fitting is done on the vertices in the polyline. Representing the contour as a sequence of line segments and circular arcs breaks the contour into sections with piecewise constant curvature. Many image analysis algorithms use curvature information.

Just as we derived the implicit formula for the line segment between two points, we need to derive the implicit formula for a circle through three points. The implicit equation for a circle with radius r and center (x_0, y_0) is

$$(x - x_0)^2 + (y - y_0)^2 = r^2. \tag{6.13}$$

Consider three points $\mathbf{p}_1 = (x_1, y_1)$, $\mathbf{p}_2 = (x_2, y_2)$, and $\mathbf{p}_3 = (x_3, y_3)$. Transform the origin of the coordinate system to point $\mathbf{p}_1$. In the new coodinate system,

$$x' = x - x_1 \tag{6.14}$$
$$y' = y - y_1 \tag{6.15}$$

and the equation for the circle is

$$(x' - x'_0)^2 + (y' - y'_0)^2 = r^2. \tag{6.16}$$

Substitute the coordinates in the x'–y' space for the points p_1, p_2, and p_3 in the implicit equation for a circle:

$$x_0'^2 + y_0'^2 - r^2 = 0 \tag{6.17}$$
$$x_2'^2 - 2x'_2x'_0 + x_0'^2 + y_2'^2 - 2y'_2y'_0 + y_0'^2 - r^2 = 0 \tag{6.18}$$
$$x_3'^2 - 2x'_3x'_0 + x_0'^2 + y_3'^2 - 2y'_3y'_0 + y_0'^2 - r^2 = 0 \tag{6.19}$$

This yields three nonlinear equations for the three unknowns x'_0, y'_0, and r.

Subtract the first equation from the second and third equations:

$$2x'_2x'_0 + 2y'_2y'_0 = x_2'^2 + y_2'^2 \tag{6.20}$$
$$2x'_3x'_0 + 2y'_3y'_0 = x_3'^2 + y_3'^2 \tag{6.21}$$

This yields two linear equations in the two unknowns x'_0 and y'_0, which are the coordinates of the center of the circle in the x'–y' space. Add (x_1, y_1) to (x'_0, y'_0) to get the center of the circle in the original coordinate system. Compute the radius of the circle from $r^2 = x_0'^2 + y_0'^2$.

To calculate the error in fitting a circular arc, define the distance of point Q from the circle as the distance of Q from the circle along a line passing through the center of the circle. Let the radius of the circle be r. Compute the distance q with coordinates (x_i, y_i) from point Q to the center (x_0, y_0) of the circle:

$$q = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2} \tag{6.22}$$

The distance from point Q to the circular arc is

$$d = q - r \tag{6.23}$$

Now that we have a formula for fitting a circular arc to three points, we need a method for evaluating the goodness of fit so we can determine whether or not the circular arc is a better approximation to the edges than the line segments. If the ratio of the length of the contour to the distance between the first and last end points is more than a threshold, then it may be possible to replace the line segments with a circular arc. The circular arc is fit between the first and last end points and one other point. There are several methods for fitting a circular arc to a sequence of polylines, depending on how the middle point is chosen:

1. Use the polyline vertex that is farthest from the line joining the first and last end points.

2. Use the edge point that is farthest from the line joining the first and last end points.

3. Use the polyline vertex that is in the middle of the sequence of vertices between the first and last end points.

4. Use the edge point that is in the middle of the list of edges between the first and last end points.

Calculate the signed distance between all edge points and the circular arc. Compute the maximum absolute error and the number of sign changes. If the normalized maximum error is below a threshold and the number of sign changes is large, then accept the circular arc; otherwise, retain the polyline approximation. The algorithm for replacing line segments with circular arcs is outlined in Algorithm 6.2.

Algorithm 6.2 Replacing Line Segments with Circular Arcs

1. *Initialize the window of vertices to the three end points of the first two line segments in the polyline.*

2. *Compute the ratio of the length of the part of the contour corresponding to the two line segments to the distance between the end points. If the ratio is small, then leave the first line segment unchanged, advance the window of vertices by one vertex, and repeat this step.*

3. *Fit a circle through the three vertices.*

4. *Calculate the normalized maximum error and number of sign changes.*

5. *If the normalized maximum error is too large or the number of sign changes is too small, then leave the first line segment unchanged, advance the window of vertices, and return to step 2.*

6. *If the circle fit succeeds, then try to include the next line segment in the circular arc. Repeat this step until no more line segments can be subsumed by this circular arc.*

7. *Advance the window to the next three polyline vertices after the end of the circular arc and return to step 2.*

After running Algorithm 6.2 over the polyline, the contour will be represented by a sequence of line segments and circular arcs. It may be inconvenient to have two different curve primitives in the representation. In the next section, we will present conic sections, which allow line segments, circular arcs, and other primitives to coexist in the same representation. Conic sections also provide smooth transitions between sections, if desired, as well as the explicit representation of corners.

6.6 Conic Sections

This section describes how to approximate lists of edge points with conic sections. As with circular arcs, the method assumes that the edge points are first approximated by a polyline and replaces subsequences of line segments by conics.

The implicit (algebraic) form of a conic is

$$f(x, y) = ax^2 + 2hxy + by^2 + 2ex + 2gy + c = 0. \tag{6.24}$$

There are three types of conic sections: hyperbolas, parabolas, and ellipses. Circles are a special case of ellipses. Geometrically, conic sections are defined by intersecting a cone with a plane as shown in Figure 6.8.

Conic sections can be fit between three vertices in the polyline approximation to a contour. The locations where conic sections are joined are called knots. Conic splines are a sequence of conic sections that are joined end to

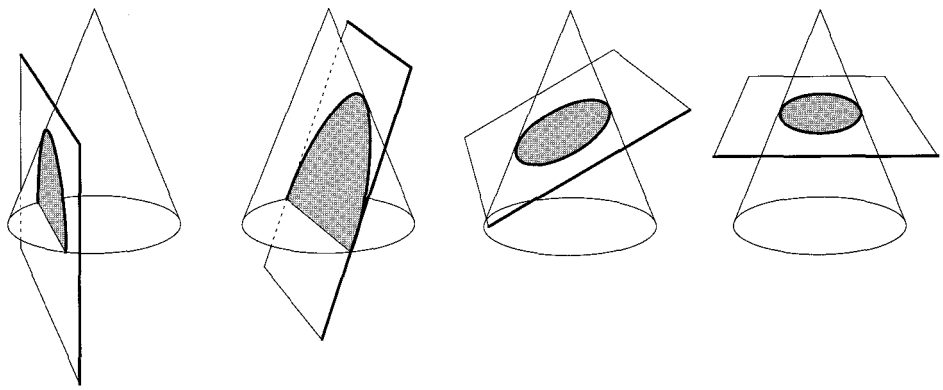

Figure 6.8: Conic sections are defined by intersecting a cone with a plane.

end, with equal tangents at the knots to provide a smooth transition between adjacent sections of the curve. Let the polyline vertices be V_i. The conic approximation is shown in Figure 6.9.

Each conic section in a conic spline is defined by two end points, two tangents, and one additional point. The knots K_i can be located between the vertices of the polyline:

$$K_i = (1 - \nu_i)V_i + \nu_i V_{i+1} \tag{6.25}$$

where ν_i is between 0 and 1. The tangents are defined by the triangle with vertices V_i, V_{i+1}, and V_{i+2}. The additional point is

$$Z_i = \gamma_i V_{i+1} + (1 - \gamma_i)\frac{K_i + K_{i+1}}{2}, \tag{6.26}$$

as shown in Figure 6.10.

