Fitting

D.A. Forsyth, CS 543

Fitting

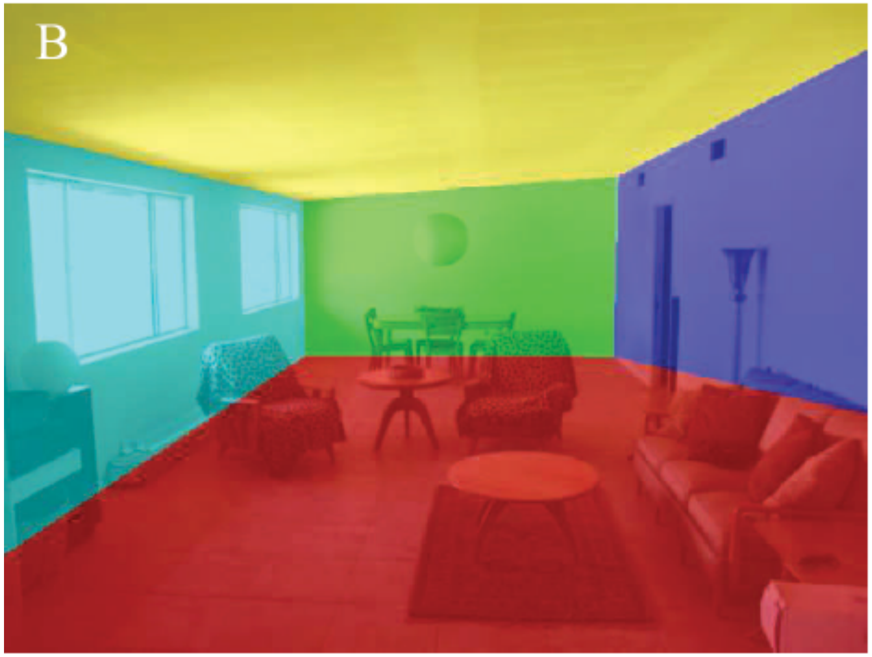
- Choose a parametric object/some objects to represent a set of tokens
- Most interesting case is when criterion is not local
  - can’t tell whether a set of points lies on a line by looking only at each point and the next.
- Three main questions:
  - what object represents this set of tokens best?
  - which of several objects gets which token?
  - how many objects are there?

  (you could read line for object here, or circle, or ellipse or...)

Scenes as stages

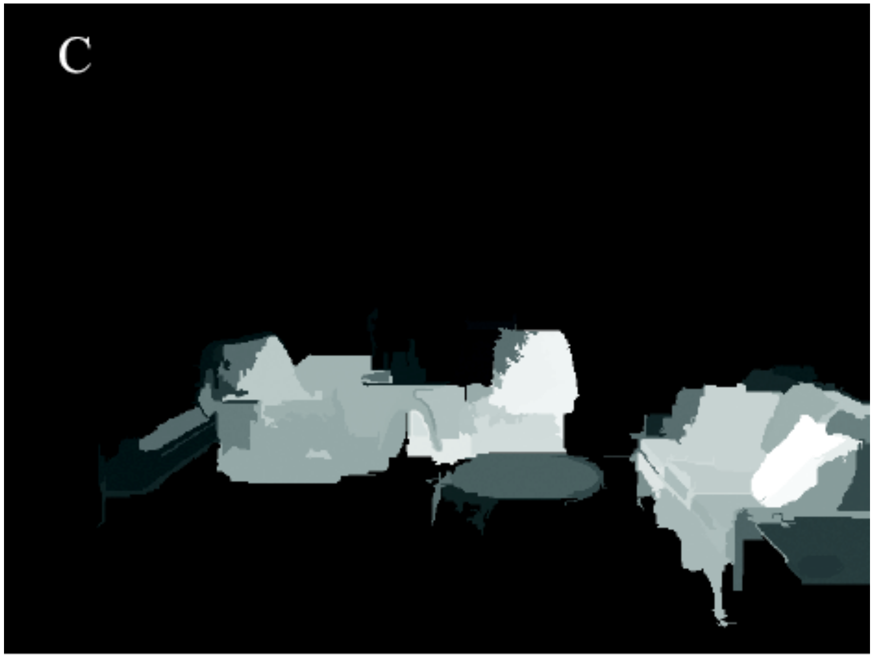
Hedau et al, in review

The "stage"



