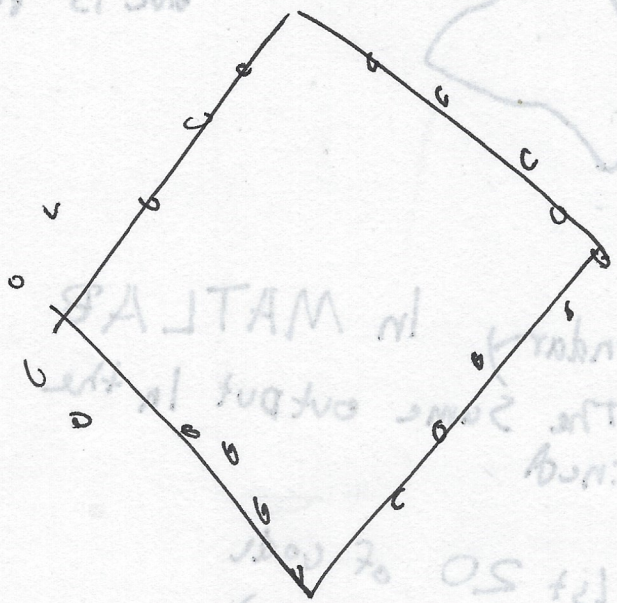
(8) We could write a polygon fitting algorithm that would specify a tolerance of a segment fitting the edge points.

Polygonal Fitting To a set of ordered points

→ we already followed our edges around to create an ordered list (the boundary following algorithm)



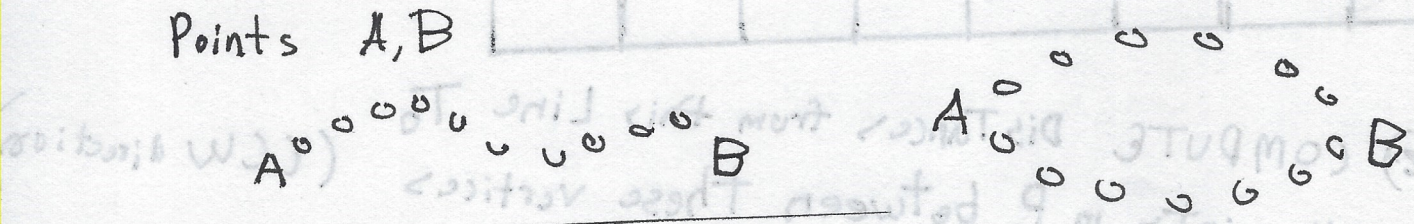
→ set of edge points not on straight lines

→ turn into polygon and we have a success as long as none of the points are too far away

Do not connect the dots, too fine detailed and noisy

- LET P be a sequence of ordered, Distinct Point, (e.g. 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 Rightmost Points.
- Specify a threshold T (Pixel Distance)
  T controls how close every point is going to have to be to the boundary lines