There are several special cases of the conic section that can be handled in a uniform way by this representation. If $\nu_{i+1} = 0$, then the ith section of the conic spline is the line segment from K_i to V_{i+1}. If $\nu_i = 1$ and $\nu_{i+1} = 0$, then K_i, K_{i+1}, and V_{i+1} collapse to the same point and there is a corner in the sequence of conic sections. These special properties allow line segments and corners to be represented explicitly in a conic spline, without resorting to different primitives or special flags.

The algorithm presented here for computing conic splines uses the guided form of a conic section, which represents a conic section using three lines that

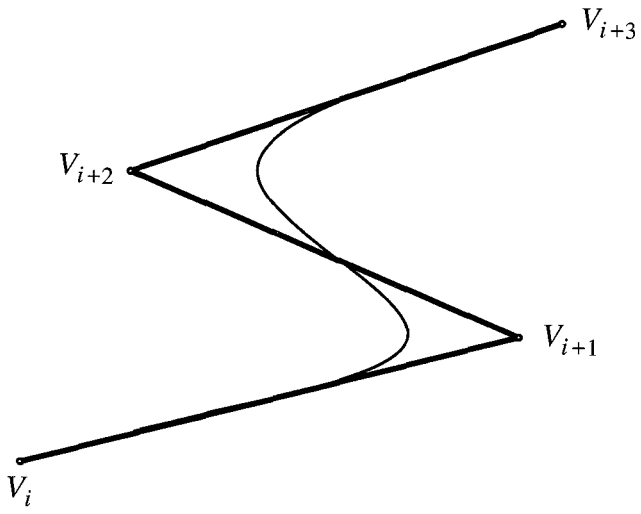

Figure 6.9: Conic sections are approximations defined between three points.

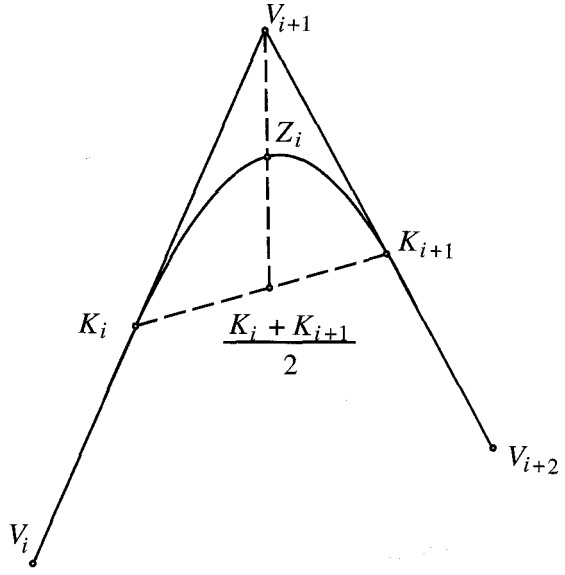

Figure 6.10: A conic section is defined by the two end points and tangents obtained from three vertices of the polyline approximation, plus one additional point.

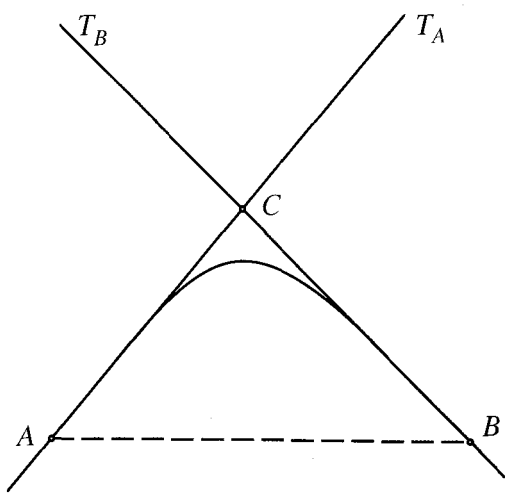

Figure 6.11: The guided form for a conic.

bound the conic. (See Figure 6.11.) The equation of a line is

$$a_0 + a_1 x + a_2 y = 0. \tag{6.27}$$

Let the first and last vertices in a polyline be A and B, and let point C be an intermediate vertex in the polyline. The first and last vertices are joined by the chord AB. The guided form of conic is the family of conics with end points at A and B and tangents AC and BC defined by the equation

$$(a_0 + a_1 x + a_2 y)(b_0 + b_1 x + b_2 y) = \rho(u_0 + u_1 x + u_2 y)^2, \tag{6.28}$$

where

$$a_0 + a_1 x + a_2 y = 0 \tag{6.29}$$

is the line containing the line segment AC,

$$b_0 + b_1 x + b_2 y = 0 \tag{6.30}$$

is the line containing the line segment BC, and

$$u_0 + u_1 x + u_2 y = 0 \tag{6.31}$$

is the line containing the chord AB. The family of conic sections is parameterized by ρ.

The algorithm for fitting a conic section to a list of edge points starts with a polyline and classifies the vertices as corners, soft vertices, or knots. Soft vertices have angles near 180°, and the adjacent line segments are nearly collinear and may be replaced with a conic section. A sequence of soft vertices corresponds to a sequence of line segments with gradually changing orientation that most likely were fitted to edge points sampled along a smooth curve. Corners have vertex angles above $180° + T_1$ or below $180° - T_1$, where T_1 is a threshold, and are unlikely to be part of the conic. Knots are placed along a line segment that has soft vertices at either end that are angled in opposite directions. A conic section cannot have an inflection, so two conic sections must be joined at the knot. The placement of the knot along the line segment is determined by the relative angles of the soft vertices at the ends of the line segment. Let the angles of the two soft vertices V_i and V_{i+1} be A_i and A_{i+1}, respectively. If $A_i = A_{i+1}$, then the knot is placed halfway between the vertices, which means that $\nu = 1/2$ in Equation 6.25. If the angles are not the same, then the knot location should be biased away from the vertex with the larger angle, since the conic may not bend away from the line segment fast enough to follow the corner. The value for ν in Equation 6.25 can be set using the formula

$$\nu_i = \frac{A_1}{A_1 + A_2}. \tag{6.32}$$

Each sequence of line segments joined by soft vertices is replaced by a guided conic through the first and last vertices (or knots). The tangents are defined by the orientation of the first and last line segments. The tangents and end points determine four of the five degrees of freedom for the conic. The conic is fully specified by having it pass through the soft vertex in the middle of the sequence.

6.7 Spline Curves

The term *spline* refers to a function represented using piecewise polynomials. Splines occur in many applications. In data analysis, splines are used to fit a set of data points when no function model is available [245]. In computer graphics and computer-aided design, splines are used to represent free-form curves. In machine vision, splines provide a general-purpose representation for curves when no simpler model is adequate.

A spline can be made from any class of functions joined end to end. The most common form of spline is the cubic spline, which is a sequence of piecewise cubic polynomials. The curve representations presented in previous sections, such as sequences of line segments, circular arcs, and conic sections, are other examples of splines. Cubic splines allow more complex curves to be represented using fewer spline segments. Cubic splines are widely used in computer drawing programs for free-form curves and for representing character outlines in fonts. Since cubic splines are so widely used, it may be necessary for a machine vision program to fit this curve model to an edge list. Since interactive graphics interfaces for manipulating cubic spline curves are well known, a contour represented as a cubic spline can be modified manually by the user if necessary. This is a very important consideration, since the results of fitting a curve to edges may never be perfect.