Hedau et al, in review

The “players”



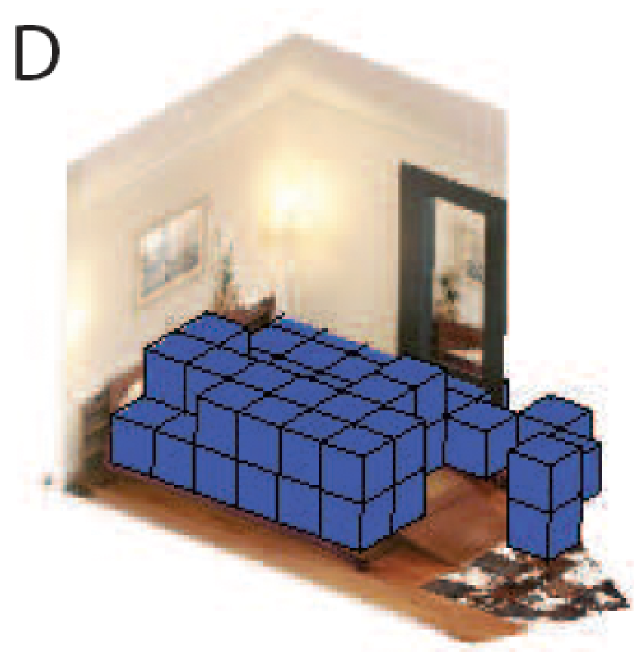
Hedau et al, in review


Hedau et al, in review

Stage and Players

Now, what can we do...

- Free space estimate
  - using standard SFM construction for camera given manhattan world
  - couple appearance model of objects to approximate geometric models?



