# Edge Contour Representation

(Jain et al., Chapt 6, Gonzalez et al., Chapt 8)

(1) The simplest representation of a contour is using an ordered list of its edge points.

* As accurate as the location estimates for the edge points.
* Not very compact representation.
* Not very effective representation for subsequent image analysis.

(2) A more powerful representation is fitting an appropriate curve having some analytical description (i.e., line segments, circular arcs, cubic splines).

* Increases accuracy (errors in edge location are reduced through averaging). (e.g., fitting a line to a set of edge points that lie along a line)
* More compact and efficient representation for subsequent image analysis.

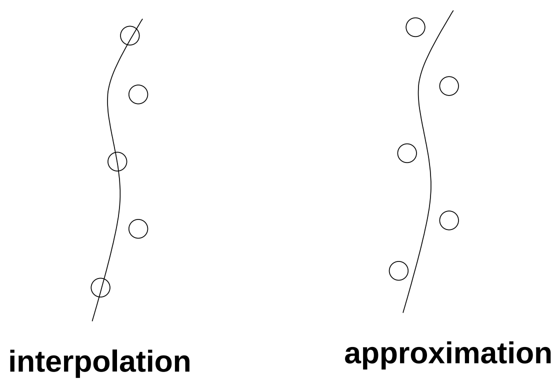
## • Curve interpolation vs curve approximation

- A curve interpolates a list of points if it passes through the points.

- A curve approximates a list of points if it passes close to the points, but not necessarily passing exactly through the points.

- Curve interpolation methods are more appropriate when the edges have been extracted accurately.

- In practice, curve approximation methods yield better results because the edge locations can not be extracted very accurately.



## • Digital contours

- The length of a digital contour $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, ..., $p_n = (x_n, y_n)$ can be approximated by adding the lengths of the individual segments between edge points:

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

- Slope (tangent) and curvature are difficult to compute precisely since the angle between neighboring edge pixels is quantized to 45 degrees increments.

- The idea is to compute the slope using edge points that are not adjacent:

<u>left k-slope:</u> the direction between points $p_{i-k}$ and $p_i$
<u>right k-slope:</u> the direction between points $p_k$ and $p_{i+k}$
<u>k-curvature:</u> the difference between the left and right $k$-slopes.

## • Distance of point from line

- Equation of line passing from two points $(x_1, y_1)$, $(x_2, y_2)$:

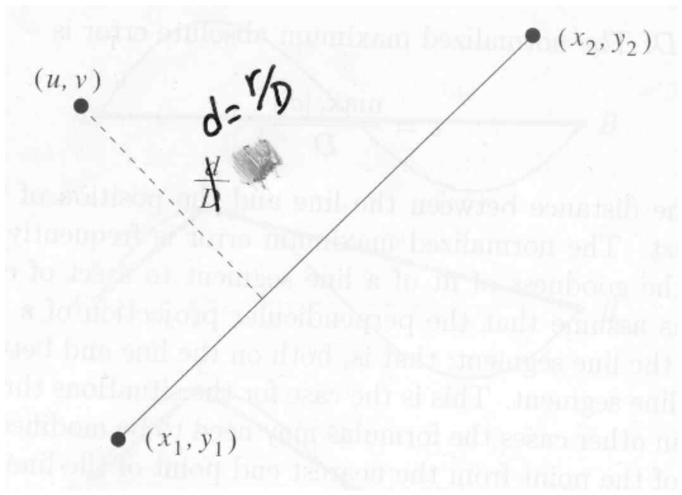
$$x(y_1 - y_2) + y(x_1 - x_2) + y_2 x_1 - y_1 x_2 = 0$$

- Distance $d$ of edge point $(u, v)$ from the line:

$$d = \frac{r}{D}$$

$$r = u(y_1 - y_2) + v(x_1 - x_2) + y_2 x_1 - y_1 x_2$$

$D$ is the distance between $(x_1, y_1)$ and $(x_2, y_2)$.



## • Evaluating the goodness of the fit

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

$$MAE = \max_i |d_i|$$

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

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

Normalized maximum absolute error: the ratio of the maximum absolute error to the length of the curve (error becomes independent of the length of the curve).

$$NMAE = \frac{\max_i |d_i|}{L}$$

Ration of curve length to the end point distance: a good measure of the complexity of the curve.

