Edge based segmentation

- relies on edges found in an image by edge detection operations
- image processed by edge detector not good enough for segmentation
- supplementary processing to combine edges into edge chains that correspond better with boundaries

edge segmentation algorithms differ by the amount of prior information they use.

no information

- edge image thresholding
  - thick edges so use non-maximal suppression of direction edge data.
  - use dual thresholds
- edge relaxation
- boundary tracing – need to know regions.
- graph searching – use as much prior knowledge as possible.
  - smoothness
  - low curvature

  } some cost function
  need to know start and stop
- dynamic programming - optimization of a boundary
- Hough transforms – need to know shape info
- border detection using border location information (deformable models) – needs rough shape

lots of information

- region construction from borders.

- **Edge-based image segmentation**

Sonka

  - Edge-based segmentation relies on edges found in an image by edge detecting operators—these edges mark image locations of discontinuities in gray-level, color, texture, etc.
  - The most common problems of edge-based segmentation, caused by image noise or unsuitable information in an image, are an edge presence in locations where there is no border, and no edge presence where a real border exists.
  - **Edge image thresholding** is based on construction of an edge image that is processed by an appropriate threshold. (Canny).
  - In **edge relaxation**, edge properties are considered in the context of neighboring edges. If sufficient evidence of the border presence exists, local edge strength increases and vice versa. Using a global relaxation (optimization) process, continuous borders are constructed.
  - Three types of region borders may be formed: **inner**, **outer**, and **extended**. The inner border is always part of a region, but the outer border never is. Therefore, using inner or outer border definition, two adjacent regions never have a common border. Extended borders are defined as single common borders between adjacent regions still being specified by standard pixel co-ordinates.

inner & outer have problems in that they overlap or don't touch.

  - If the criterion of optimality is defined, globally optimal borders can be determined using **(heuristic) graph searching** or **dynamic programming**. Graph-search-based border detection represents an extremely powerful segmentation approach.

The border detection process is transformed into a search for the optimal path in the weighted graph. Costs are associated with each graph node that reflect the likelihood that the border passes through the particular node (pixel). The aim is to find the optimal path (optimal border, with respect to some objective function) that connects two specified nodes or sets of nodes that represent the border's beginning and end.

  - **Cost definition** (evaluation functions) is the key to successful border detection. Cost calculation complexity may range from simple inverted edge strength to complex representation of a priori knowledge about the sought borders, segmentation task, image data, etc.
  - Graph searching uses Nilsson's **A-algorithm** and guarantees optimality. **Heuristic graph search** may substantially increase search speed, although the heuristics must satisfy additional constraints to guarantee optimality.
  - **Dynamic programming** is based on the principle of optimality and presents an efficient way of simultaneously searching for optimal paths from multiple starting and ending points.
  - Using the A-algorithm to search a graph, it is not necessary to construct the entire graph since the costs associated with expanded nodes are calculated only if needed. In dynamic programming, a complete graph must be constructed.
  - If calculation of the local cost functions is computationally inexpensive, dynamic programming may represent a computationally less demanding choice. However, which of the two graph searching approaches (A-algorithm, dynamic programming) is more efficient for a particular problem depends on the evaluation functions and on the quality of heuristics for an A-algorithm.
  - **Hough transform** segmentation is applicable if objects of known shape are to be detected within an image. The Hough transform can detect straight lines and curves (object borders) if their analytic equations are known. It is robust in recognition of occluded and noisy objects.
  - The generalized Hough transform can be used if the analytic equations of the searched shapes are not available; the parametric curve (region border) description is based on sample situations and is determined in the learning stage.
  - While forming the regions from complete borders is trivial, **region determination from partial borders** may be a very complex task. Region construction may be based on probabilities that pixels are located inside a region closed by the partial borders. Such methods do not always find acceptable regions but they are useful in many practical situations.

Algorithm 5.1 Canny Edge Detection — Overview

1. Smooth image with a Gaussian filter
2. compute gradient magnitude & orientation using finite-difference approximations
3. apply non maxima suppression to the gradient magnitude
4. use double thresholding algorithm to detect and link edges.

5.6.1. Canny Edge Detector

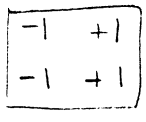
(1) First, smooth the image: $S[i,j] = G[i,j;\sigma] \star I[i,j]$

Gaussian

Smoothed image

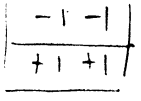
(2) second, compute gradient using 2x2 first difference approximations

$$P[i,j] \approx \frac{1}{2}\left(S[i+1,j] - S[i,j] + S[i+1,j+1] - S[i,j+1]\right)$$



$$Q[i,j] \approx \frac{1}{2}\left(S[i,j] - S[i,j-1] + S[i+1,j] - S[i+1,j-1]\right)$$

This yields computed derivatives at $\left[i+\frac{1}{2}, j+\frac{1}{2}\right]$



Now, convert to magnitude and direction.

$$m[i,j] = \sqrt{P[i,j]^2 + Q[i,j]^2}$$

$$\theta[i,j] = \arctan\left(\frac{Q[i,j]}{P[i,j]}\right)$$

can be done mostly in integer by look up methods.

Third, now do
Non-maxima suppression (3)

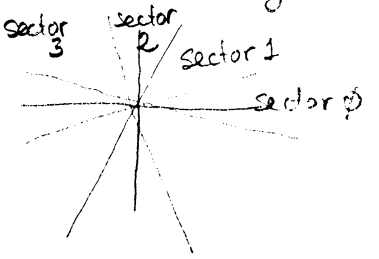
$m[i,j]$ will have large values where gradient is large.
still need to find local maxima in this array
to locate edges.

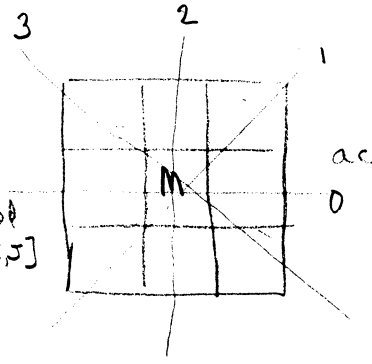
→ must thin so only points of greatest local change remain

This is done by

(1) reduce angle θ to one of four sectors.

$$\zeta[i,j] = \text{Sector}(\theta[i,j])$$





(2) pass a 3×3 neighborhood across $M[I,J]$ across $m[i,j]$.

(3) compare m to the two neighbors in direction specified by $\zeta[i,j]$, i.e. the angle of greatest change.

(4) If $m[i,j]$ not larger than elements on either side in sector, then $m[i,j] = 0$.

→ denote this entire process $N[i,j] = nms(m[i,j], \zeta[i,j])$

this image will still contain many false edge fragments caused by noise & fine texture.

(4) Fourth, do double thresholding.

Typically threshold $N[i,j]$

a single threshold usually is hard to achieve

Use double thresholding

use $N[i,j]$ as input

use T_1 and $T_2 \approx 2T_1$ to produce two thresholded images $T_1[i,j]$ and $T_2[i,j]$. basically it must be connected

link into contours
If a gap is encountered go to T_1 until bridged to edge in T_2.