One point to make clear is the difference between geometric and parametric equivalence. Two curves are geometrically equivalent if they trace the same set of points. In other words, the two curves are geometrically equivalent if they correspond to the same shape (or set of points) in space. Two curves are parametrically equivalent if their equations are identical. In other words, two curves are parametrically equivalent if their representation uses the same formula with the same parameters. Parametric equivalence is stronger than geometric equivalence.

Two curves can be geometrically equivalent but have different parametric representations. This is an important concept for fitting curves in machine vision. A machine vision system might produce a representation based on cubic splines that is very close (geometrically) to the true representation of an object boundary, but the representation may not be at all similar in a parametric sense. In applications such as object recognition or comparing the image of an industrial part with its model, it is not possible to compare the parametric forms of the cubic spline curves. The comparison must be based on geometric equivalence.

Cubic splines have enough degrees of freedom to allow the orientation of edge fragments to be used in the approximation. Recall that most edge detection algorithms can provide estimates of edge orientation (gradient angle) as well the position of the edge. Only the positions of edges were used in the algorithms for fitting line segments, circular arcs, and conic sections. With cubic splines, we can introduce an example of how to use the orientation information produced by an edge detector.

The equation for a cubic curve in the plane is

$$\mathbf{p}(u) = (x(u), y(u)) = \mathbf{a}_0 + \mathbf{a}_1 u + \mathbf{a}_2 u^2 + \mathbf{a}_3 u^3, \tag{6.33}$$

where the coefficients a_0, a_1, a_2, and a_3 are two-element vectors (points in the image plane) and the parameter u covers the interval $[0, 1]$. The cubic curve begins at point $\mathbf{p}(0) = (x(0), y(0))$ and ends at point $\mathbf{p}(1) = (x(1), y(1))$. The cubic spline is a sequence of cubic curves $\mathbf{p}_1(u), \mathbf{p}_2(u), \ldots, \mathbf{p}_n(u)$, defined over successive intervals $[0, 1], [1, 2], \ldots, [n-1, n]$ and joined at the end points so that $\mathbf{p}_i(i) = \mathbf{p}_{i+1}(i)$. Each of the cubic curves in the spline is called a spline segment, and the edge points where the segments are joined are called knots.

As with the curve fitting algorithms presented in previous sections, the sequence of edge points is partitioned into subsequences and a spline segment is fit to each subsequence. Each cubic curve segment in the spline requires eight parameters. The positions of the first and last edge points in the subsequence provide four constraints. First-order continuity (equal tangent vectors) at the knots provides two more constraints. The orientation of the edges at the knots provides only one additional constraint on each segment, since the edge is shared by adjacent segments. Second-order continuity (equal curvature) at the knots would provide two more constraints, but then there would be too many equations for the eight parameters of each cubic spline segment.

It is important for the spline segments to be joined smoothly at the knots, and this is achieved in computer graphics by requiring second-order continuity. Requiring second-order continuity would overconstrain each spline segment, since the segments are already constrained to pass through selected edges with the orientation (tangent angle) constrained by the orientation of the edge; but one additional constraint can be provided by minimizing the magnitude of the second-order discontinuity at the knot. In other words, minimize the difference in curvature at the knots.

For the entire cubic spline curve, minimize the sum of the squared magnitude of the difference in the second derivative at the $n-1$ knots:

$$\chi^2 = \sum_{i=1}^{n-1} (\Delta \ddot{\mathbf{p}}), \tag{6.34}$$

where the difference in the second derivatives of two spline segments at their common knot is

$$\Delta\ddot{\mathbf{p}} = \ddot{\mathbf{p}}_{i-1}(1) - \ddot{\mathbf{p}}_i(0) \tag{6.35}$$

$$= 2\left(\mathbf{t}_{i-1} + 4\mathbf{t}_i + \mathbf{t}_{i+1} + 3(\mathbf{p}_{i-1} - \mathbf{p}_i)\right). \tag{6.36}$$

The variable $\mathbf{t}_i$ is the tangent vector at knot i. The tangent vector has an orientation $\hat{\mathbf{t}}_i$ given by the edge orientation (gradient angle) and a signed magnitude γ_i which is unknown:

$$\mathbf{t}_i = \gamma_i \hat{\mathbf{t}}_i. \tag{6.37}$$

In other words, the orientation of an edge at the knot is modeled as a unit tangent vector, but the cubic spline requires a tangent with sign and magnitude to indicate from which direction the curve should pass through the knot and at what speed. The algorithm solves a system of linear equations for the n unknowns γ_i which provide the missing information for constructing the cubic spline segments between the knots.

This algorithm does not have any additional parameters or thresholds, but, as with the algorithms for fitting polylines, circular arcs, and conics presented in previous sections, the knots must be chosen from the edge list. The knot locations can be determined by using any of the polyline algorithms described above to compute a polyline approximation to the contour. The polyline vertices can be used as the knot locations. The number and placement of knots can be adjusted to improve the fit of the cubic spline to the entire set of edge points.

The cubic spline fitting algorithm is very efficient, since the solution only requires solving a small system of linear equations for the tangent signs and magnitudes. There are many interactive graphics interfaces that allow the user to easily adjust the cubic spline curve, if necessary.

6.8 Curve Approximation

The curve fitting methods described in previous sections interpolated the curve through a subset of the edges. Higher accuracy can be obtained by computing an approximation that is not forced to pass through particular


edges.

This section presents methods for approximating a curve. There are several ways to approximate curves, depending on the reliability with which edge points can be grouped into contours. If it is certain that all of the edge points linked into a contour actually belong to the contour, then total least-squares regression can be used to fit a curve to the edge points. If some grouping errors are present, then robust regression methods can be used for computing the curve approximation. Finally, if the grouping of edges into contours is very unreliable, or if the edges are so scattered that grouping cannot be easily done using the edge linking or following methods discussed previously, then cluster analysis techniques must be used to perform grouping and curve fitting simultaneously. An excellent example of an algorithm for grouping and fitting scattered edge points is the Hough transform. All of these methods will be presented in the following sections.

The methods for fitting line segments, circular arcs, conic sections, and cubic splines to edge points presented in Sections 6.4 through 6.7 are trivial regression problems that fit curve segments between end points. These algorithms assume that the edge locations can be accurately computed, possibly using subpixel methods. Edge points in between the end points were not used in the regression. The accuracy of the curve approximation is determined by the accuracy of the location of the edge points chosen as the segment end points. The methods presented in this section will use all of the edge points to calculate the best approximation of the curve to the edge points.

The general curve fitting problem is a regression problem with the curve modeled by the implicit equation

$$f(x, y; a_1, a_2, \ldots, a_p) = 0 \tag{6.38}$$

with p parameters. The curve estimation problem is to fit the curve model to a set of edge points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$.

In the noise-free case, one can use p observations to formulate p equations for the p unknown curve parameters. Unfortunately, in most applications this direct approach is not suitable due to noise. Real applications usually require the best estimate of the parameter values using all of the information in the edge list.

The next section will cover least-squares regression as it is used for curve fitting in machine vision. Least-squares methods are appropriate when the errors are normally distributed. Section 6.8.3 will present robust methods

for curve fitting that are useful when some of the edge points have been incorrectly linked into the contour. These incorrectly assigned points are called *outliers.*

6.8.1 Total Regression

Classical linear regression minimizes the difference between a data point and the model in only one dimension, the dimension of the dependent variable. For example, a functional model of the form

$$y = f(x, a_1, \ldots, a_p) \tag{6.39}$$

relating the dependent variable y to the independent variable x, with the p model parameters a_1 through a_p, assumes that there are no errors in the independent variable x. In machine vision, errors in the x and y coordinates of location are equally likely and the curve model may be a vertical line, for instance, which cannot be represented in functional form. In machine vision, lines and other curve models are fitted to edges using total regression, which minimizes the sum of the squares of the perpendicular distances of the data points from the regression model. The advantage of this technique is that it compensates for errors in both the x and y directions. Total regression has actually already been presented in Chapter 2 where it was used to derive the equations for determining the orientation of a blob, although the term *total regression* was not used at the time.