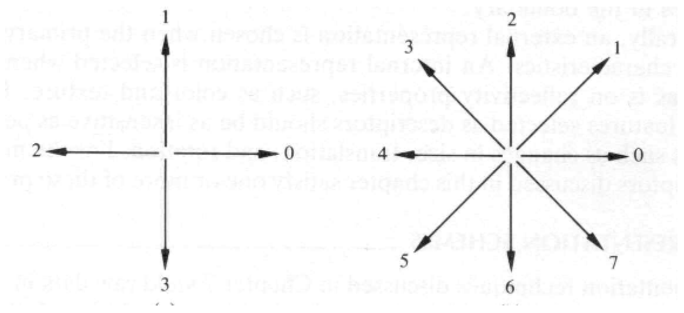
# Contour representation using edge points

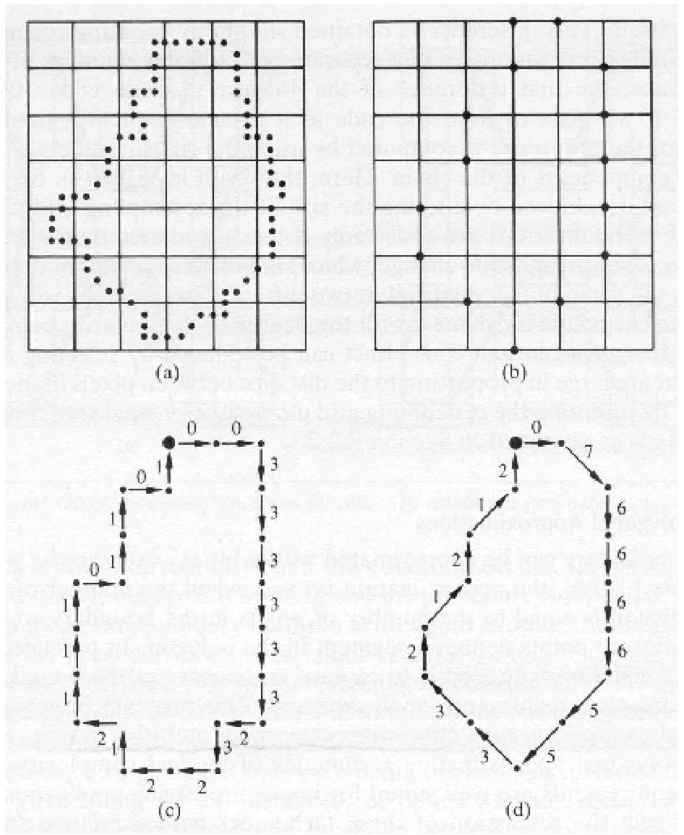
## • Chain-code representation

- Specifies the direction of each edge point along the contour.

(1) Start at the first edge point and go clockwise around the contour.

(2) The direction to the next edge point is specified using one of the four or eight quantized directions.





chain code: 0033333...01

- The chain-code can be made invariant to starting point by:

    * Treating the code as a circular sequence

    * Choosing the starting point such as the resulting code forms an integer of minimum magnitude.

- The derivative of the chain-code (difference code) is invariant to rotation:

    10103322 -> 33133030    (i.e., 2-1->3, 1-0->3, 0-1->1, 1-0->3, ....)

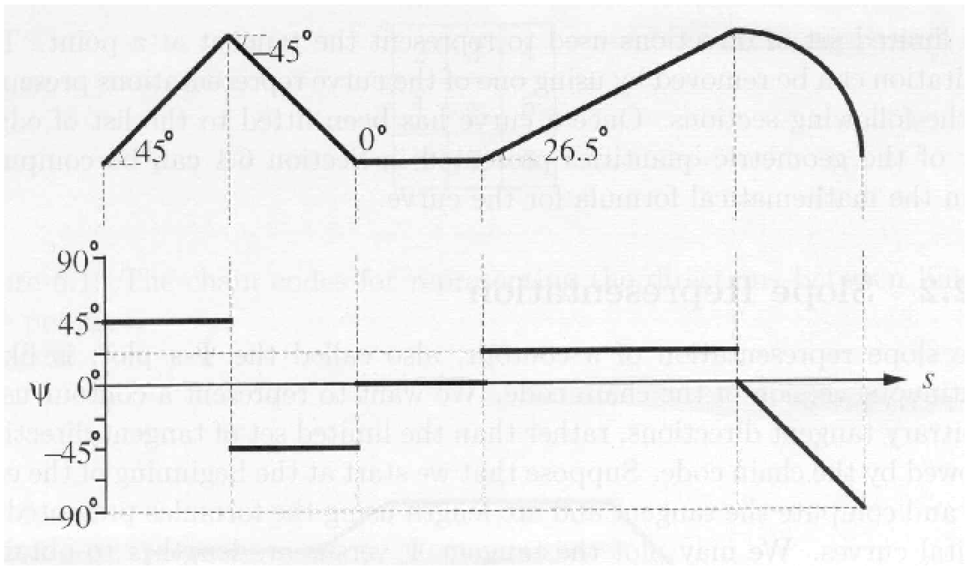
• **Slope representation** (*ψ*-*s* plot)

- A continuous version of the chain code (arbitrary directions).