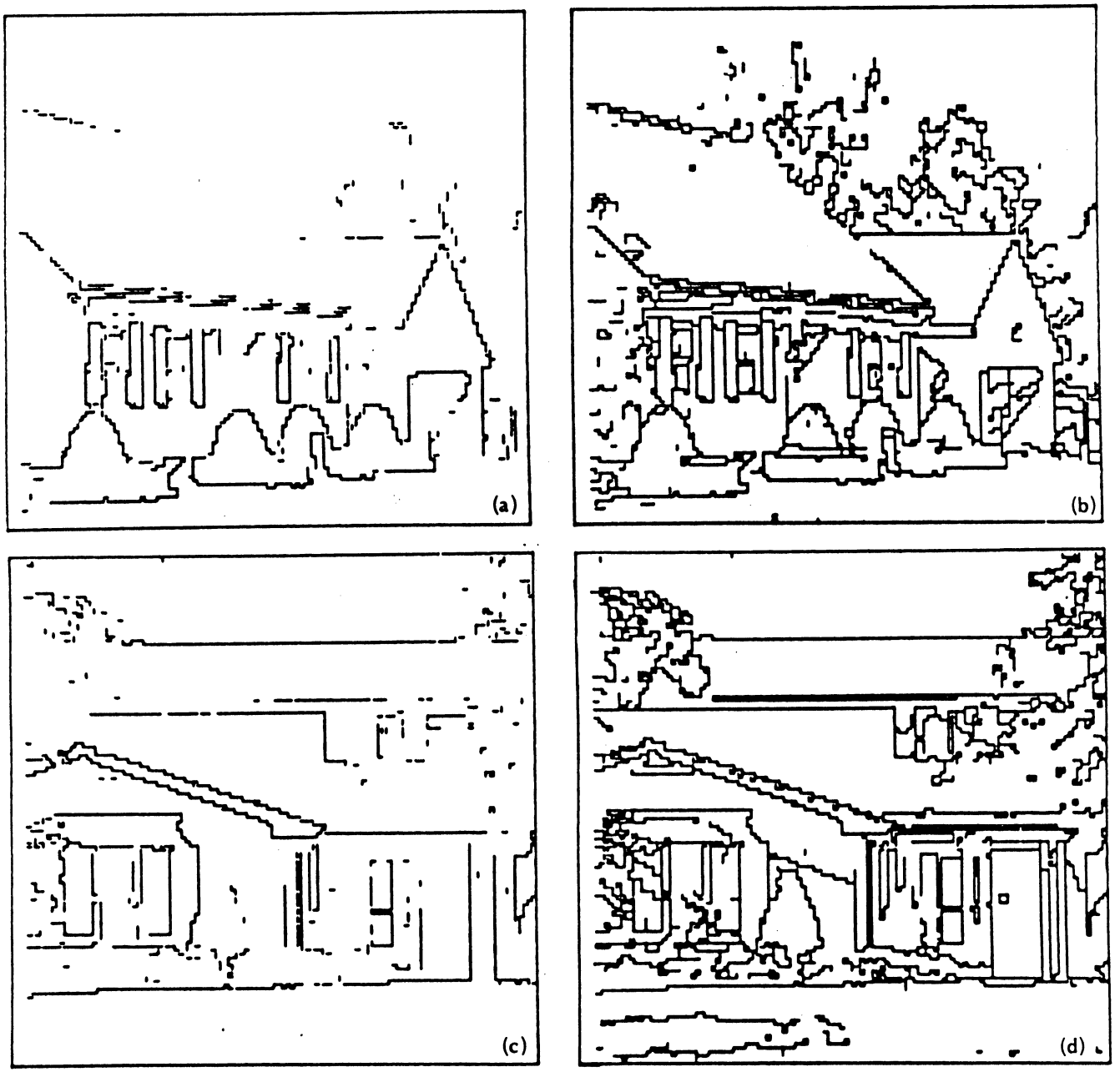


Fig. 3.22 Edge relaxation results. (a) Raw edge data. Edge strengths have been thresholded at 0.25 for display purposes only. (b) Results after five iterations of relaxation applied to (a). (c) Different version of (a). Edge strengths have been thresholded at 0.25 for display purposes only. (d) Results after five iterations of relaxation applied to (c).

edge relaxation – improve edge operator estimate by re-adjusting edge estimate based upon local information

algorithm:

$c^0(e)$ = of any edge.

$$c^0(e) = \frac{\text{gradient magnitude}}{\text{maximum gradient amplitude in image}}$$

higher is better

initial confidence of edge is normalized gradient

$k = 1$

while any $c^k(e) \neq (0 \text{ or } 1)$ do

begin

classify edge: edge_type = f(edge_ neighbors)

adjust confidence: $c^k(e) = f(\text{edge\_type}, c^{k-1}(e))$

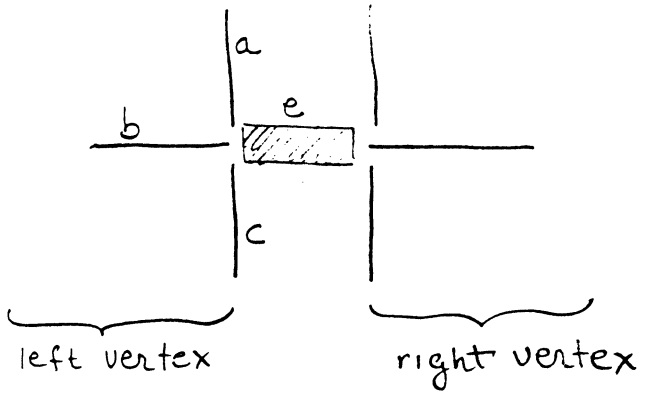
iteration counter: $k = k + 1$

end

algorithm forces all points in edge image to converge to 0 or 1

original relaxation work done by Hanson & Riseman
Prager

edge type classified by vertex type



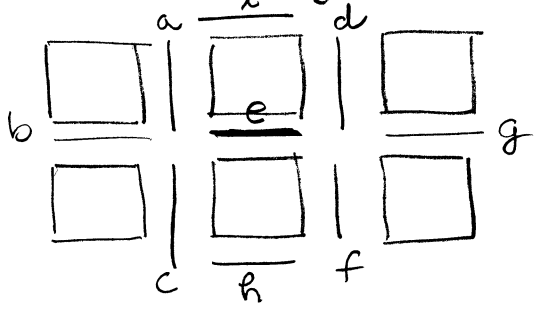
e = edge to be updated

a, b, c are normalized gradient magnitudes

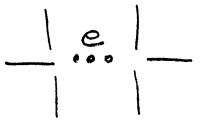
$m = \max(a, b, c, q)$

q = constant (a threshold), often = 0.1

basic rules for evaluating crack edges.



following situations



0-0 isolated edge, negative influence on edge confidence

0-1 uncertain, week positive, or no influence on edge confidence

0-2, 0-3 dead end, negative influence on edge confidence

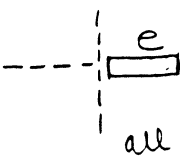
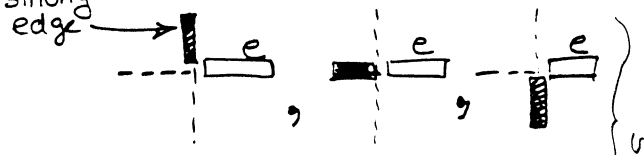
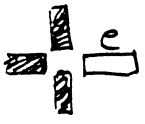
1-1 continuation, strong positive influence on edge confidence

1-2, 1-3 continuation to border intersection — medium positive influence on edge confidence

2-2, 2-3, 3-3 bridge between borders — not necessary for segmentation, no influence on edge confidence.

left vertex types and associated confidences

suppose confidence originally 0.9

type	figure	confidence	Example: strong = 0.8, weak = 0.1, m = 0.9
0	 all weak edges	$(m-a)(m-b)(m-c)$	$(.9-.1)(.9-.1)($ .51
1	 basically one strong edge	$a(m-b)(m-c)$	$.8(.8)(.8)$ .51
2	 basically two strong edges	$a\,b\,(m-c)$	.39
3	 basically three strong local edges	$a\,b\,c$	.34

strength of edges

If all weak decrease confidence.

increase confidence

increasing strength (confidence)

ambiguous so keep about same

Not sure if example correct. Seems to be backwards

actual edge type is concatenation of left and right vertex types

edge (e) = (i, j) with 3,3 being the strongest edge.