To avoid problems when the line is vertical, represent the equation for a line by using polar coordinates:

$$x \cos\theta + y \sin\theta - \rho = 0. \tag{6.40}$$

Minimize the sum of the squared perpendicular distances of points (x_i, y_i) from the line:

$$\chi^2 = \sum_i (x_i \cos\theta + y_i \sin\theta - \rho)^2. \tag{6.41}$$

The solution to the total regression problem is

$$\rho = \bar{x} \cos\theta + \bar{y} \sin\theta \tag{6.42}$$

with

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{6.43}$$

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i. \tag{6.44}$$

The orientation of the total regression line is θ, given by

$$\tan 2\theta = \frac{a}{b}, \tag{6.45}$$

with

$$a = 2\sum_{i=1}^{n} x_i' y_i' \tag{6.46}$$

$$b = \sum_{i=1}^{n} x_i'^2 - \sum_{i=1}^{n} y_i'^2 \tag{6.47}$$

and

$$x_i' = x_i - \bar{x} \tag{6.48}$$

$$y_i' = y_i - \bar{y}. \tag{6.49}$$

Total regression uses a least-squares error norm that is optimal if the errors are from a normal distribution, but is not suitable if there are outliers present in the data. In the case of fitting a curve model to edge data, outliers would occur if the edge linking procedure incorporated one or more edges from other contours into the edge list for a contour. Outliers can occur even if the edge linking procedure performs flawlessly. For example, consider a list of edges from two adjacent sides of a rectangle. The corner must be identified in order to segment the edges into the two sides before fitting a line to the edges. If the corner point is not identified correctly, some edges may be assigned to the wrong side, and these edges are outliers.

In general, errors in classification introduce errors into the regression problem that are not normally distributed. In such a case, the errors may be modeled by a mixture distribution that combines a Gaussian distribution for modeling the normal errors with a broad-tailed distribution for modeling the outliers due to imperfect classification.

6.8.2 Estimating Corners

The best method for estimating corners is to use one of the methods for fitting a line to edge points and then compute the intersection of the lines. This method compensates for the error introduced by edge detection operators that round off the corners and is more accurate than using a corner detector which only uses local information.

Given the implicit equations for two lines,

$$a_1 x + b_1 y + c_1 = 0 \tag{6.50}$$

$$a_2 x + b_2 y + c_2 = 0, \tag{6.51}$$

the location of the intersection is

$$y = \frac{c_1 a_2 - c_2 a_1}{a_1 b_2 - a_2 b_1} \tag{6.52}$$

$$x = \frac{c_2 b_1 - c_1 b_2}{a_1 b_2 - a_2 b_1}. \tag{6.53}$$

If $a_1 b_2 - a_2 b_1$ is close to zero, then the lines are nearly parallel and cannot be intersected.

A good method for detecting corners is to try to fit pairs of lines over successive sublists of $2n + m$ edge points along the contour. The parameter n is the number of edge points required for an accurate line fit, and the parameter m is the number of edge points to skip between the sides of the corner. The gap skips over the edge points in the rounded part of the corner. A corner is detected by testing the magnitude of $a_1 b_2 - a_2 b_1$ against a threshold.

6.8.3 Robust Regression

If the errors are not from a normal distribution, then least-squares is not the best fitting criterion. Figure 6.12 shows an example of the problems encountered by least-squares regression when the data set contains outliers. Even a single outlier is enough to pull the regression line far away from its correct location. Robust regression methods try various subsets of the data points and choose the subset that provides the best fit.

The physical analogy shown in Figure 6.13 may make this discussion more clear. Imagine that you want to find the center of mass of a set of points

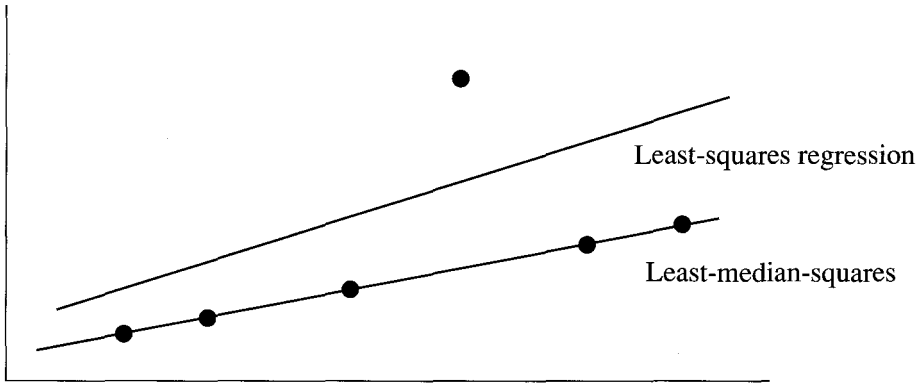

Figure 6.12: Illustration of the difference between fitting a curve using least-squares regression and fitting a curve using robust methods to a data set that contains outliers.

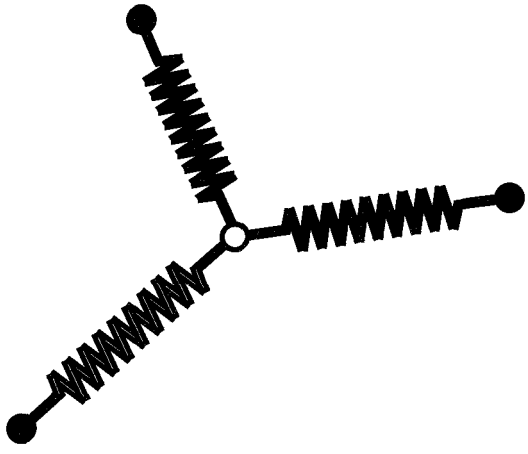

Figure 6.13: A physical analogy that illustrates the sensitivity of least-squares methods to outliers. Even a single outlier renders a least-squares solution useless.

in the plane. Attach springs with equal spring constants to the fixed points and to a small object that can move freely. The object will be pulled to the average of the locations of the points. The springs implement a least-squares norm through the spring equation for potential energy. This physical analogy corresponds to the derivation of the calculation of an average from the criterion that the sum of the squares of the residuals, the differences between each point and the average, should be minimized. Now suppose that one of the points can be moved. Call this point a leverage point. It is possible to force the location of the average to be shifted to any arbitrary point by pulling the leverage point far enough away. This illustrates the extreme sensitivity of estimators based on least-squares criteria to outliers. Even a single outlier can ruin an estimate. Ideally, one would like to break the spring connected to an outlier so that the estimate remains unharmed. Changing the spring constants so that points that are far away exert little influence on the estimate corresponds to the implementation of robust estimators based on influence functions. Breaking the springs attached to outliers corresponds to resampling schemes where a consistent subset of samples is determined. Resampling plans repeatedly draw random subsets and choose the subset that yields the best estimate. Examples of resampling algorithms include random sample consensus, least-median-squares regression, and other computer-intensive methods in regression.