(1) Start at the first edge point and go clockwise around the contour.

(2) Estimate the slope and arc length using the formulas given previously for digital contours.

(3) Plot the slope versus the arc-length.



- The $\psi$-$s$ plot is a representation of the shape of the contour:

  * Horizontal line segments in the $\psi$-$s$ plot correspond to straight lines in the contour.

  * Line segments at other orientations correspond to circular arcs.

  * Portions of the $\psi$-$s$ plot that are non straight lines correspond to other curve primitives.

- Some comments

  * Different starting points cause a shift in the $s$-axis.

  * Rotations cause a shift in the $\psi$-axis.

  * The $\psi$-$s$ plot is not very tolerant to noise.

## • Slope density function

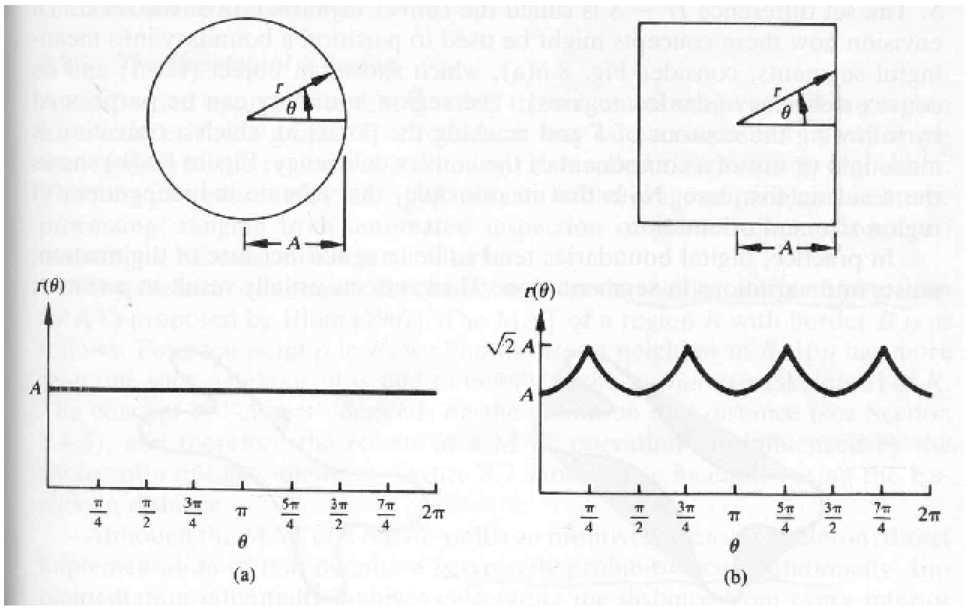
- The histogram of the slopes along a contour.

- Can be very useful for object recognition.

- Orientation can be determined using correlation of slope histograms.

• **Centroidal profile representation**

- Plot the distance from the centroid to the boundary as a function of angle.



- Some comments

  * Invariant to rotation but depends on starting point.

  * Can be made invariant to scaling (e.g., divide by the largest distance).

  * Tolerant to noise.

# Contour representation using curve fitting (interpolation)

- The main assumption when using interpolation is that the edge points have been extracted accurately.

- Errors in edge locations can be handled better using curve-fitting based on approximation (will be discussed later).

## • Curve fitting models

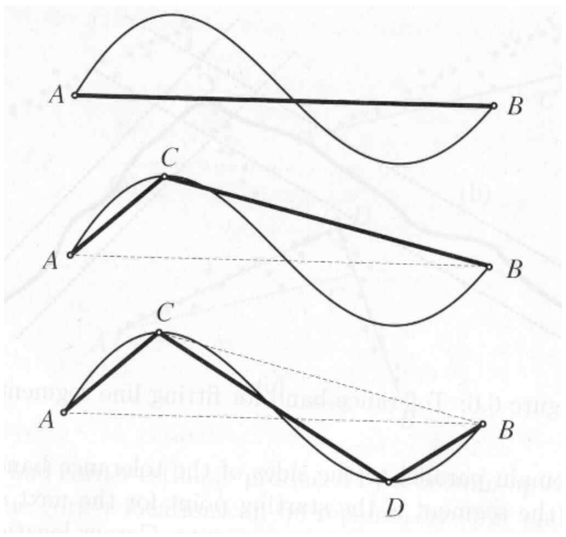
- Straight line segments
- Circular arcs
- Conic sections
- Cubic splines

## • Polygonal representation

- Fits the edge points with a sequence of line segments (i.e., contour is represented as a polygon).