to update edge confidence in algorithm

increment : $C^k(e) = \min(1, C^{k-1}(e) + \delta)$

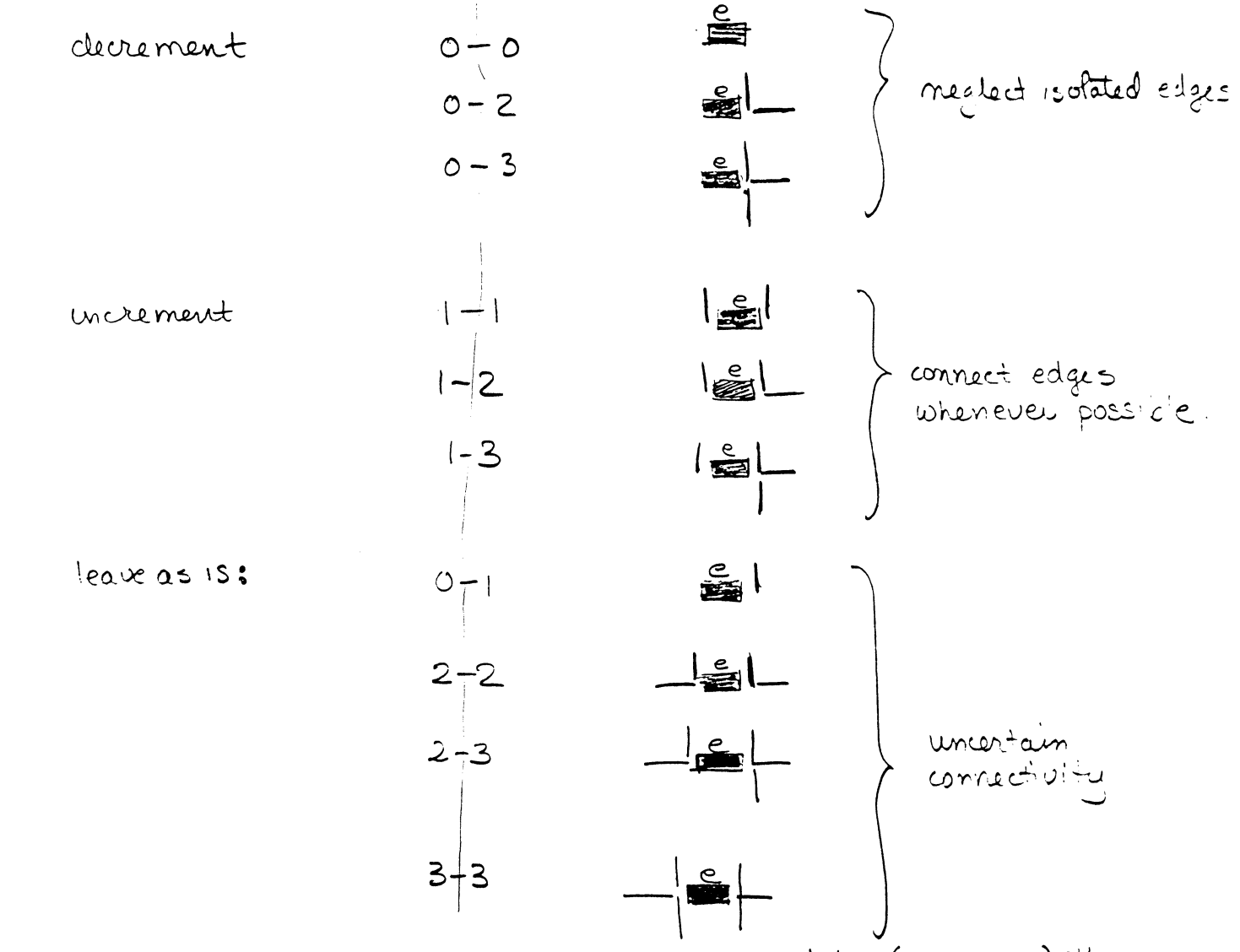
decrement : $C^k(e) = \max(0, C^{k-1}(e) - \delta)$

leave as is : $C^k(e) = C^{k-1}(e)$

where $0.1 < \delta < 0.3$ typically. Pick dependent on problem.

jump to here

examples : leftvertex – right vertex



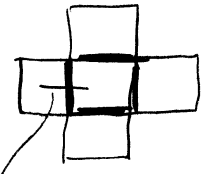
The idea is. if a strong edge is near by update (increase) the confidence of nearby edges. As the number of nearby edges increases

5.2.3 Border tracing
assumes binary image or labeled regions

(global)
border ≡ set of pixels within the region R that have one or more neighbors outside R. — border of the object
(this is called an inner border)
you can also define an outer border based upon the border of the background

(local)
edge — local property of a pixel an its immediate neighborhood

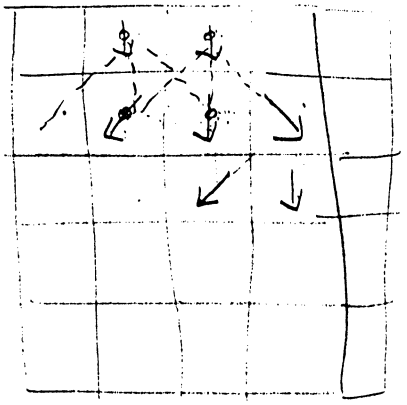
crack edges



difference in properties between pairs of pixels
direction is in that of increasing brightness, multiple of 90°.

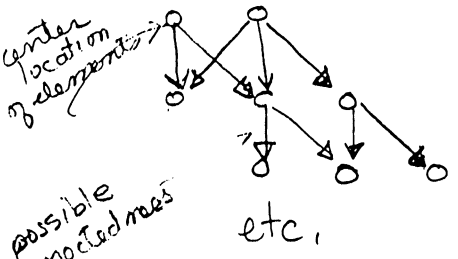
edge following as graph searching:

8 connected (!)

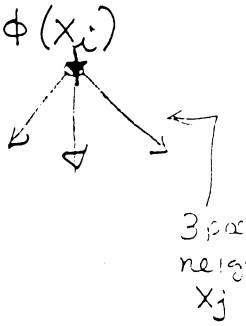


direction image $\phi(\underline{x})$.

graph



- basic template



graph consists of a set of **nodes** $\{n_j\}$ and arcs between nodes $\langle n_i, n_j \rangle$

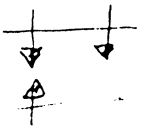
sample rules.

for an arc to connect from $\underline{x}_i \rightarrow \underline{x}_j$

1. $\underline{x}_j$ must be one of the three possible eight-neighbors in front of the contours direction $\phi(\underline{x}_i)$

2. and $s(\underline{x}_i) > T$, $s(\underline{x}_j > T$ $\quad s$ = edge strength.

3. $\left| \left\{ \left[\phi(\underline{x}_i) - \phi(\underline{x}_j) \right] \bmod 2\pi \right\} \right| < \frac{\pi}{2}$.



don't allow these cases.

can add a curvature/smoothness constraint, ϕ is the angle of the edge and s is its strength.

a boundary of an object is simply a lowest-cost search between two nodes of a weighted graph.

how do you get these nodes?

Use heuristic search (basically a tree search)

express evaluation function $f(x_i) = s(x_i) + h(x_i)$

where $f(x_i)$ = estimate of the ~~optimal~~ cost of the path from x_A to x_B constrained to go through x_i

$s(x_i)$ = estimated cost of x_A to x_i

$h(x_i)$ = estimated cost of x_i to x_B, the goal

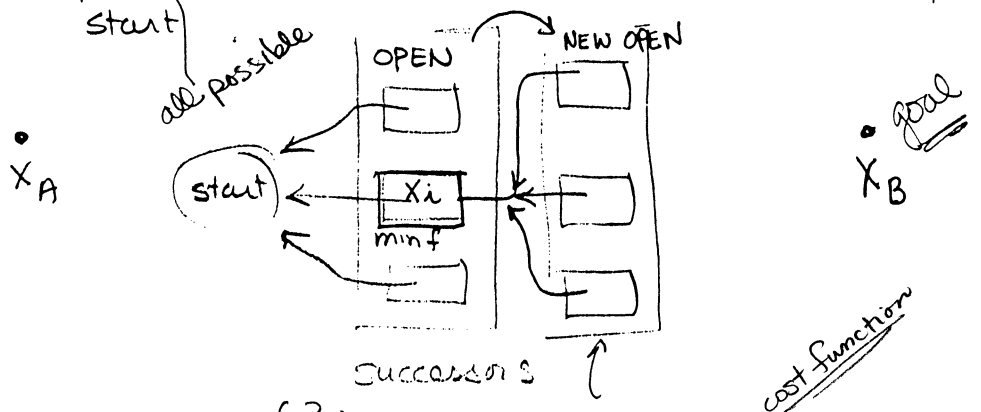
this is often just a distance function.

minimize overall cost of curve.

Algorithm 4.4 Heuristic Search (A algorithm)

expand START

1. put successors on a list called OPEN with pointers back to start



Sort of like original graph search except not geometrically constrained

2. If OPEN = { } then FAILED.
   remove node X_i of minimum f from OPEN.

Check it. If $X_i = X_B$ stop. Trace back to find optimum path and stop.

3. else expand x_i, putting successors on OPEN with pointers back to X_i. Go to 2.

$$f(x_i) = s(x_i) + h(x_i)$$

If $h(x_i) = 0 \ \forall \ i$ search is minimum-cost.