The spring analogy also extends to linear regression with the same conclusions: even a single outlier will distort the regression estimate. A linear, multivariate model of order n is represented by the equation

$$\hat{y_i} = \hat{\theta_1} x_{i1} + \hat{\theta_2} x_{i2} + \cdots + \hat{\theta_n} x_{in} \tag{6.54}$$

for the ith data point, where $\hat{\theta_i}$ are the estimates of the model parameters θ_i. The residual for each data point (the deviation of the data point from the estimated model) is $r_i = y_i - \hat{y_i}$. In least-squares regression, the estimates of the model parameters are given by minimizing the sum of the squares of the residuals:

$$\min_{\hat{\theta}} \sum_{i=1}^{n} r_i^2. \tag{6.55}$$

As demonstrated by the spring analogy described above, the model parameters can be arbitrary if only one of the data points is an outlier.

Often, the noise and outliers can be modeled as a mixture distribution: a linear combination of a normal distribution to model the noise and a broad-

tailed distribution to account for outliers. In this case, it makes sense to formulate an estimator with a norm that resembles a least-squares norm for small errors but is insensitive to large errors so that outliers are ignored. This is called the influence function approach.

The breakdown point is the smallest percentage of data points that can be incorrect to an arbitrary degree and not cause the estimation algorithm to produce an arbitrarily wrong estimate [207]. Let Z be a set of n data points. Suppose that the set Z' is a version of set Z with m points replaced with arbitrary values. Let a regression estimator be denoted by $\hat{\theta} = T(Z)$. The bias in an estimate due to outliers is given by

$$Bias = \sup_{Z'} \parallel T(Z') - T(Z) \parallel . \tag{6.56}$$

The idea behind the breakdown point is to consider what happens to the bias as the number of outliers m as a percentage of the number of data points n is increased. Since the data points can be replaced with arbitrary values, for some ratio of m to n the bias can potentially be unbounded. This is the breakdown point. Below the breakdown point, the regression estimator may be able to totally reject outliers, or the outliers may have only some small effect on the estimate. Beyond the breakdown point, the outliers can drive the estimator to produce an arbitrary answer in the sense that the answer will depend on the outliers and not on the legitimate data. In other words, the result provided by the estimator is unpredictable. The breakdown point is defined as

$$\epsilon_n^\star = \min \left\{ \frac{m}{n} : Bias(m; T, Z) \text{ is infinite} \right\} . \tag{6.57}$$

For least-squares regression, $\epsilon_n^\star = 1/n$, and in the limit as the number of data points becomes large, $\epsilon_\infty^\star = 0$. In other words, least-squares regression has no immunity to outliers; a single outlier can completely ruin the result.

Least-median-squares regression is a very simple technique to implement and has proven to be very powerful in solving regression problems when there is a large percentage of outliers. Least-median-squares regression can tolerate up to 50 percent outliers in a data set. What this means is that up to half of the data points in a data set can be arbitrary without significantly affecting the regression result.

In least-median-squares regression, the estimates of the model parameters

Algorithm 6.3 Least-Median-Squares Regression
Assume that there are n data points and p parameters in the linear model.

1. *Choose p points at random from the set of n data points.*
2. *Compute the fit of the model to the p points.*
3. *Compute the median of the square of the residuals.*

The fitting procedure is repeated until a fit is found with sufficiently small median of squared residuals or up to some predetermined number of resampling steps.

are given by minimizing the median of the squares of the residuals:

$$\min_{\hat{\theta}} \operatorname*{med}_i r_i^2. \tag{6.58}$$

The least-median-squares algorithm is described in Algorithm 6.3.

The median has a 50 percent breakdown point, and this property carries over to least-median-squares regression [207]. In other words, even if as many as half of the data points are outliers, the regression estimate is not seriously affected. If more than 50 percent of the data points are outliers, then least-median-squares regression may not work well, and more powerful techniques, such as the Hough transform, must be used.

6.8.4 Hough Transform

The last few years have seen increasing use of parameter estimation techniques that use a voting mechanism. One of the most popular voting methods is the Hough transform. In the Hough transform, each point on a curve votes for several combinations of parameters; the parameters that win a majority of votes are declared the winners. Let us consider this approach for fitting a straight line to data. Consider the equation of a straight line:

$$y = mx + c. \tag{6.59}$$

In the above equation, x and y are observed values, and m and c represent the parameters. If the values of the parameters are given, the relationship

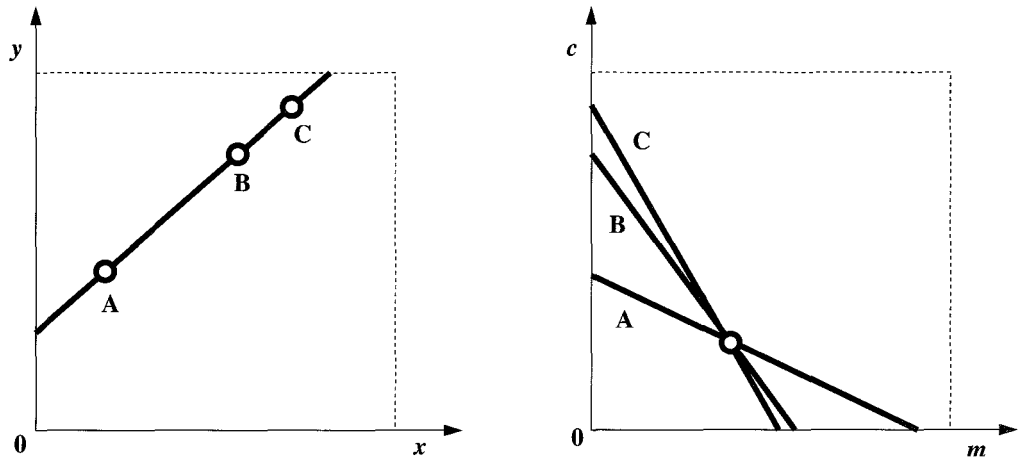

Figure 6.14: Image–to–parameter space mapping of a point in the Hough transform.

between the coordinates of the point is clearly specified. Let us rewrite the above equation as

$$c = -xm + y. \tag{6.60}$$

Now, in the above equation, let us assume that m and c are variables of interest, and x and y are constants. The equation above represents a straight line in the m–c space. The slope and intercept of this line are determined by x and y. A point (x, y) corresponds to a straight line in m–c space. This mapping is shown in Figure 6.14. It should be mentioned here that the shape of the curve in the parameter space depends on the original function used to represent the curve. In practice, the polar form of the line

$$\rho = x\cos\theta + y\sin\theta \tag{6.61}$$

is used rather than the explicit form to avoid problems with lines that are nearly vertical. Edge points (x, y) are mapped into the (ρ, θ) parameter space.

In the case of a straight line, as represented above, if there are n points lying on this straight line, then these points will correspond to a family of straight lines in the parameter space, as shown in Figure 6.14. All these lines will pass through the point (m, c) in the parameter space. This point gives the parameters of the original straight line.

Algorithm 6.4 Hough Transform Algorithm

1. *Quantize the parameter space appropriately.*
2. *Assume that each cell in the parameter space is an accumulator. Initialize all cells to zero.*
3. *For each point (x, y) in the image space, increment by 1 each of the accumulators that satisfy the equation.*
4. *Maxima in the accumulator array correspond to the parameters of model instances.*

If we are interested in finding the straight line that best fits n points in an image, then we can use the above mapping from the image space to the parameter space. In this approach, called the Hough transform, we represent the parameter space as an array of accumulators, representing discrete parameter values. Each point in the image votes for several parameters, according to the transformation equation. To find parameters that characterize the line, we should detect peaks in the parameter space. This general idea is highlighted in Algorithm 6.4.