- The polygon interpolates a selected subset of edge points (i.e., the ends of each line segment are edge points).

- We will be referring to those edge points as vertices.



- Three ways to compute the polygonal approximation of a contour:

<u>Split algorithm (top-down)</u>

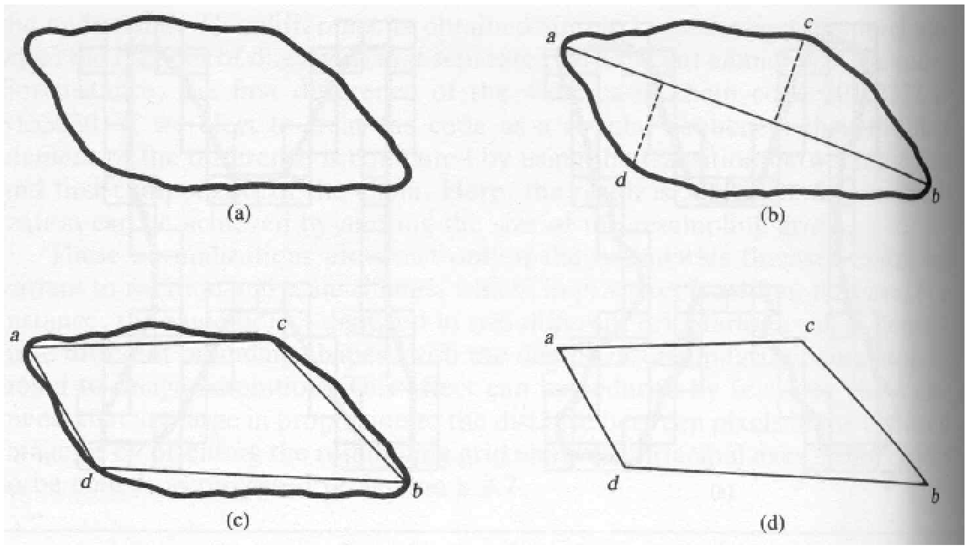
(1) Take the line segment connecting the end points of the contour (if the contour is closed, consider the line segment connecting the two farthest points).

(2) Find the farthest edge point from the line segment.

(3) If the normalized maximum error of that point from the line segment is above a threshold, split the segment into two segments at that point (i.e., new vertex).

(4) Repeat the same procedure for each of the two subsegments until the error is very small.

- Can be used for closed contours too!



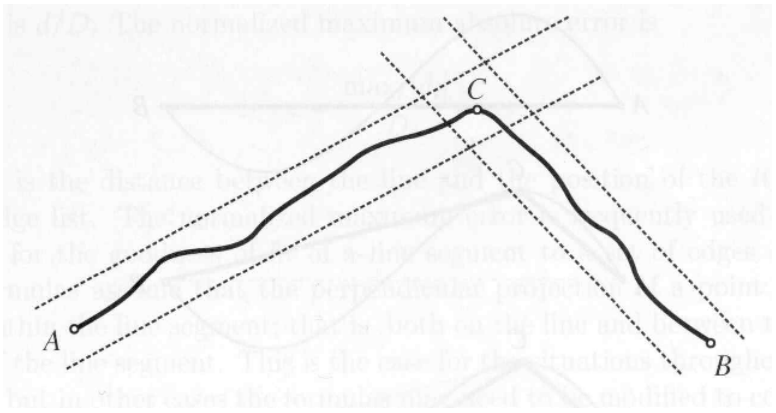
## Merge algorithm (bottom-up)

(1) Use the first two edge points to define a line segment.

(2) Add a new edge point if it does not deviate too far from the current line segment.

(3) Update the parameters of the line segment using least-squares.

(4) Start a new line segment when edge points deviate too far from the line segment.