Other $h(x_i)$ can be used.

Problem: gaps may terminate boundary

4.4.1 Good evaluation functions

max. edge in picture

edge value at that point

- edge strength — cost of adding an edge = $M - s(\underline{x})$ where $M = \max_{\underline{x}} s($
- curvature — diff $[\phi(x_i) - \phi(x_j)]$
  ↑ measures angle between edge elements.
- proximity to an a priori boundary — dist (x_i, B)
  ↑ min. distance
  favor points near goal
  $d(\underline{x}_i, \underline{x}_{GOAL})$
  ↑ plain old distance.
- estimates of a distance to a goal

4.5 Edge following as a dynamic program

dynamic programming — technique for solving optimization problems when not all variables in the evaluation function are interrelated simultaneously

Wilson will do

Example $\max_{x_i} h(x_1, x_2, x_3, x_4)$

if nothing is known about h must try all x_i

On the other hand, if $h(\ ) = h_1(x_1, x_2) + h_2(x_2, x_3) + h_3(x_3, x_4)$

maximize over x_1 in h_1 and tabulate the best value of $h_1(x_1, x_2)$ for each x_2:

i.e. $f_1(x_2) = \max_{x_1} h_1(x_1, x_2)$

since h_2 and h_3 do not depend upon x_1, they can be temporarily ignored.

Now eliminate x_2 by computing $f_2(x_3)$ as

$$f_2(x_3) = \max_{x_2} \left[f_1(x_2) + h_2(x_2, x_3) \right].$$

basically you do maximization one variable at a time.

and continue with x_3

$$f_3(x_4) = \max_{x_3} \left[f_2(x_3) + h_3(x_3, x_4) \right]$$

so that

$$\max_{x_i} h = \max_{x_4} f_3(x_4)$$

generalizing to N variables

$$f_{n-1}(x_n) = \max_{x_{n-1}} \left[f_{n-2}(x_{n-1}) + h_{n-1}(x_{n-1}, x_n) \right]$$

$$\max_{x_i} h(x_i, \ldots, x_N) = \max_{x_N} f_{N-1}(x_N)$$

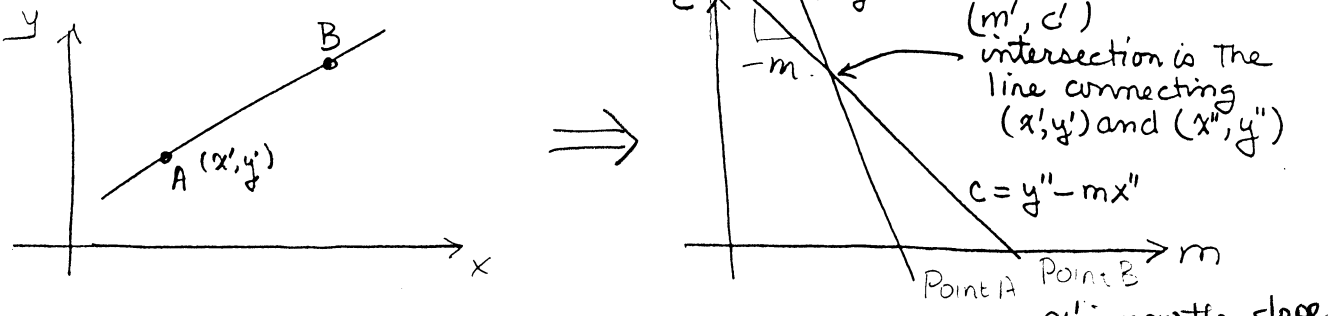
after all earlier variables optimized

4.3 The Hough Method for curve detection (parametric estimator)

transform to a parametric space (i.e. curve)

relatively unaffected by gaps in curves or noise

{ organizes tentative edge points into lines



$y = mx + c$.

now re-write

$$c = y - mx = -xm + y$$

now plot

$-x$ = slope, y = y-intercept

$y = mx + c$

this is m-c space (parameter space.

x' is now the slope

y is now intercept

for point x', y' we have $c = y' - mx'$

draw line

x' fixed

for point x'', y'' we might have.

$c = y'' - mx''$

draw line

Important observation:

ALL points on the line A-B will transform to lines which intersect at (m', c') — the slope and intercept of $\overline{AB}$

Didn't publish it — patented it!

Algorithm: 4.1 (B&B)

1. quantize parameter space between appropriate maxima and minima for c & m (i.e. set up a 2-D or 3-D array), i.e. A(m, n

2. initialize the accumulator array $A(c,m) := 0$

3. For each point (x,y) in the edge enhanced image such that $E(x,y) > T$, increment all points in $A(c,m)$ along the appropriate line in m-c space

   i.e. $A(c,m) := A(c,m) + 1 \quad \forall \; c = y - mx$. (quantized)

   Note: This is time and space consuming. Each thresholded point might generate, say 102$\frac{1}{4}$ points, in $A(c,m)$.

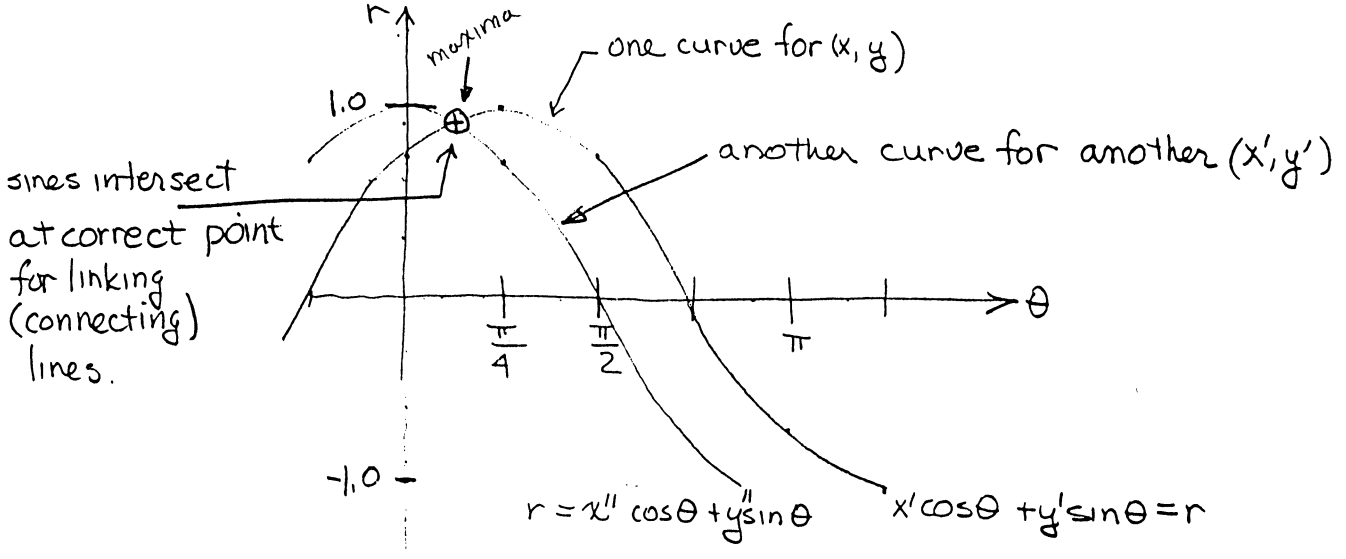
4. Local maxima in $A(c,m)$ correspond to collinear points (i.e. lines) in the image array. Values in $A(c,m)$ correspond to how many points exist on that line.