The Hough transform does not require prior grouping or linking of the edge points, and the edge points that lie along the curve of interest may constitute a small fraction of the edges in the image. In particular, the number of edges that actually lie along the curve could be less than half of the number of edges in the scene, which would rule out most robust regression methods. The assumption behind the Hough transform is that in the presence of large amounts of noise, the best that can be done is to find the point in the parameter space that satisfies the maximum number of edges in the image. If the peak in the parameter space covers more than one accumulator, then the centroid of the region that contains the peak provides an estimate of the parameters.

If there are several curves in the image that are matched by the model, then there will be several peaks in the parameter space. It is possible to detect each peak, remove the edges associated with the curve instance corresponding to the peak, and continue to detect the remaining curves, until the peaks are not significant. However, it can be difficult to determine whether a peak is significant.

Another problem with the Hough transform is that the size of the discrete parameter space increases very quickly as the number of parameters increases. For a circular arc, the parameter space has three dimensions; for other curves the dimensionality may be even higher. Since the number of accumulators increases exponentially with the dimension of the space, the Hough transform may be computationally very inefficient for complex models. Several methods have been suggested to improve the performance of the Hough transform. One method uses gradient information for boundaries to reduce work in the parameter space. Suppose that the curve model is a circle. This model has three parameters: two parameters for the center of the circle and one parameter for the radius of the circle. If the gradient angle for edges is available, then this provides a constraint that reduces the number of degrees of freedom and hence the required size of the parameter space. The direction of the vector from the center of the circle to each edge is determined by the gradient angle, leaving the value of the radius as the only unknown parameter. There are other methods that may be used to reduce the size of the parameter space.

The details of using the gradient angle to reduce the size of the parameter space are explained for circle fitting. The algorithm is detailed in Algorithm 6.5. The implicit equation for a circle is

$$(x-a)^2 + (y-b)^2 = r^2. \tag{6.62}$$

The parametric equations for a circle in polar coordinates are

$$x = a + r\cos\theta \tag{6.63}$$

$$y = b + r\sin\theta. \tag{6.64}$$

Solve for the parameters of the circle to obtain the equations:

$$a = x - r\cos\theta \tag{6.65}$$

$$b = y - r\sin\theta. \tag{6.66}$$

Given the gradient angle θ at an edge point (x, y), compute $\cos\theta$ and $\sin\theta$. Note that these quantities may already be available as a by-product of edge detection. Eliminate the radius from the pair of equations above to yield

$$b = a\tan\theta - x\tan\theta + y. \tag{6.67}$$

Algorithm 6.5 Circle Fitting Algorithm

1. *Quantize the parameter space for the parameters a and b.*
2. *Zero the accumulator array $M(a, b)$.*
3. *Compute the gradient magnitude $G(x, y)$ and angle $\theta(x, y)$.*
4. *For each edge point in $G(x, y)$, increment all points in the accumulator array $M(a, b)$ along the line*

$$b = a \tan\theta - x \tan\theta + y. \tag{6.68}$$

5. *Local maxima in the accumulator array correspond to centers of circles in the image.*

This is the equation for updating the accumulators in the parameter space. For each edge point at position (x, y) with edge orientation θ, increment the accumulators along the line given by Equation 6.67 in the (a, b) parameter space.

If the radius is known, then it is only necessary to increment the accumulator for the point (a, b) given by

$$a = x - r\cos\theta \tag{6.69}$$
$$b = y - r\sin\theta. \tag{6.70}$$

It is not necessary that the curves to be detected by the Hough transform be described by a parametric equation. The Hough transform can be generalized into a voting algorithm (see Algorithm 6.6) that implements template matching efficiently.

Algorithm 6.6 encodes the shape of the object boundary in a table for efficient access. One point on the object is chosen as the reference point. By definition, the location of the reference point in the image is the location of the object. For each image gradient point at (x, y) with gradient angle θ, the possible locations of the reference point are given by

$$a = x - r(\theta)\cos(\alpha(\theta)) \tag{6.71}$$
$$b = y - r(\theta)\sin(\alpha(\theta)). \tag{6.72}$$

Algorithm 6.6 Generalized Hough Transform

1. *Pick a reference point on the object.*
2. *Compute the gradient angles θ_i along the object boundary.*
3. *For each gradient point θ_i, store the distance r_i and angle α_i from the reference point.*

Each possible reference point location is incremented. The location of the peak in the parameter space is the estimate for the location of the object. It is not easy to generalize this technique to incorporate changes in scale or rotation.

6.9 Fourier Descriptors

Since the position along a closed contour is a periodic function, Fourier series may be used to approximate the contour. The resolution of the contour approximation is determined by the number of terms in the Fourier series.

Suppose that the boundary of an object is expressed as a sequence of coordinates $\mathbf{u}(n) = [x(n), y(n)]$, for $n = 0, 1, 2, \ldots, N-1$. We can represent each coordinate pair as a complex number so that

$$u(n) = x(n) + j\, y(n) \tag{6.73}$$

for $n = 0, 1, 2, \ldots, N-1$. In other words, the x axis is treated as the real axis, and the y axis is treated as the imaginary axis of a series of complex numbers. Note that for a closed boundary, this sequence is periodic with period N and that now the boundary is represented in one dimension.

The discrete Fourier transform (DFT) representation of a one-dimensional sequence $u(n)$ is defined as

$$u(n) = \sum_{k=0}^{N-1} a(k) e^{\frac{j2\pi kn}{N}}, \qquad 0 \le n \le N-1 \tag{6.74}$$

$$a(k) = \frac{1}{N} \sum_{n=0}^{N-1} u(n) e^{\frac{-j2\pi kn}{N}}, \qquad 0 \le k \le N-1. \tag{6.75}$$

The complex coefficients $a(k)$ are called the *Fourier descriptors* of the boundary.

Fourier descriptors are compact representations for closed contours. However, low-resolution approximations, using only the low-order terms in the series, can be used as an even more compact representation. If only the first M coefficients are used, which is equivalent to setting $a(k) = 0$ for $k > M-1$, the following approximation to $u(n)$ is obtained:

$$\hat{u}(n) = \sum_{k=0}^{M-1} a(k) e^{\frac{j2\pi kn}{N}}, \qquad 0 \leq n \leq N-1. \tag{6.76}$$

Although only M terms are used to obtain each component of the boundary $u(n)$, n still ranges from 0 to $N-1$. In other words, the same number of points are in the approximated boundary, but not as many terms are used in reconstructing each point.

Simple geometric transformations of a boundary, such as translation, rotation, and scale, are related to simple operations of the boundary's Fourier descriptors (see Exercise 6.18). This makes the use of Fourier descriptors attractive for boundary matching. However, Fourier descriptors do have problems with occluded shapes. There are other methods for obtaining similar descriptors, using other boundary representations.

Further Reading

The Hough transform is an efficient method for detecting lines and other features from imperfect edges. A discussion of generalized Hough transform methods is given in the paper by Ballard [18]. Generalizing the Hough transform to detect arbitrary shapes, Stockman used Hough transform techniques for pose clustering in 2-D and 3-D problems in object recognition and localization [228]. Asada and Brady [45, 10] present a rigorous approach to contour and region descriptors. Work on measuring the deviation from roundness of a circle has been reported by Van-Ban and Lee [243].