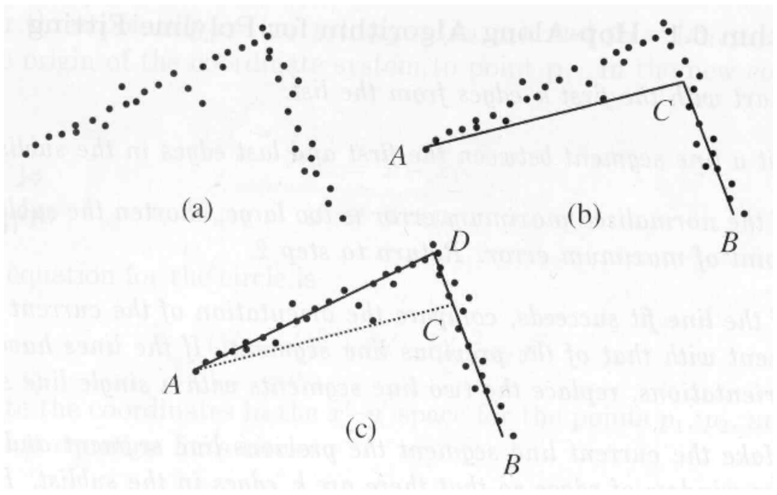


## Split and Merge algorithm

- The accuracy of line segment approximations can be improved by interleaving merge and split operations.

(1) After recursive subdivision (split), allow adjacent segments to be replaced by a single segment (merge).

(2) Alternate applications of split and merge until no segments are merged or split.



- Some comments

  * Polygonal representations are more economical that using edge points.

  * We can make the error as small as desired by splitting the contour into very small line segments.

• **Circular arcs**

- Once an edge list has been approximated by line segments, subsequences of the line segments can be replaced by circular arcs.

- This involves fitting circular arcs through the endpoints of two or more line segments.

(1) Initialize the window of vertices to the three endpoints of the first two line segments in the polygonal approximation.

(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) If the fit is not good (i.e., normalized maximum error is too large), then leave the first line segment unchanged, advance the window of vertices, and return to step 2.

(5) 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.

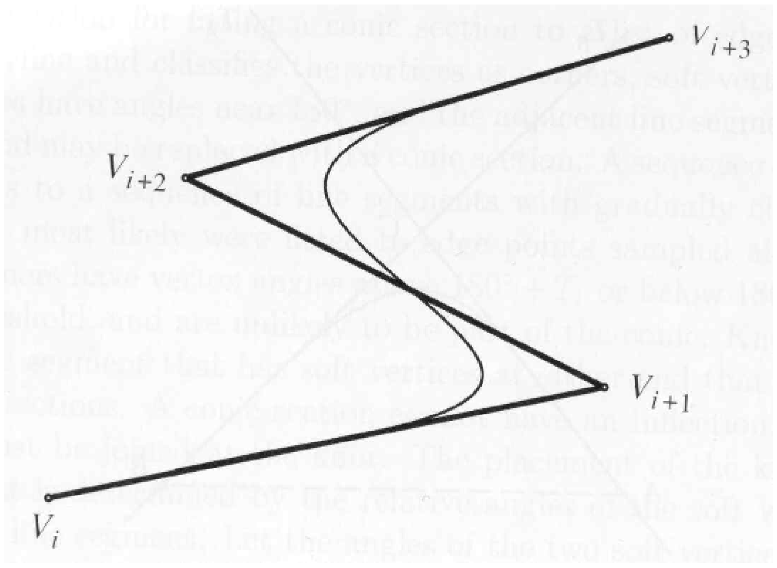
(6) Advance the window to the next three polygon vertices after the end of the circular arc and return to step 2.

## • Conic sections and cubic splines

- Conic sections correspond to hyperbolas, parabolas, and ellipses (2nd degree polynomials).

- Cubic splines correspond to 3rd degree polynomials.

- Conic sections and cubic splines allow more complex contours to be represented using fewer segments.



# Contour representation using curve fitting (approximation)

- Higher accuracy can be obtained by computing an approximation that is not forced to pass through particular edge points.

- Approximation-based methods use all the edge points to find a good fit (this is contrast to the previous methods that use only the interpolated edge points).

- Methods to approximate curves depend on the reliability with which edge points can be grouped into contours.

  (1) Least-squares regression can be used if it is certain that the edge points grouped together belong to the same contour.

  (2) Robust regression is more appropriate if there are some grouping errors.

## • Noise-free case

- Suppose the curve model we are trying to fit is described by the equation

$$f(x, y, a_1, a_2, \ldots, a_p) = 0$$

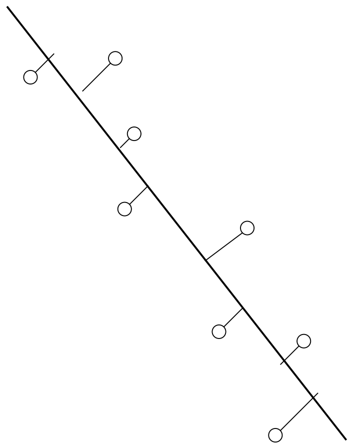
- In the noise-free case, it suffices to use $p$ edge points to solve for the unknown parameters $a_1$, $a_2$, ..., $a_p$.