Points A, B



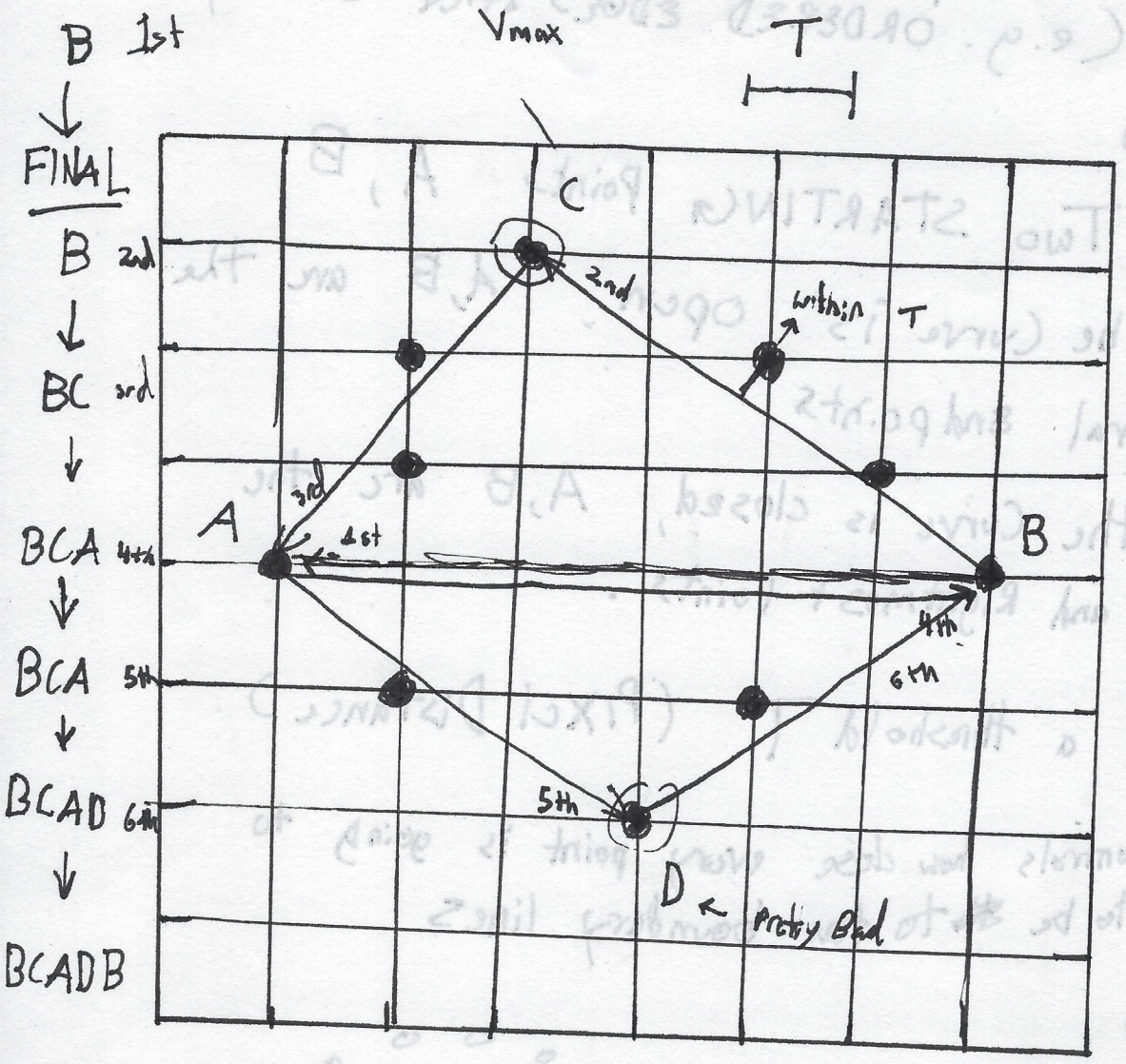
CREATE TWO STACKS:

Final
B

(Denote an end point)

In Process
B A (If closed curve)
(A if OPEN curve)

1) COMPUTE Line connecting the Last VERTICES of Final and In Process



BA 1st
IN Process
BAC 2nd
BA 3rd
B 4th
BD 5th
B 6th

2) COMPUTE Distances from this Line To All points in P between these vertices (CCW direction
(point Vmax have max Distance Dmax)
maximum vertex

3) IF $D_{max} > T$, put Vmax at the end of IN Process and go to step 1)

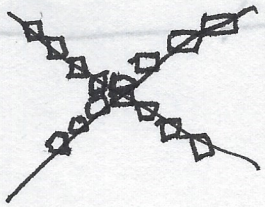
4) Otherwise Renove Last Vertex from ~~In Process~~ In Process and Make it the Last Vertex of Final.

5) If In Process is not Empty, Go to step 1

6) If In Process is Empty, we are Done - The Vertices in Final are the OrDered Vertices of a Polygon.