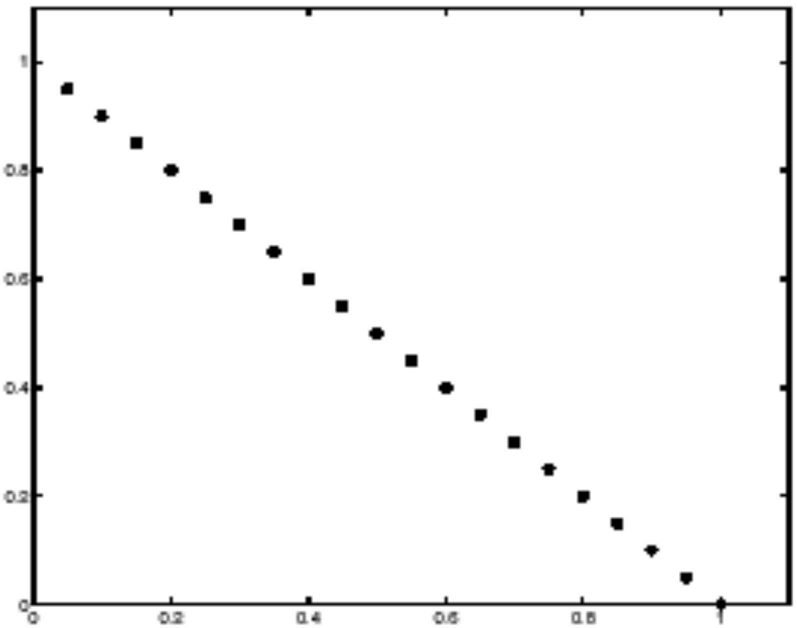
Hedau et al, in review

Fitting and the Hough Transform

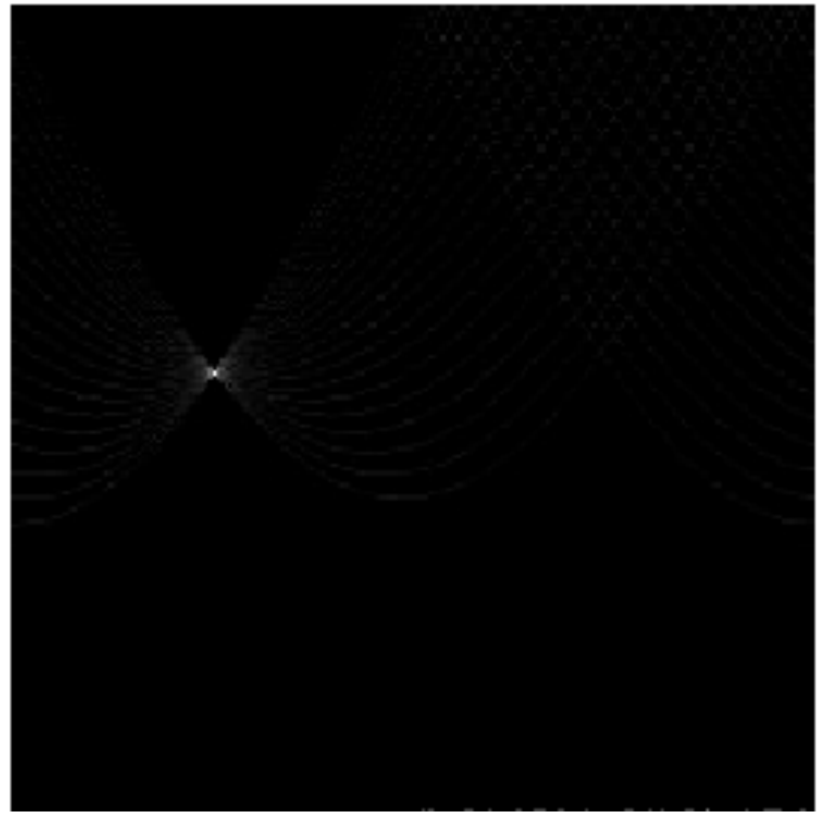
- Purports to answer all three questions
  - in practice, answer isn't usually all that much help
- We do for lines only
- A line is the set of points (x, y) such that

$$\boxed{(\sin\theta)x + (\cos\theta)y + d = 0}$$

  - Different choices of θ, $d>0$ give different lines
    - For any (x, y) there is a one parameter family of lines through this point, given by $\boxed{(\sin\theta)x + (\cos\theta)y + d = 0}$
    - Each point gets to vote for each line in the family; if there is a line that has lots of votes, that should be the line passing through the points