Problem: m is unbounded $-\infty < m < +\infty$

Solution: convert to polar coordinates transforming.

i.e. $r = x\cos\theta + y\sin\theta$

This gives sine waves instead of straight lines in Hough space



the Hough transform can be applied to any curve of the form

$$f(\underline{x}, \underline{a}) = 0$$

$\underline{x}$: position vector; $\underline{a}$: parameter vector

For example, $(x-a)^2 + (y-b)^2 = r^2$
is a three parameter space (a,b,r)

This approach is impractical for too many parameters n-dimensional space

Effectively it is a matched filtering process

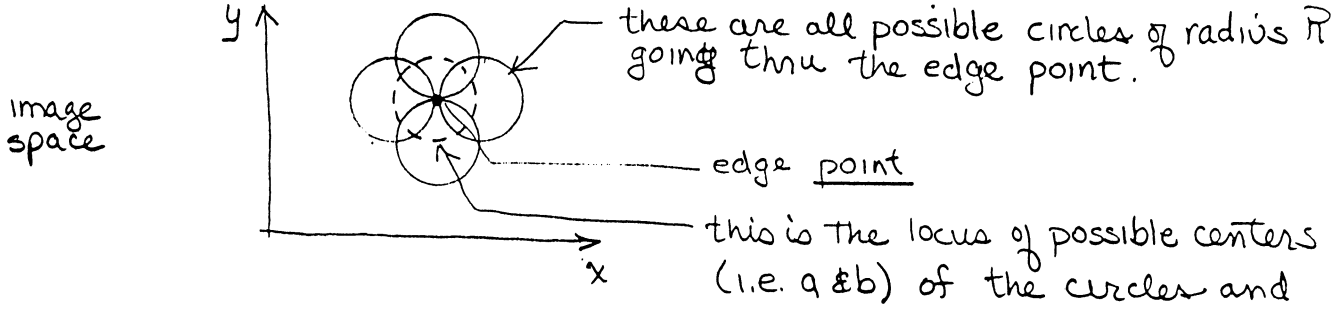
Consider looking for a circle of 1's (edge only)

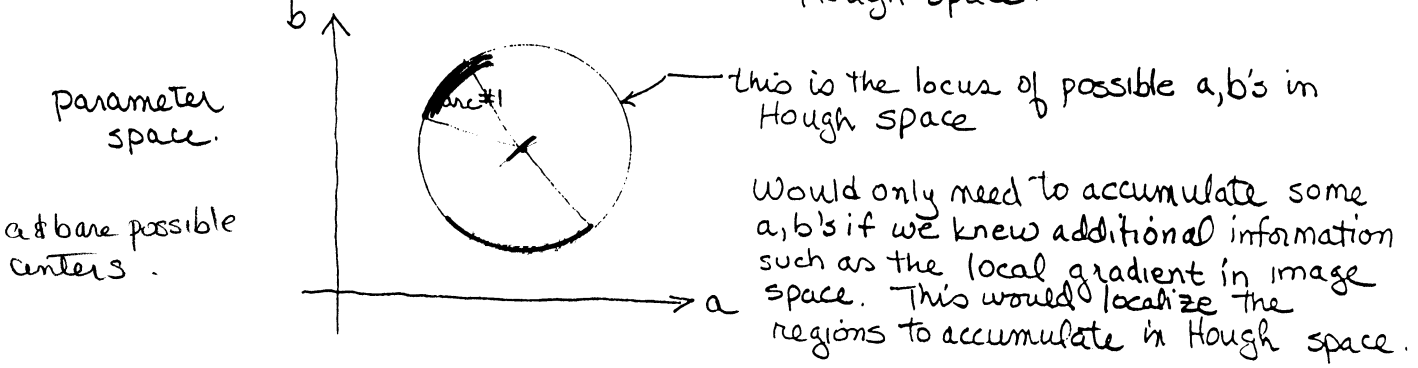
A (a,b,r) is the correlation with that circle template.
- (a,b): possible centers
- r: possible radii

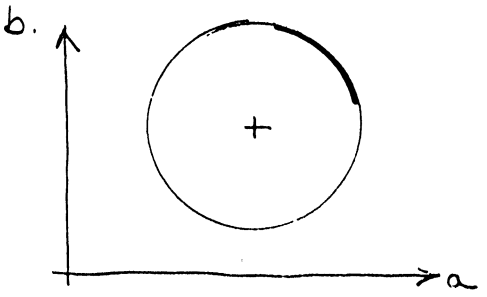
SPECIAL techniques needed to find end points of a line segment.

4.3.1. The Hough transform is traditionally slow.
Use gradient direction to reduce computations.

→ Example: find a circle of fixed radius R







how to use gradient information to localize circle.

Consider equation of circle

$$(x-a)^2 + (y-b)^2 = R^2$$

differentiating $2(x-a)\,dx + 2(y-b)\,dy = 0.$

$$\frac{dy}{dx} = -\frac{x-a}{y-b} \qquad (1)$$

define $\frac{dy}{dx} = \tan\phi$ (the gradient of the edge in image space).

Solve for a & b in terms of x, y, r and ϕ

r — radius of circle we are looking for

eqn. of circle $(x-a)^2 + (y-b)^2 = r^2$

substituting (1) $\left[\frac{dy}{dx}(y-b)\right]^2 + (y-b)^2 = r^2$

$$(y-b)^2\left[\left(\frac{dy}{dx}\right)^2 + 1\right] = r^2$$

$$(y-b)^2\left[\tan^2\phi + 1\right] = r^2 = (y-b)^2 \frac{1}{\cos^2\phi} = r^2$$

$$\therefore (y-b)^2 = r^2\cos^2\phi$$

$$y-b = \pm r\cos\phi$$

$$b = y \mp r\cos\phi$$

known diameter of circle you are looking for

computed slope.

in terms of x, y, r, ϕ.

in House space

from image space

similarily for a using the equation of the circle and (1)

$$(x-a)^2 + (y-b)^2 = r^2$$

$$(x-a)^2 + \left[\frac{x-a}{\frac{dy}{dx}}\right]^2 = r^2$$

$$(x-a)^2 \left[1 + \frac{1}{\frac{dy}{dx}}\right] = r^2$$

$$(x-a)^2 \left[1 + \cot^2\phi\right] = r^2 = (x-a)^2 \frac{1}{\sin^2\phi}$$

$$(x-a)^2 = r^2 \sin^2\phi$$

$$a = x \mp r\sin\phi.$$

look for circles of radius R.

$\Rightarrow$ Use $\boxed{\begin{aligned} a &= x \mp R\sin\phi \\ b &= y \mp R\cos\phi \end{aligned}}$

R is known in this example
ϕ is known from gradient in image
(x,y) is location of edge candidate.

Although this gives a single point in (a,b) space. Do an arc thru (a,b) to cover errors in ϕ due to quantization, noise, etc.

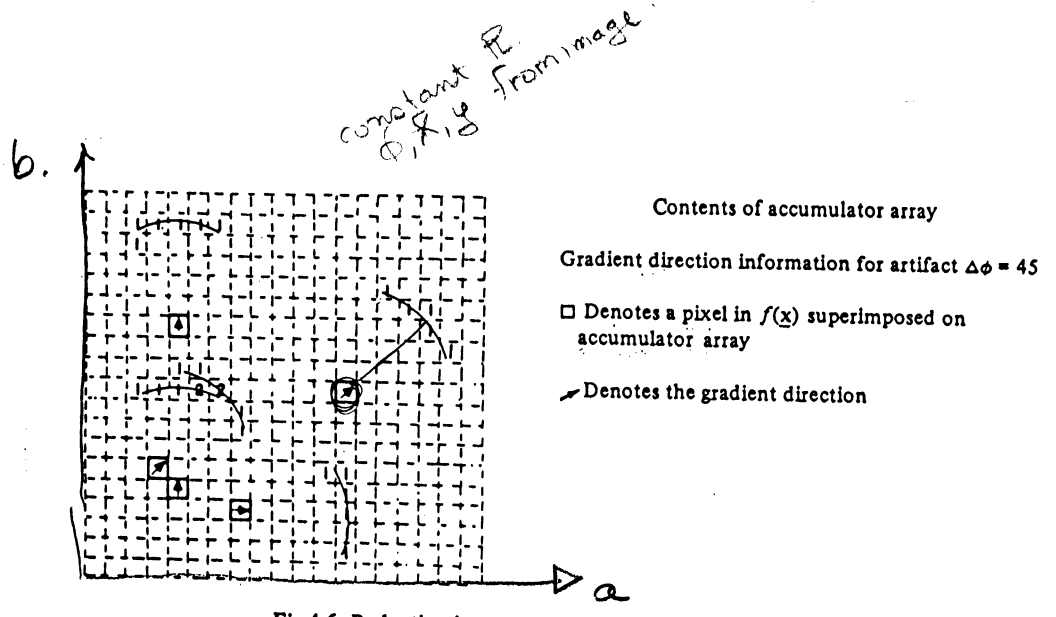
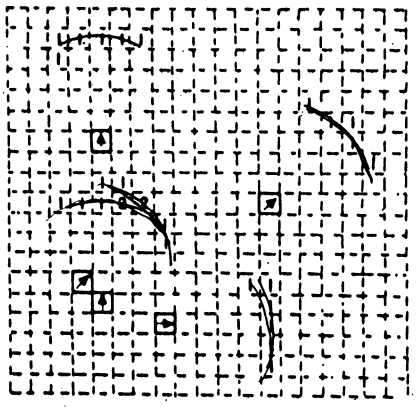


Fig 4.6 Reduction in computation with gradient information



Contents of accumulator array

Gradient direction information for artifact $\Delta\phi = 45$

□ Denotes a pixel in $f(\underline{x})$ superimposed on accumulator array

Denotes the gradient direction

Fig 4.6 Reduction in computation with gradient information

This is for a circle.