Note: These techiques won't work when we have multiple shapes whose "support" is spread out over the Image. Also, may be clutter/Noise.

Fitting Straight Lines in Images. 45:10



Easy to do with the previous algorithm with a only one line

What if there are multiple lines?
And Clutter? (Many edge pixels not on any Line?)

- Separate the liney stuff and non-liney stuff

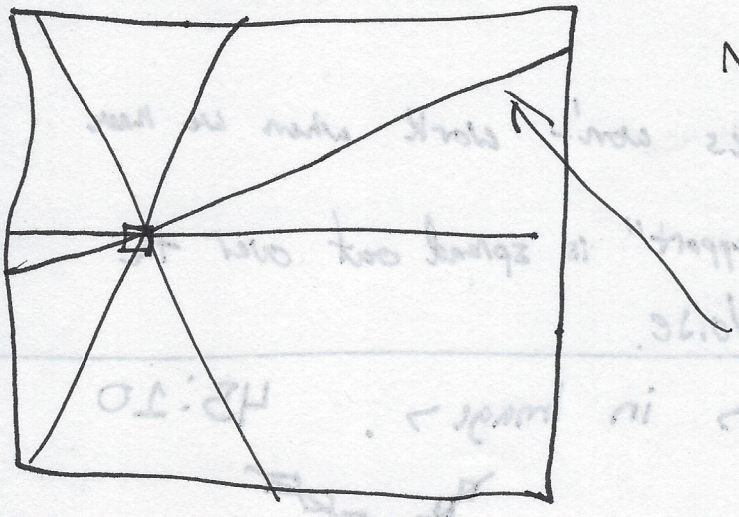
matLab:

```
im=imread('selves.jpg');
im=rgb2gray(im);
imshow(im)
imshow(edge(im)) — Lot of pixels along edges
```

So which edges lie on lines and which don't?

The Hough Transform

consider 1 edge pixel in this image



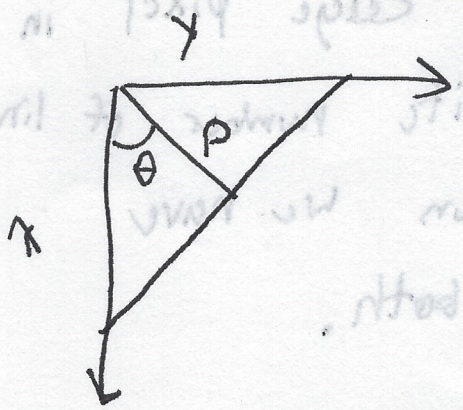
Now, think about all lines going through the pixel (of every degree)