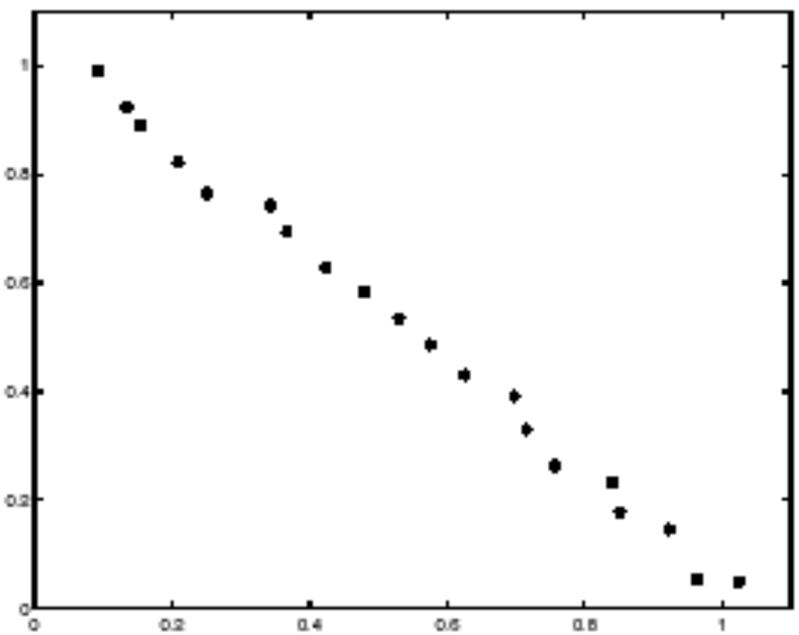
tokens



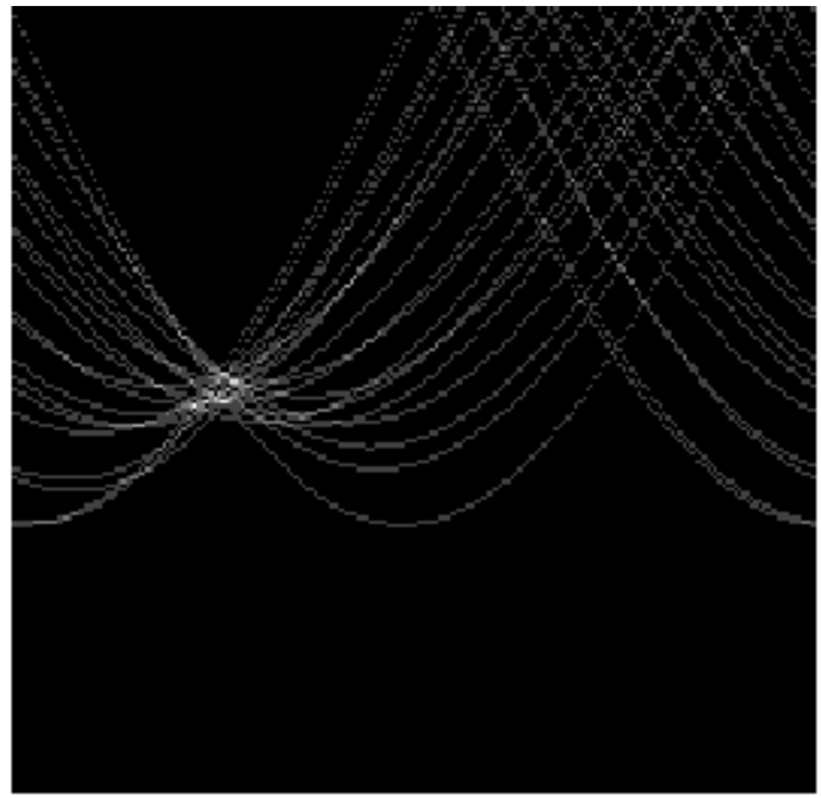
votes

Mechanics of the Hough transform

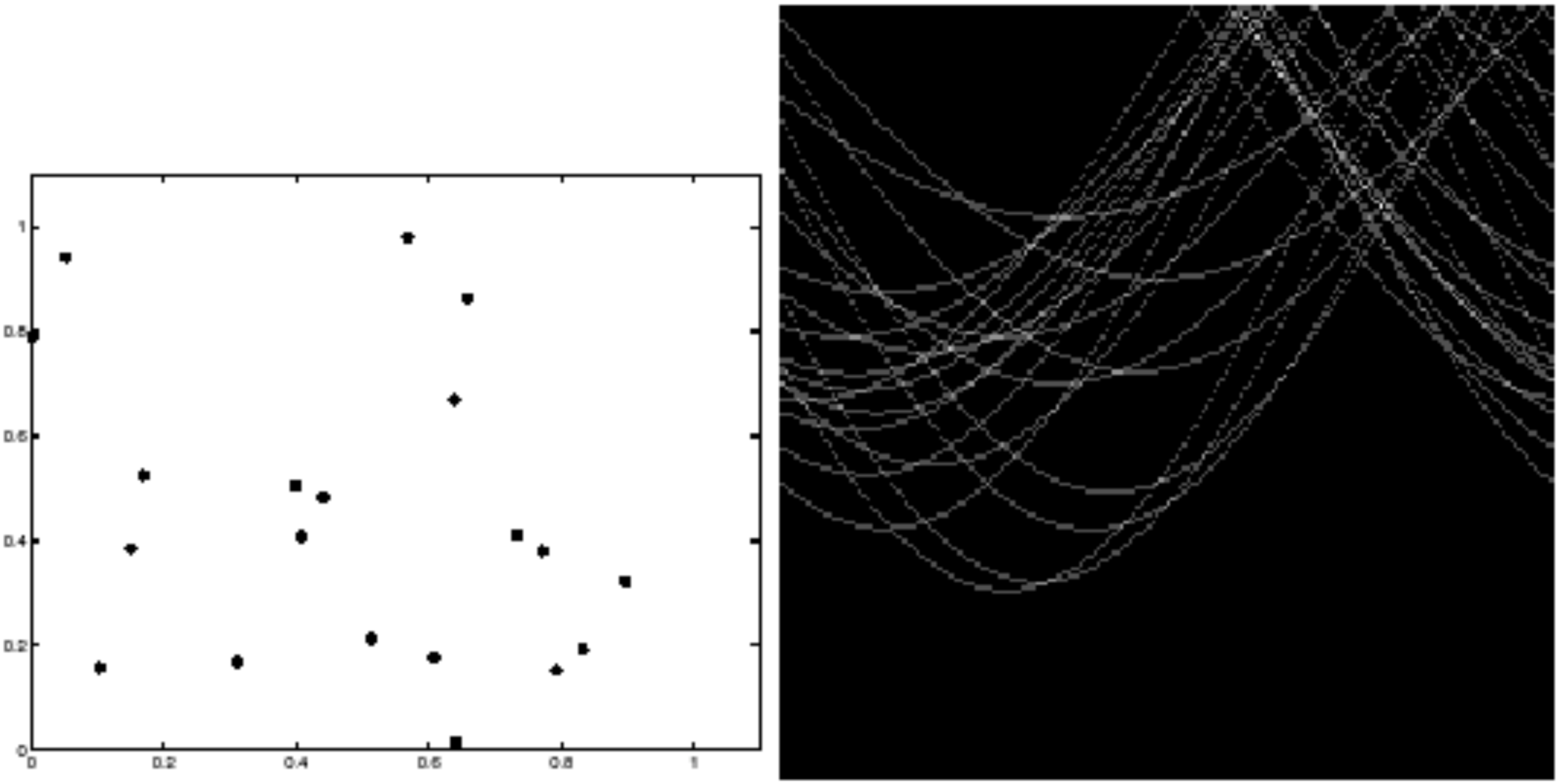
- Construct an array representing θ, d
- For each point, render the curve (θ, d) into this array, adding one at each cell
- Difficulties
  - how big should the cells be? (too big, and we cannot distinguish between quite different lines; too small, and noise causes lines to be missed)
  - How many lines?
    - count the peaks in the Hough array
  - Who belongs to which line?
    - tag the votes
- Hardly ever satisfactory in practice, because problems with noise and cell size defeat it