- In practice, we need to use all the edge points available to obtain good parameter estimates.

## • Least-squares line fit

- Linear regression fits a line to the edge points by minimizing the sum of squares of the perpendicular distances of the edge points from the line being fit.

$$\mathrm{X}^2 = \sum_{i=1}^{n} (x_i \sin\theta - y_i \cos\theta + \rho)^2$$



- The solution to the linear regression problem is

$$\rho = \bar{y}\cos\theta - \bar{x}\sin\theta$$

where

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i,$$

and

$$\tan\theta = \frac{a}{b+c}$$

where

$$a = 2\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}),$$

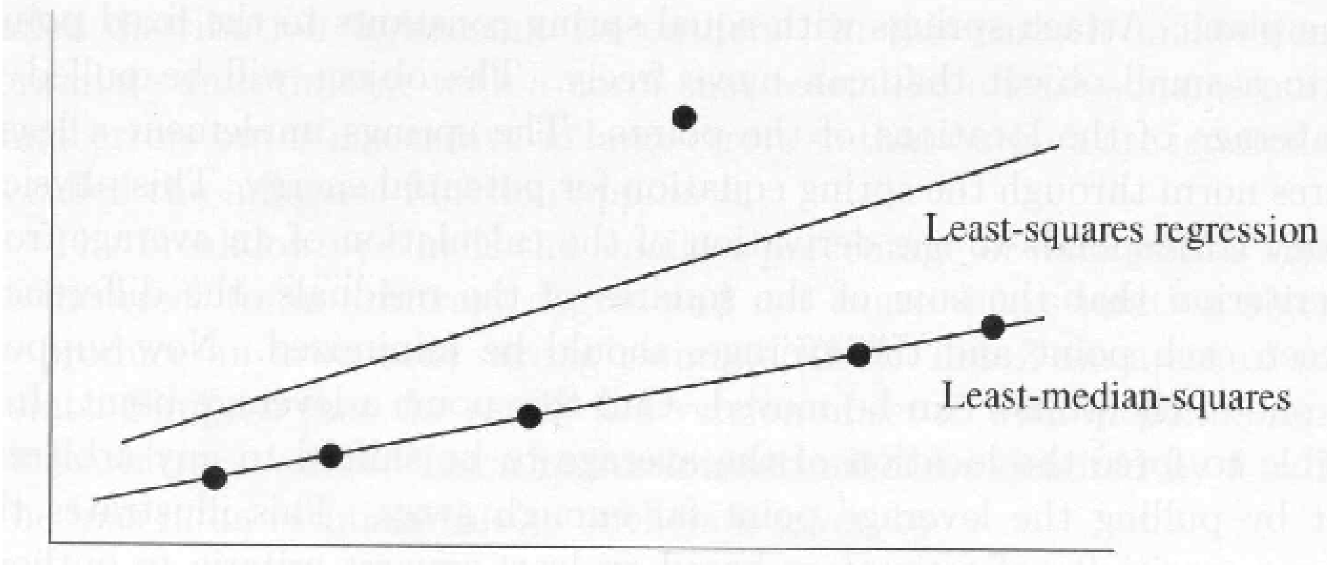
$$b = \sum_{i=1}^{n}(x_i - \bar{x})^2 - \sum_{i=1}^{n}(y_i - \bar{y})^2$$

$$c = \sqrt{a^2 + b^2}$$

- Some problems with linear regression:

  * It works well for cases where the noise follows a Gaussian distribution.

  * It does not work well when there are outliers in the data (e.g., due to errors in edge linking ).



## • **Robust regression methods**

- Outliers can 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.

Iterative fitting

```
(1) Fit the line to the data set.

(2) Remove the point with the worst error.

(3) If the line parameters do not change significantly, quit, else repeat from step 1
```

Least-median of squares
(assume that there are *n* data points and *p* parameters in the 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.

