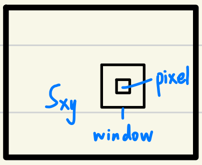
last time: edge detection

First: edge linking

start with edge pixels and corrsponding

$M(x,y)$, $\alpha(x,y)$

idea: for each edge pixel (x,y), make a window Sxy around that pixel



for each $(s,t) \in$ Sxy, "link" (x,y) to (s,t) if

$|M(x,y) - M(s,t)| \le T_1$

$|\alpha(x,y) - \alpha(s,t)| \le T_2$

trace out long edges

2) boundary following

we have edge points around a closed contour want to link/order them in a clockwise direction.

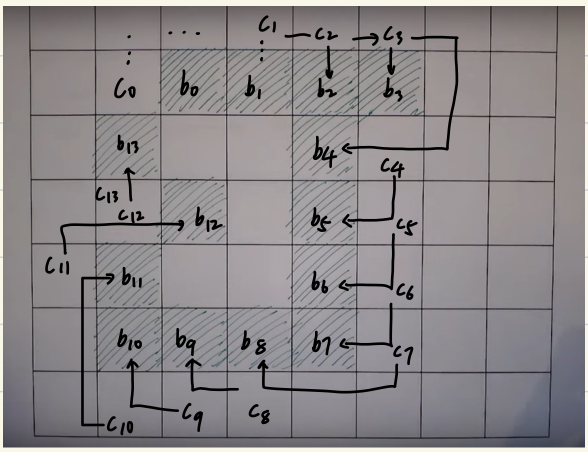
Moore's boundary following algorithm

algo: 0) edge map: 1=edge pixel, 0=Not

1) let starting point b_0 be the uppermost, left-most point labeled "1"

let c_0 be the left neighbour of b_0

2) examine 8-neighbour of b_0 , starting at c_0 and going clockwise. Let b_1 be first point I find "1" and c_1 be the proceding 0

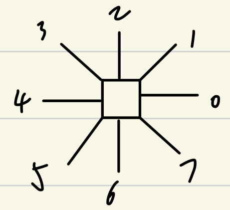


let b= b_1 , c= c_1 . repeat process of updating b and c

4) continue until $b = b_0$ and next boundary point found is b_1

5) the ordered list of b's is the boundary

once we have such a boundary, we can describe it with a chain code



for the graph before, our list is {0,0,0,5,6,6,6,4,4,4,2,1,3,1} the order depends on the starting point.

to be able to match shapes at different orientations, we can

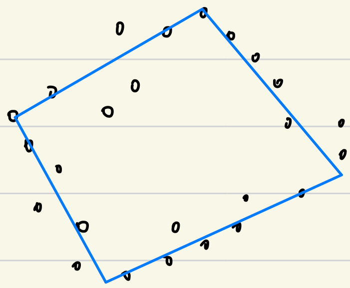
1) order the chain code so it always starts with the minimum magnitude integer.

2) just encode the difference between the direction

{0,0,5,1,0,0,6,0,0,6,7,2,6,7}

$(b_{i+1} - b_i) \bmod 8$

polygonal fitting to a set of ordered points



Let P be a sequence of ordered distinct point (eg, ordered edges after boundary following)

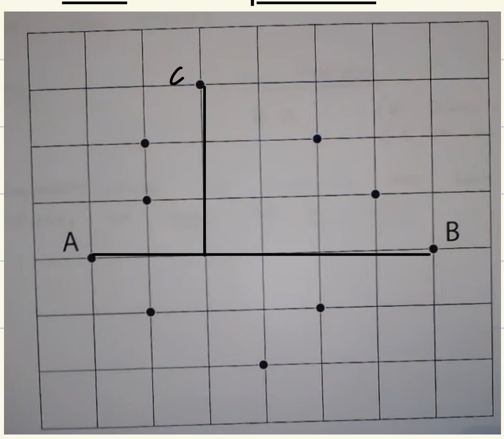
specify two starting points A, B

- if the curve is open, A, B are the natural endpoints.
- if the curve is closed, A, B are the left and right most points
- specify a threshold T (pixel distance)

Create two stacks:

<u>Final</u>	<u>In process</u>
B	B A (if closed curve)
	A (if opened curve)

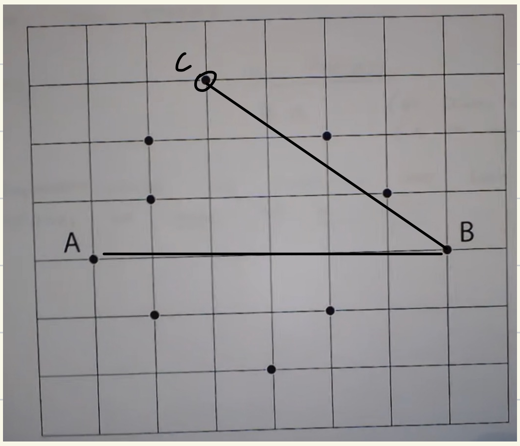
1) computer the line connecting the last vertices of <u>final</u> and in <u>process</u>



2) compute distance from this line to all points between these vertices (counter-clockwise dir) select vertex Vmax with max distance Dmax

3) if Dmax > T, put Vmax at the end of In process and go to step1

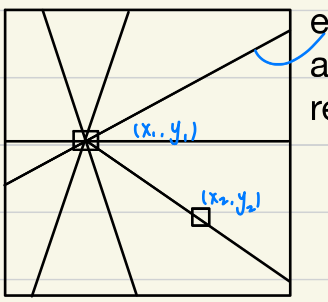
Final	In process
B	BAC



4) otherwise, remove last vertex from in process and make it the last vertex of final
5) if in process is not empty, go to step1
6) otherwise, done - the vertices in final are the ordered vertices of a polygon

Fitting straight lines in images.
easy to do with previous algorithm when there is only one line. What about multiple line and clutter (many edge pixels not on any line)?

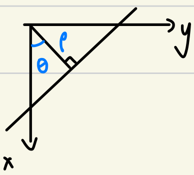
The hough transform:



each possible line through an edge pixel can be represented as an equation

$y = mx + b$

$x \cos\theta + y \sin\theta = \rho$



$\theta = 0$ horizontal
$\theta = 90$ vertical

A single edge point (x,y) could belong to many possible lines in the (ρ, θ) plane

basic idea:
1) detect edge points -> binary image
2) subdivide of (ρ, θ) plane
3) for each edge point, increment corresponding (ρ, θ) cell by +1
4) look for (ρ, θ) cells with large pixel counts (peaks)
5) select highest peaks
6) map the corresponding (ρ, θ)'s into lines in the (x,y) plane (or get line segment that corresponding to votes)