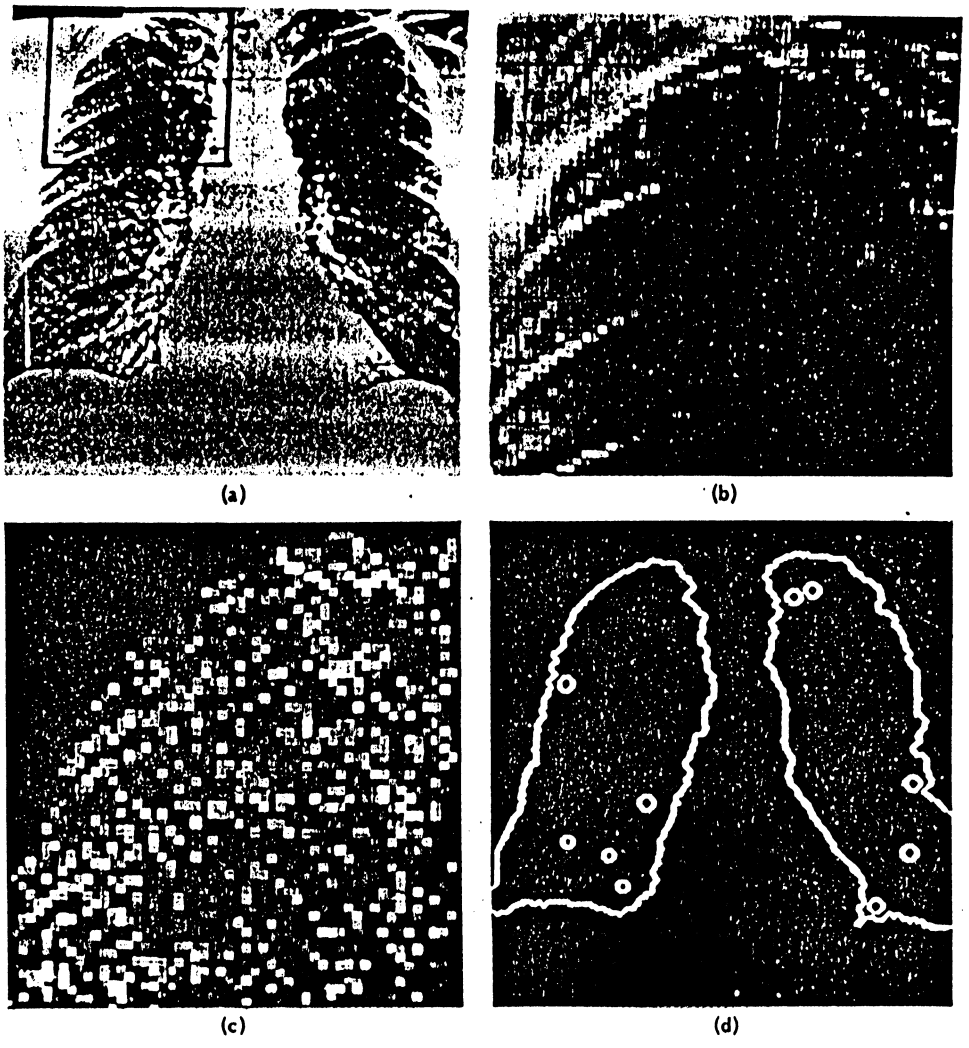
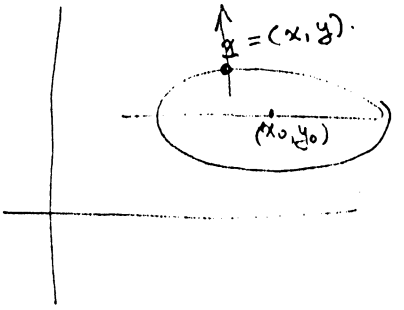


Fig. 4.7 Using the Hough technique for circular shapes. (a) Radiograph. (b) Window. (c) Accumulator array for $r = 3$. (d) Results of maxima detection.

4.3.3 Trading off work in parameter space for work in image space



$$\frac{(x - x_0)^2}{a^2} + \frac{(y - y_0)^2}{b^2} = 1$$ ellipse

four parameters x_0, y_0, a, b. ← four-space

Ellipses

$$\frac{2(x - x_0)}{a^2}\,dx + \frac{2(y - y_0)}{b^2}\,dy = 0$$

$$\frac{x - x_0}{a^2} + \frac{y - y_0}{b^2}\,\frac{dy}{dx} = 0.$$

Algorithm 4.2. Hough transform applied to ellipses

For each (x, y), increment the point in parameter space given by (a, b, x_0, y_0) where.

in image space.

measured

$$\begin{cases} x = x_0 \pm \dfrac{a}{\sqrt{1 + \frac{b^2}{a^2}\tan^2\phi}} \\ y = y_0 \pm \dfrac{b}{\sqrt{1 + \frac{a^2}{b^2}\tan^2\phi}} \end{cases}$$ solve these for unmeasured.

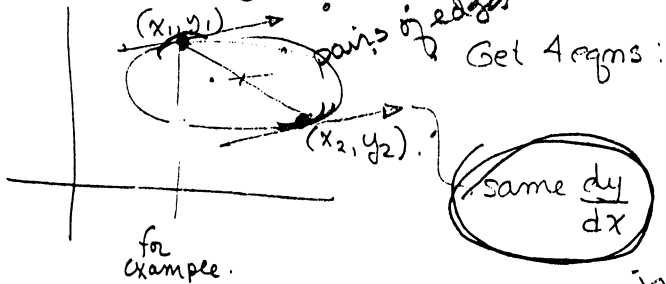
should derive the equation to be sure.

increment $A(a, b, x_0, y_0) := A(a, b, x_0, y_0) + 1$

for any set of points if a and b have m values there will be $O(m^2)$ solutions.

pairwise curves.

Suppose we only look at pairwise combinations of edge elements (good for ellipses, etc.



Get 4 eqns:

$$\begin{cases} \dfrac{(x_1 - x_0)^2}{a^2} + \dfrac{(y_1 - y_0)^2}{b^2} = 1 \\ \dfrac{(x_2 - x_0)^2}{a^2} + \dfrac{(y_1 - y_0)^2}{b^2} = 1 \end{cases}$$

derivatives

$$\begin{cases} \dfrac{x_1 - x_0}{a^2} + \dfrac{y_1 - y_0}{b^2}\,\dfrac{dy}{dx} = 0 \\ \dfrac{x_2 - x_0}{a^2} + \dfrac{y_2 - y_0}{b^2}\,\dfrac{dy}{dx} = 0 \end{cases}$$

↑ known from ∇f

4 eqns in 4 unknowns
unique solution

i.e. all pairwise combinations.

other points

$(n-1) + (n-1) \cdots + (n-1)$

n points

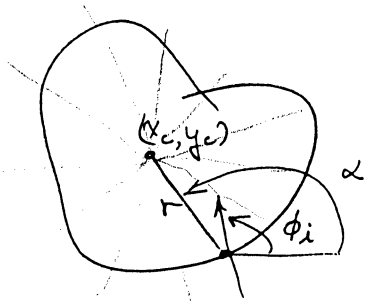
$\therefore n(n-1) \approx n^2$

computation $\propto$ (# of edge elements)^2

Generalizing the Hough transform.

basic premise of Hough

edge point data in image space ⟶ loci of possible points in parameter space.