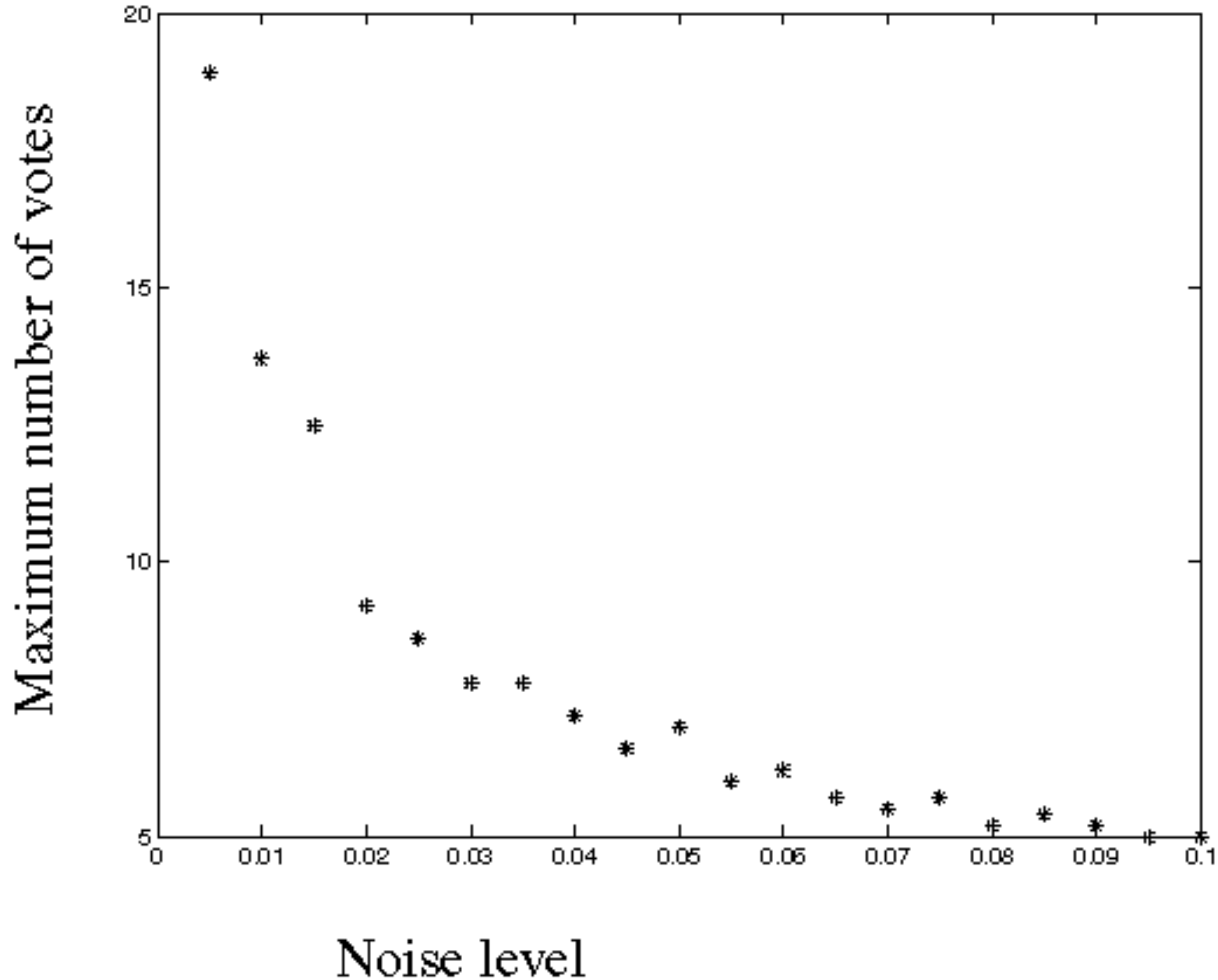
tokens



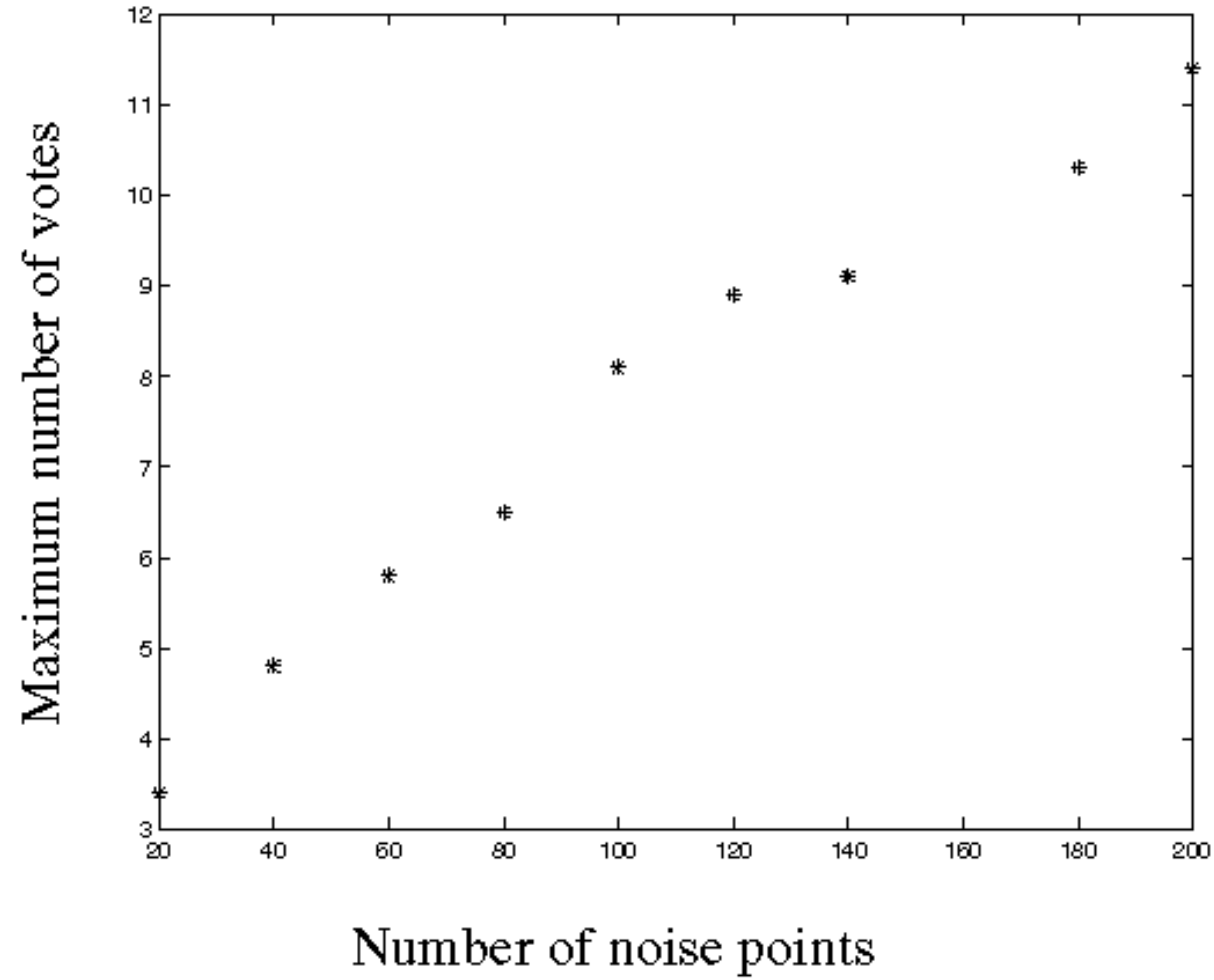
votes