The hop-along algorithm approximates a contour by a sequence of line segments but works on short subsequences of edges and is given in [193]. Curve fitting using line segments (polylines) and circular arcs is adapted from Pavlidis [193]. The method for fitting cubic splines to edge points using orientation was published by Tehrani, Weymouth, and Schunck [231].

Algorithms for fitting a circle to three points have been developed in computer graphics [204].

A discussion of robust regression using M-estimators is provided in numerical recipes [197, pp. 558–565]. Another robust regression technique is least-median-squares regression [207]. The influence function approach was pioneered by Huber [115, 97]. For resampling algorithms including random sample consensus, see [38, 80]. Least-median squares regression is discussed by Rousseeuw and Leroy [207], and another good source for information on computer-intensive methods in regression is a review article by Efron [71].

The Fourier descriptor technique has been applied to medical imagery by Staib and Duncan [227].

Exercises

6.1 What is a contour? How is it related to a region? What does an open contour represent?

6.2 List the criterion you will consider in selecting a contour representation. Discuss the implications of these factors for object recognition.

6.3 What is the difference between interpolation and approximation methods? Which one is better?

6.4 To implement rotation by $n \times 22.5°$, one may use a chain code using 16 directions. How can you implement such a code? Why is the 8-direction chain code almost always used?

6.5 The D_8 distance with respect to the origin is defined as $\max(x, y)$. Using this measure, find the signature of the contour in Figure 6.15 by plotting the D_8 distance as a function of pixel number. In addition, find the 8-direction chain code and difference code measured in the counterclockwise direction for the following object. Note that the starting point is an empty circle as opposed to a solid dot.

6.6 What are the criteria for choosing the threshold for normalized maximum error? (*Hint:* Consider the variance in the estimate of edge location.)

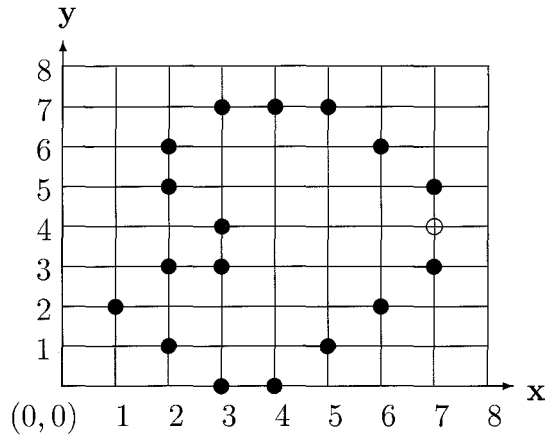

Figure 6.15: Contour for Exercise 6.5.

6.7 Consider the method for estimating the location of a corner presented in Section 6.8.2. What is the value of $a_1b_2 - a_2b_1$ when the lines are at right angles? What is the value when the lines meet at an angle θ? This is the formula to use for setting the threshold for corner detection.

6.8 Consider the method for estimating the location of a corner presented in Section 6.8.2. Assume that the error in the x and y coordinates of the edge locations has a normal distribution with variance σ^2. What is the error distribution for the location of the corner? How is the error affected by the angle of the corner?

6.9 Why is the Ψ-s representation considered a continuous chain code? What are its most attractive features?

6.10 How can you compare two objects using their Ψ-s representations?

6.11 Discuss different error measures that you can use in approximation. What is the role of an error measure in an approximation technique?

6.12 What is a conic section? How many types of conic sections are possible? Are there mathematical conditions that define types of a conic section?

6.13 What is a spline? Where can it be used? Why are cubic splines considered more powerful representations than polylines and conic sections?

6.14 Discuss the difference between geometric equivalence and parametric equivalence.

6.15 Why is the least-squares measure used in total regression? List its advantages and disadvantages.

6.16 How does robust regression overcome limitations of the total regression? Why is robust regression not very popular in approximation?

6.17 What are the strengths and weaknesses of Fourier descriptors for approximating and representing closed contours?

6.18 Several geometric transformations of object boundaries are related to simple operations on the Fourier descriptors as follows:

Transformation	Boundary	Fourier descriptor
Identity	$u(n)$	$a(k)$
Translation	$\tilde{u}(n) = u(n) + u_0$	$\tilde{a}(k) = a(k) + u_0\delta(k)$
Scaling or zooming	$\tilde{u}(n) = \alpha u(n)$	$\tilde{a}(k) = \alpha a(k)$
Starting point	$\tilde{u}(n) = u(n - n_0)$	$\tilde{a}(k) = a(k)e^{\frac{-j2\pi n_0 k}{N}}$
Rotation	$\tilde{u}(n) = u(n)e^{j\theta_0}$	$\tilde{a}(k) = a(k)e^{j\theta_0}$
Reflection	$\tilde{u}(n) = u^*(n)e^{j2\theta} + 2\gamma$	$\tilde{a}(k) = a^*(-k)e^{j2\theta} + 2\gamma\delta(k)$

Consider a simple square object with coordinates of boundary points $A = (0,0), B = (0,1), C = (1,1)$, and $D = (1,0)$.

a. Find its Fourier descriptors when the starting point is A and the boundary is traversed in the order A, B, C, D.

b. Find the descriptors when the object translates such that A is at $(2,3)$.

c. Find the descriptors of the translated object if the length of the sides of the square is changed to 2.

d. Find the descriptors of the translated and scaled object if the starting point is changed to B.

6.19 The Fourier descriptors of a simple square object with coordinates of boundary points given by $A = (-0.5, -0.5), B = (-0.5, 0.5), C = (0.5, 0.5), D = (0.5, -0.5)$ are given by

$$a(0) = a(1) = a(2) = 0 \tag{6.77}$$

$$a(3) = -0.5 - j\,0.5. \tag{6.78}$$

Starting with this data and using the properties of the Fourier descriptors given in the above problem, find the Fourier descriptors for the object given by $P = (0, 1), Q = (-1, 2), R = (0, 3), S = (1, 2)$.

6.20 What is the Hough transform? Is it related to robust regression? How?

6.21 Can you extend the Hough transform to detect an arbitrary shape? How will you develop a Hough transform that detects an object in its rotated and scaled versions?

6.22 Find the Hough transform of the lines enclosing an object with vertices $A = (2, 0), B = (2, 2)$, and $C = (0, 2)$. Sketch the modified object enclosed by lines obtained by replacing (ρ, θ) of the object lines by $(\rho^2, \theta + 90°)$. Calculate the area of the modified object.

6.23 The two-dimensional Hough transform for line detection can be generalized to the 3-D case to detect planes. The Hough domain parameters are then specified by the three variables ρ, θ, and ϕ, where the angles are measured as shown in Figure 6.16. Consider a unit cube whose diagonal corners, A and G, are located at $(1, 1, 1)$ and $(2, 2, 2)$ as shown. Find the Hough transform of the plane passing through the vertices C, H, and F.

Computer Projects

6.1 Develop an algorithm to find the chain code of a given curve. Can you determine corners using chain code? If so, implement this algorithm and test it on several images.

6.2 Develop a program to start with an image and find the slope–arc–length plot of the largest object in the image. Segment this object in its linear and circular segments.