Each possible Line through an edge pixel can be Represented as an Equation

$y = mx + b$ — problems with a vertical line

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

Image space



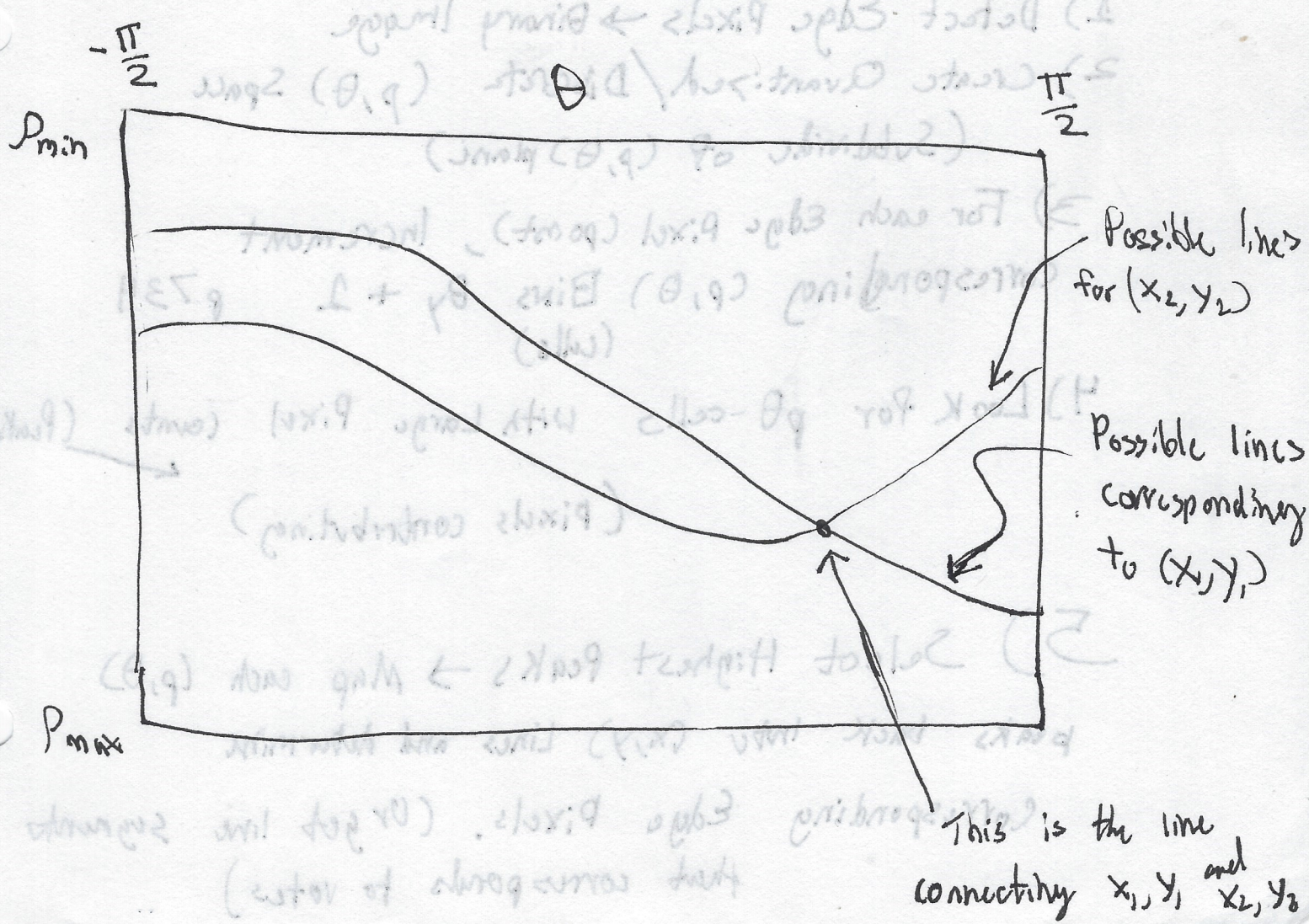
$\theta = 0$ Horizontal line $x = \rho$

$\theta = \pi/2$ vertical line $y = \rho$

Use θ and ρ to represent a line

5分:50

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



(14) consider (x_2, y_2) as another Edge pixel in the Image, we have an infinite number of line going through it, so then we have one line going through both.

Going through a lot of Edge pixels, we can see a lot of lines in $p\theta$-plane going through a point.

So that point in $p\theta$-plane is likely our best bet.

Process

1) Detect Edge Pixels → Binary Image
2) Create Quantized/Discrete (p, θ) Space (Subdivide of (p, θ) plane)
3) For each Edge Pixel (point), Increment corresponding (p, θ) Bins (cells) By +1 p734
4) Look for $p\theta$-cells with Large Pixel counts (Peaks) ← (Pixels contributing)
5) Select Highest Peaks → Map each (p, θ) peaks back into (x, y) Lines and determine Corresponding Edge Pixels. (Or get line segments that corresponds to votes)

MatLab 58:00

hough

houghlines

houghpeaks

myhough — 58:52

Hough transform gives only Straight Lines so lens distortion can affect the expected results of the hough transform

Do Better With LPF? 1:13:00

Before hough

Next time, thresholding