(4) Repeat until the median squared error is less than some threshold, or take the
    best median error result after some number of runs.
```

- This algorithm can handle up to 50% outliers in the data set.

# Contour representation at multiple scales

- The curvature scale space (CSS) approach has been introduced as a shape representation for planar curves.

- The idea is finding points of inflection (i.e, curvature zero-crossings) on the curve at varying levels of detail.

- The curvature $k$ of a planar curve, can be expressed as follows:

$$k(t) = \frac{\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)}{(\dot{x}(t)^2 + \dot{y}(t)^2)^{3/2}}$$

- To compute the curvature of the curve at varying levels of detail, $x(t)$ and $y(t)$ are convolved with a Gaussian function g(t,$\sigma$).

- Defining $X(t, \sigma)$ and $Y(t, \sigma)$ as th e convolution of $x(t)$ and $y(t)$ respectively with the Gaussian function, the smoothed curve curvature $k(t, \sigma)$ can be expressed as follows:

$$k(t, \sigma) = \frac{X_t(t, \sigma)Y_{tt}(t, \sigma) - X_{tt}(t, \sigma)Y_t(t, \sigma)}{(X_t(t, \sigma)^2 + Y_t(t, \sigma)^2)^{3/2}}$$

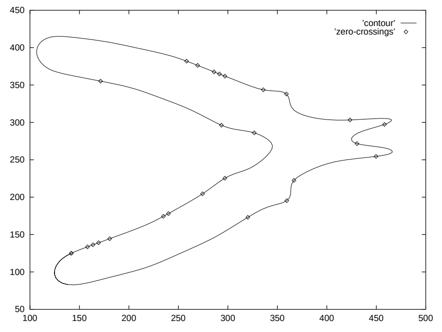
where $X_t(t, \sigma)$ and $X_{tt}(t, \sigma)$ are defined as:

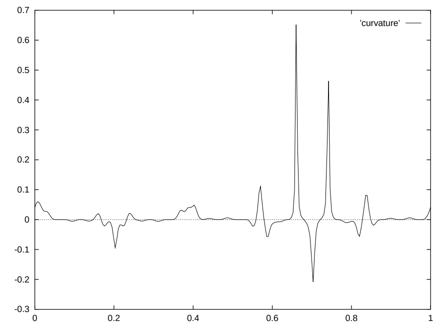
$$X_t(t, \sigma) = x(t) * \frac{\partial g(t, \sigma)}{\partial t}$$

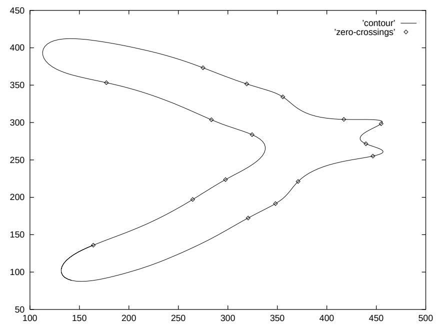
$$X_{tt}(t, \sigma) = x(t) * \frac{\partial^2 g(t, \sigma)}{\partial t^2}$$

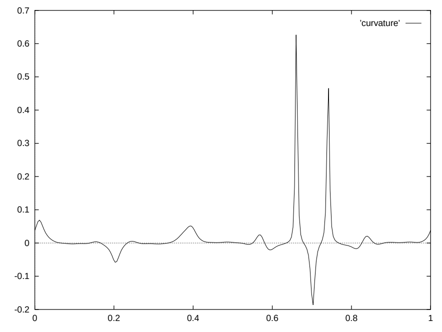
- $Y_t(t, \sigma)$ and $Y_{tt}(t, \sigma)$ are defined in a similar manner.

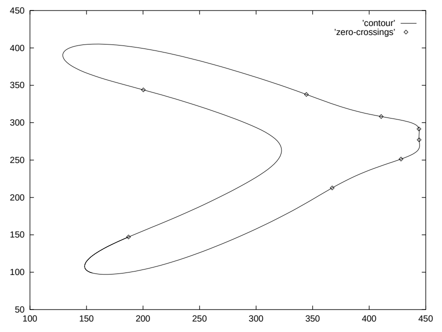
- The function defined implicitly by $k(t, \sigma) = 0$ is the CSS image of the curve.

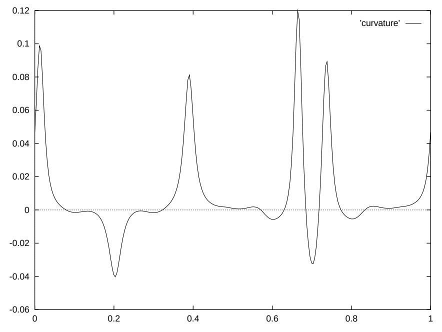

'contour'
'zero-crossings'

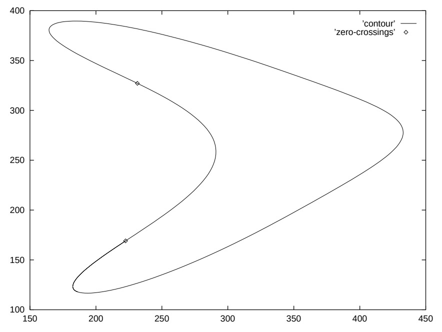

'curvature'

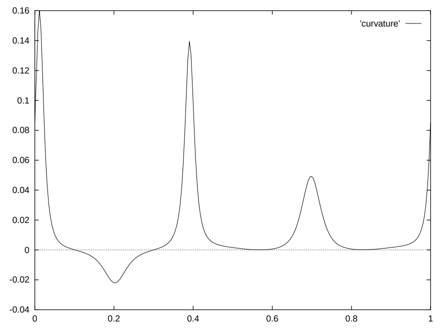

'contour'
'zero-crossings'

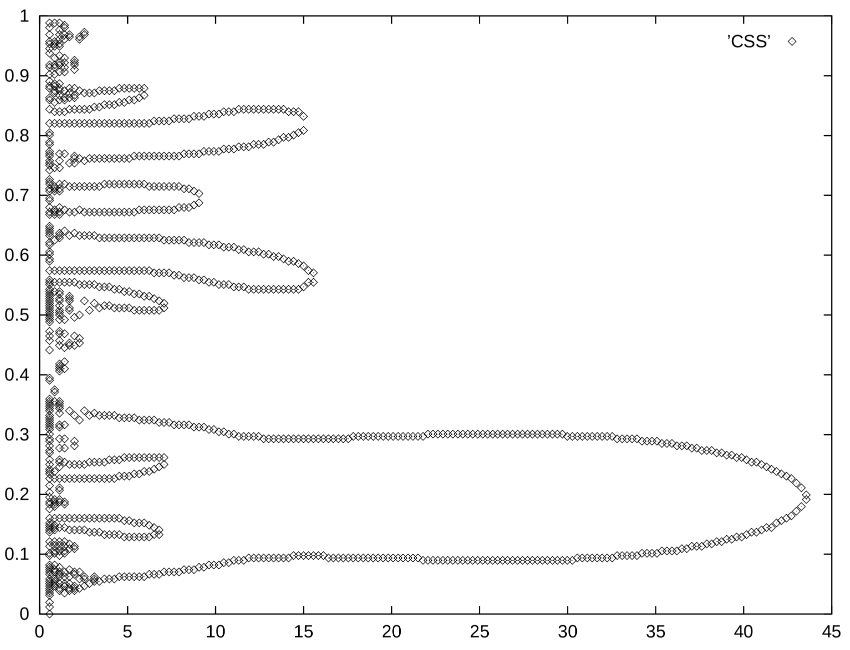

'curvature'


'contour'
'zero-crossings'

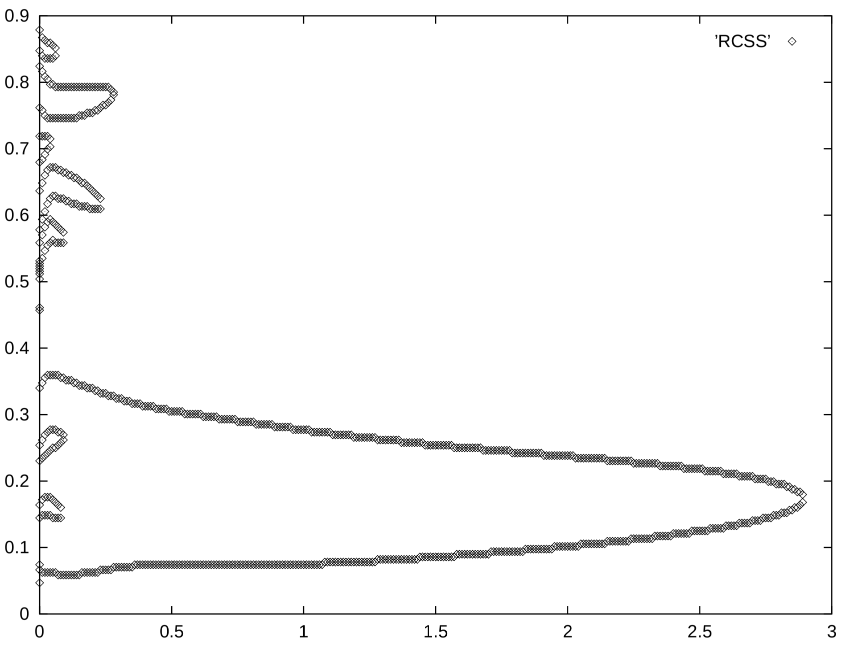

'curvature'


'contour'
'zero-crossings'


'curvature'



'CSS'
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20
25
30
35
40
45



'RCSS'
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
0.5
1
1.5
2
2.5
3

## Contour Description

- Extract a number of features (descriptors).

- Descriptors can be used for object recognition.

- Good descriptors are invariant to geometrical transformations (i.e., translation, rotation, and scaling).