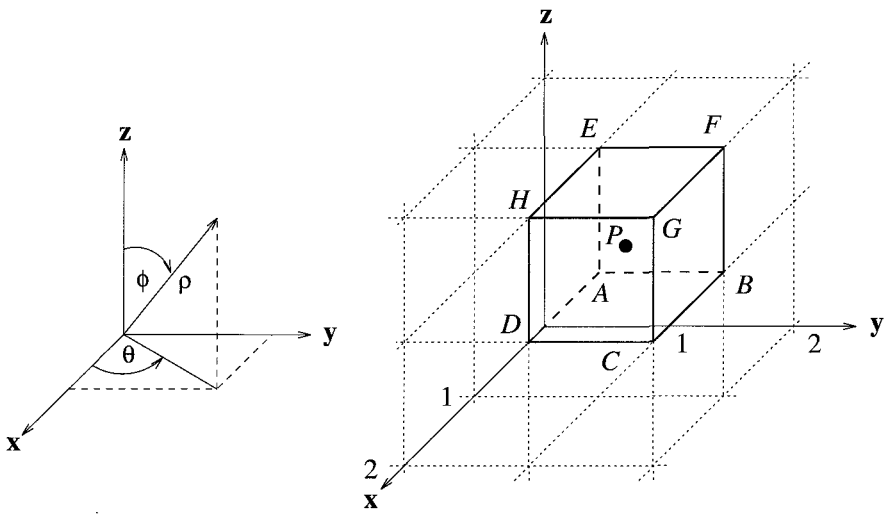

Figure 6.16: Diagram for Exercise 6.23.

6.3 Develop an algorithm to accept given points and provide a cubic spline. Apply this algorithm to the output of an edge detector.

6.4 Develop a Hough transform algorithm to detect straight lines in images. Use this to approximate an image by finding all lines that are above a fixed size, say 20 points.

6.5 Four binary objects are shown in Figure 6.17(a). Scaled versions of the same objects are shown in Figure 6.17(b), and scaled and rotated versions are shown in Figure 6.17(c). Consider various contour representation methods and comment on their suitability for matching scaled and scaled-rotated versions of objects with their corresponding original images. Implement one of the methods as a computer program.

6.6 Create a synthetic image of a rectangle with uniform intensity against a uniform background. Compute the edges using any edge detector from Chapter 5. Fit polylines to the edges. How close are the vertices to the true corner locations? Is the error consistently biased in one direction?

6.7 Create a synthetic image of a rectangle with rounded corners, modeled as quarter circles. Use edge detection and polyline fitting, and then replace runs of polylines with circular arcs using the algorithm presented

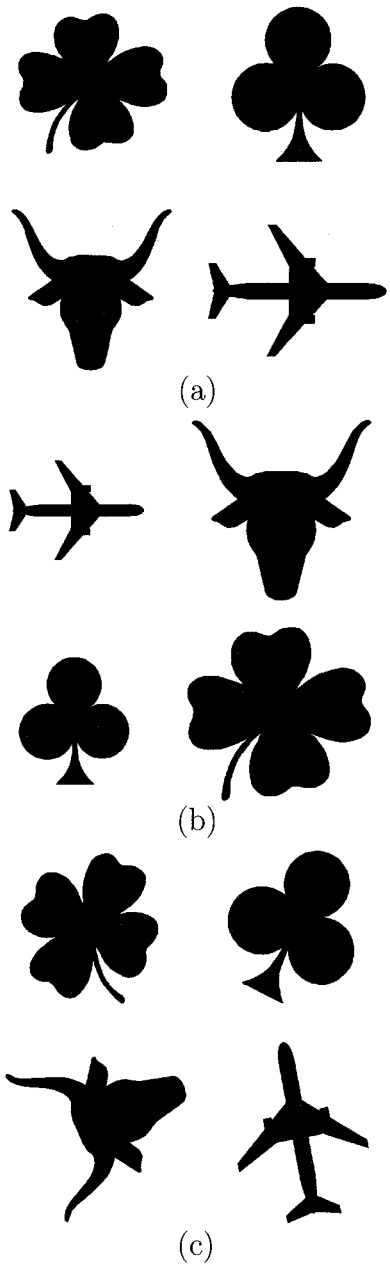

Figure 6.17: Binary objects used in Computer Project 6.5. (a) Four reference binary objects. (b) Scaled versions of the objects in (a). (c) Scaled and rotated versions of the objects in (a).

in this chapter for fitting circular arcs. Measure the error in the computed end points of the circles. Are the errors symmetrical, or are the corners unevenly distorted so that the rectangle appears skewed?

6.8 Experiment with the corner fitting algorithm described in Section 6.8.2. Create a synthetic image of a rectangle, adding noise from a normal distribution and scattered edges to simulate false positives in edge detection. Plot the error in the corner estimates versus the noise level.

6.9 Implement least-median-squares regression for fitting a line to a list of edge points. Add false edges to the list to simulate grouping errors. Plot the maximum distance of the estimated line from the true line versus the number of false edges.

6.10 Consider an object that is symmetrical about a vertical axis. Suppose that two lists of edge fragments along the left and right sides of the object are available. Adapt the algorithm for fitting cubic splines to edges to enforce the symmetry constraint. Generalize the algorithm so that the axis can be at any orientation.

6.11 Suppose that the list of edge points along the contour of an object is available and that the object is symmetric about some axis. Process the edge list into a sequence of line segments and circular arcs. Develop an algorithm for matching line segments and circular arcs to detect that the object is symmetric and estimate the axis of symmetry. After the axis of symmetry has been determined, this information can be used to improve the estimates of the line segments and circular arcs. Develop an algorithm for refining the contour representation. Experiment with an iterative algorithm that detects the axis of symmetry, uses this information to refine the contour representation, and then uses the improved contour representation to refine the axis estimate, repeating these steps until the axis and contour representations converge.

6.12 Consider an image of two rectangles. The task is to measure the gap between the two rectangles. The facing sides of the two rectangles are parallel, so the width of the gap is constant along its length. Explore different algorithms for measuring the gap.

a. Develop a formula for the average distance between two line segments that are nearly parallel and approximately of the same length. (*Hint:* Represent each line segment in parametric form on the interval $[0, 1]$.)

b. Compute the polyline representation for the rectangles. Are the line segments that bound the gap skewed due to poor estimates of the corner locations? Use the formula from part **a** to estimate the width of the gap.

c. Modify the algorithm to use improved techniques for estimating the locations of the corners. Measure the improvement in the estimate of the gap width.

d. Implement least-median-squares regression and repeat the measurements of the gap width, comparing the results using least median squares regression with the previous techniques.

e. Repeat the experiment using lines estimated with the Hough transform.

Synthetic noise, including noise from a normal distribution and scattered edges that model false positives in edge detection, can be added to the image to test the accuracy of gap measurement at various noise levels. Prepare a plot of the error in gap width measurement versus the noise level for each of the techniques.

6.13 Consider a line of text on a printed page that has been scanned into the computer. The task is to fit a line through the baseline of the text. The estimated baseline can guide algorithms for character recognition. Note that the baseline is not necessarily horizontal since the page may not be exactly vertical when scanned.

a. Suppose that edge detection has reduced the line of text to a list of edge points. Compare line fitting with least-squares, least-median-squares, and the Hough transform.

b. Consider multiple lines of text, reduced to multiple edge lists. Assume that the baseline separation is constant and that the baselines are parallel. Use these constraints to formulate a more accurate algorithm for determining the baselines.

c. Suppose that the baselines are parallel, but the baseline separation varies between lines. How does this change the algorithm?

6.14 Consider a list of edge points that form a circle. The task is to estimate the position and radius of the circle. Use nonlinear regression to fit a circle to the edge points, assuming that the radius of the circle is known. Modify the algorithm to determine both the position and radius. Finally, modify the algorithm to determine both the position and radius of the circle, but include a penalty term for variation in the estimated radius from a nominal value.