R-table.

gradient angle	location of reference points (centers, etc.) from boundary points
ϕ_1 direction of ~~gradient~~.	$\underline{r}'_1, \underline{r}'_2, \underline{r}'_3, \ldots \underline{r}'_{n_1}$ $\quad \underline{r} = (r, \alpha)$
ϕ_2	
⋮	
ϕ_m	

different position vectors, functions of r & α from center

Algorithm 4.3 Generalized Hough

0. Construct an R-table for the shape to be located
1. Initialize Accumulator array $A(x_{cmin} : x_{cmax}, y_{cmin} : y_{cmax})$ for all possible reference points
2. For each edge point.
   - 2.1 Compute $\phi(\underline{x})$ direction of gradient
   - 2.2a. calculate all possible centers, i.e. (look up in table and compute)
     $$x_c := x + r(\phi)\cos(\alpha(\phi))$$
     $$y_c := y + r(\phi)\sin(\alpha(\phi))$$
   - 2.2b. increment the accumulator array
     $$A(x_c, y_c) := A(x_c, y_c) + 1$$
3. Possible locations for shape are local maxima of A.

Scary thought:

can handle scale S and rotation θ by increasing dimensionality of accumulator arrays.

6.8 Curve Approximation

previously curves had to pass through a subset of the edge points

approximation – not forced to pass through particular edges.

methods.

(a) least-squares regression (if sure all given points actually belong to edge)

(b) robust regression (some grouping errors present)

(c) cluster analysis (grouping of edges into contours is very unreliable or edge linking/following do not work well, i.e. edges are very scattered
example is Hough transform.

in principle, use p observations to formulate p equations for p unknown curve parameters. Doesn't work due to noise.

6.8.1 Total Regression

minimizes the sum of the squares of the perpendicular distances of the data points from the regression model.
(Used earlier to find orientation of a blob).
to allow for vertical lines, use polar coordinates.

$$x \sin\theta - y\cos\theta + \rho = 0.$$

squared perpendicular distances of points (x_i, y_i) from line

$$\lambda^2 = \sum_i (x_i \sin\theta - y_i \cos\theta + \rho)^2$$

minimum is given by

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

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

orientation of total regression line is θ given by

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

$$a = 2\sum_{i=1}^{n} x_i' y_i', \quad b = \sum_{i=1}^{n} x_i'^2 - \sum_{i=1}^{n} y_i'^2, \quad c = \sqrt{a^2+b^2}$$

$$x_i' = x_i - \bar{x} \text{ and } y_i' = y_i - \bar{y}$$

total regression
- optimal if errors are normally distributed
- poor if outliers, such as points from other contours due to poor edge linking procedure or simply an incorrect identification of corners.

6.8.2. Estimating corners

best: fit a line to edge points and then compute intersection of lines
corner location using corner operators only uses local info.

lines. $a_1 x + b_1 y + c_1 = 0$

$a_2 x + b_2 y + c_2 = 0$

location of intersection is

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

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

if $a_1 b_2 - a_2 b_1$ near zero, then lines are nearly parallel.
usually $a_1 b_2 - a_2 b_2 >$ some threshold for algorithm to work.
good procedure: fit pairs of lines over successive sublists of $2n+m$ edge points.

n = # of edge points for accurate fit

m = # of edge points to skip between sides of the corner (skips over rounded edges at corner)

6.8.3. Robust regression

try various subsets of the data points and chooses subset that provides best fit.

use resampling algorithms to determine subset that gives best estimate

1. random sample consensus
2. least-median-squares regression
3. various computer intensive methods.

linear multivariate model of order n

$$\hat{y}_i = \hat{\theta}_1 x_{i1} + \hat{\theta}_2 x_{i2} + \cdots + \hat{\theta}_n x_{in} \quad \text{for } i\text{-th data point}$$

$\hat{\theta}_i$ are the estimates of the model parameters θ_i

residuals are $r_i = y_i - \hat{y}_i$

in ordinary least squares regression used

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

$\hat{\theta}_i$ can become arbitrary even if only one data point is an outlier

influence function approach

come up with a new estimator that resembles least-squares norm but is insensitive to large errors (i.e. ignores outliers).

breakdown point – smallest % of data points that can be incorrect and not cause estimation algorithm to produce arbitrarily wrong estimates.

Z' = version of Z with m points replaced by arbitrary values

regression estimator $\hat{\theta} = T(Z)$

bias in an estimate due to outliers is

$$\text{Bias} = \sup_{Z'} \| T(Z') - T(Z) \|$$

basically a measure of robustness, below breakdown point the outliers have some small effect on the estimate. Above breakdown point, bias might become unbounded.

breakdown point

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

for least squares regression $\epsilon_n^* = \frac{1}{n}$

and as $n \to \infty$ $\epsilon_n^* \to 0$, i.e. no immunity to outliers

least-median-squares regression.

can have up to 50% bad data, i.e., $\epsilon_n^* = .5$

minimize median of squares of residuals i.e.

$$\min_{\hat{\theta}} \operatorname{med}_i r_i^2$$

Algorithm 6.3 Least-Median-Squares Regression

Assume 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 data points
3. Compute median of the square of the residuals.

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

6.4 Polyline representation – sequence of line segments.

polyline algorithm inputs ordered list of edge points

$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ can be computed to subpixel resolution

Not all points will be used, polyline algorithm interpolates a selected subset.

two point formula $\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$

gives implicit form.

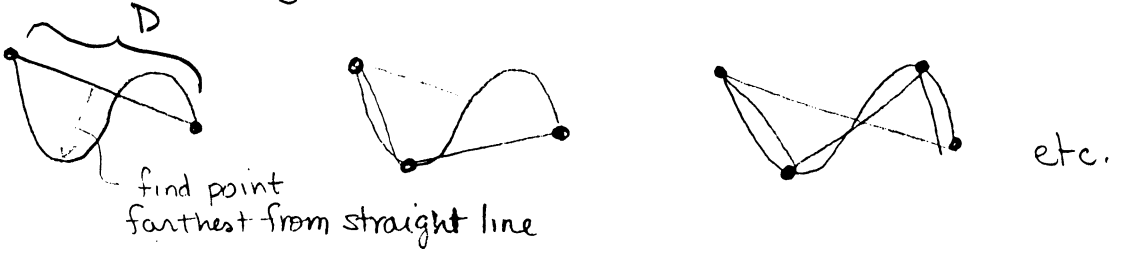
$$x(y_1 - y_2) + y(x_2 - x_1) + y_2 x_1 - y_1 x_2 = 0.$$

distance of any point from the line is $d = \frac{r}{D}$ (evaluated value of function / (distance between end points)).

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

6.4.1. Polyline splitting

recursively adds points



very efficient.

- find point in edge list farthest from straight line
- if normalized max. error > threshold add vertex at farthest edge point

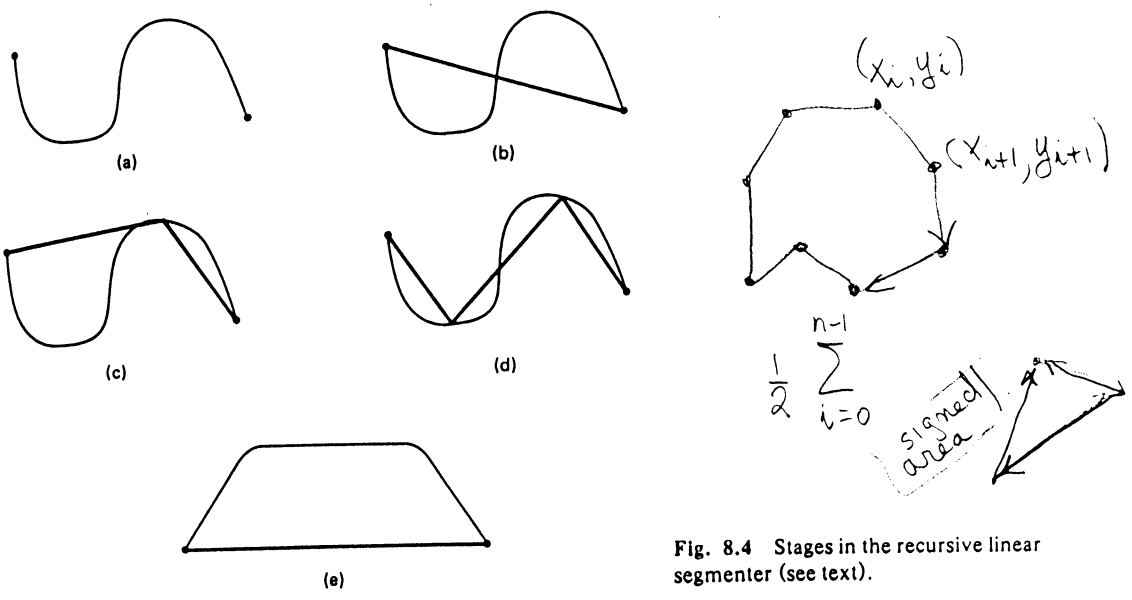


Fig. 8.4 Stages in the recursive linear segmenter (see text).

The area of a polygon may easily be computed from its polyline representation [Roberts 1965]. For a closed polyline of n points $(x(i), y(i))$, $i=0, \ldots, n-1$, labeled clockwise around a polygonal boundary, the area of the polygon is

$$\frac{1}{2}\sum_{i=0}^{n-1}(x_{i+1}y_i - x_i y_{i+1}) \tag{8.1}$$

where subscript calculations are modulo n. This formula can be proved by considering it as the sum of (signed) areas of triangles, each with a vertex at the origin, or of parallelograms constructed by dropping perpendiculars from the polyline points to an axis. This method specializes to chain codes, which are a limiting case of polylines.

8.2.2 Chain Codes

Chain codes [Freeman 1974] consist of line segments that must lie on a fixed grid with a fixed set of possible orientations. This structure may be efficiently represented because of the constraints on its construction. Only a starting point is represented by its location; the other points on a curve are represented by successive displacements from grid point to grid point along the curve. Since the grid is uniform, direction is sufficient to characterize displacement. The grid is usually considered to be four- or eight- connected; directions are assigned as in Fig. 8.5, and each direction can be represented in 2 or 3 bits (it takes 18 bits to represent the starting point in a 512 × 512 image).

Chain codes may be made position-independent by ignoring the "start point." If they represent closed boundaries they may be "start point normalized" by choosing the start point so that the resulting sequence of direction codes forms

an integer of minimum magnitude. These normalizations may help in matching. Periodic correlation (Section 3.2.1) can provide a measure of chain code similarity. The chain codes without their start point information are considered to be periodic functions of "arc length." (Here the arc length is just the number of steps in the chain code.) The correlation operation finds the (arc length) displacement of the functions at which they match up best as well as quantifying the goodness of the match. It can be sensitive to slight differences in the code.

The "derivative" of the chain code is useful because it is invariant under boundary rotation. The derivative (really a first difference mod 4 or 8) is simply another sequence of numbers indicating the relative direction of chain code segments; the number of left hand turns of $\pi/2$ or $\pi/4$ needed to achieve the direction of the next chain segment.

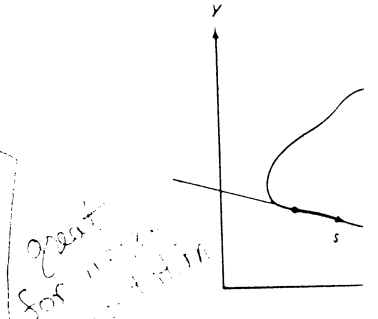
Chain codes are also well-suited for merging of regions [Brice and Fennema 1970] using the data structure described in Section 5.4.1. However, the pleasant properties for merging do not extend to union and intersection. Chain codes lend themselves to efficient calculation of certain parameters of the curves, such as area. Algorithm 8.2 computes the area enclosed by a four-neighbor chain code.

Algorithm 8.2: Chain Code Area

Comment: For a four-neighbor code (0: $+x$, 1: $+y$, 2: $-x$, 3: $-y$) surrounding a region in a counterclockwise sense, with starting point (x, y):

```
begin Chain Area;
1. area := 0;
2. yposition := y;
3. For each element of chain code
   case element-direction of
   begin case
     [0] area := area-yposition;
     [1] yposition := yposition + 1;
     [2] area := area + yposition;
     [3] yposition := yposition - 1;
   end case;
end Chain Area;
```

To merge two region boundaries is to remove any boundary they share, obtaining a boundary for the region resulting from gluing the two abutting regions together. As we saw in Chapter 5, the chain codes for neighboring regions are closely related at their common boundary, being equal and opposite in a clearly defined sense (for N-neighbor chain codes, one number is equal to the other plus $N/2$ modulo N (see Chapter 5). This property allows such sections to be identified readily, and easily scissored out to give a new merged boundary. As with polylines, it is not immediately obvious from a chain-coded boundary and a point whether the point is within the boundary or outside. Many algorithms for use with chain code representations may be found in [Freeman 1974; Gallus and Neurath 1970].



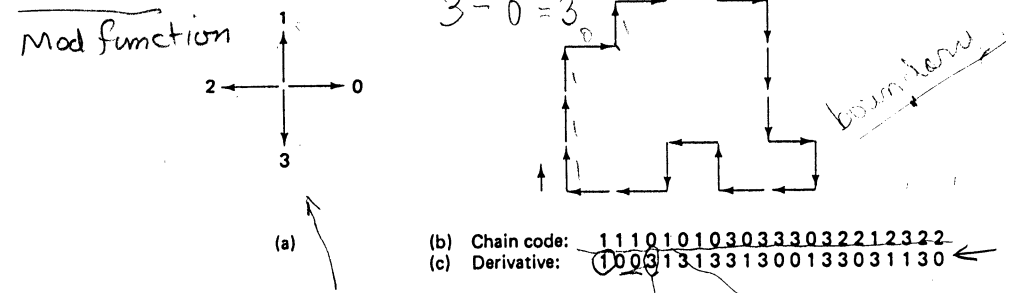


Fig. 8.5 (a) Direction numbers for chain code elements. (b) Chain code for the boundary shown. (c) Derivative of (b).

8.2.3 The ψ -s Curve

The $\psi-s$ curve is like a continuous version of the chain code representation; it is the basis for several measures of shape. ψ is the angle made between a fixed line and a tangent to the boundary of a shape. It is plotted against s, the arc length of the boundary traversed. For a closed boundary, the function is periodic, with a discontinuous jump from 2π back to 0 as the tangent reattains the angle of the fixed line after traversing the boundary.

Horizontal straight lines in the $\psi-s$ curve correspond to straight lines on the boundary (ψ is not changing). Nonhorizontal straight lines correspond to segments of circles, since ψ is changing at a constant rate. Thus the $\psi-s$ curve itself may be segmented into straight lines [Ambler et al. 1975], yielding a segmentation of the boundary of the shape in terms of straight lines and circular arcs (Fig. 8.6).

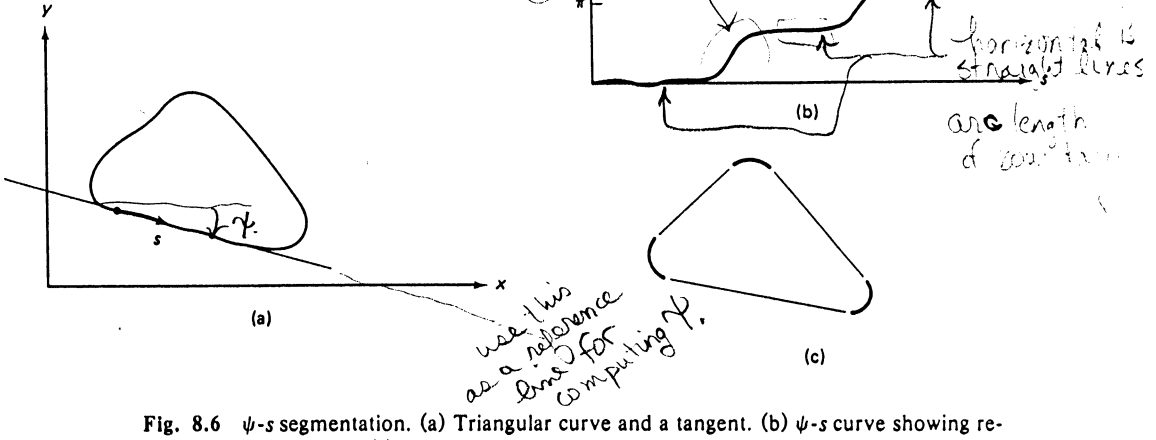


Fig. 8.6 ψ-s segmentation. (a) Triangular curve and a tangent. (b) ψ-s curve showing regions of high curvature. (c) Resultant segmentation.