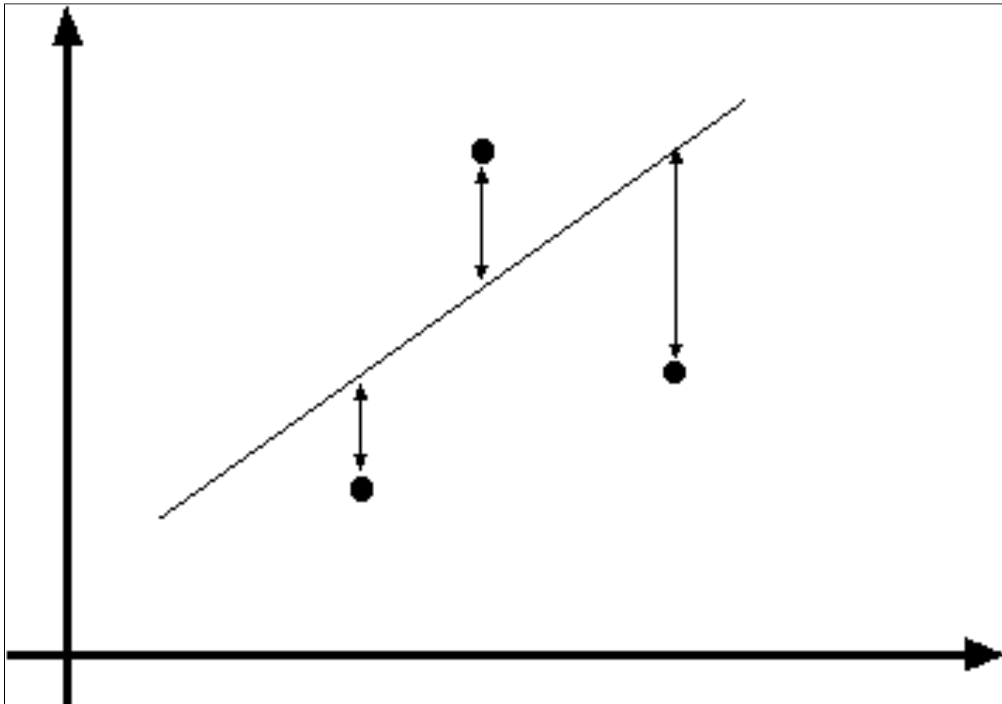
1
0.8
0.6
0.4
0.2
0
0.2
0.4
0.6
0.8
1



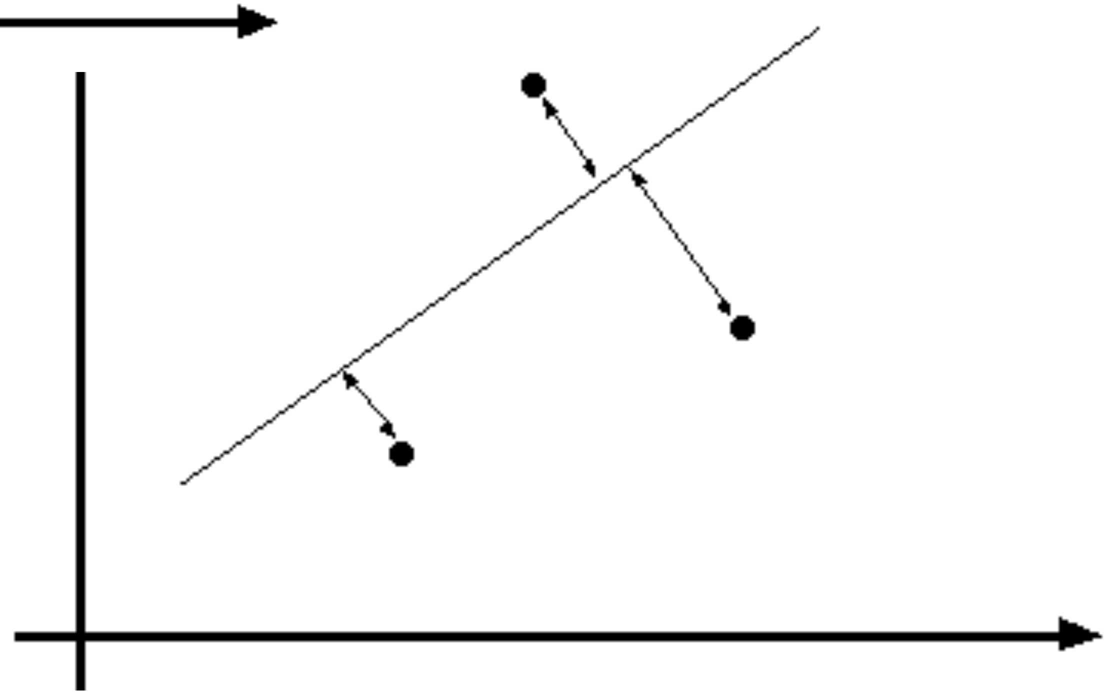
20
15
10
5
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
Maximum number of votes
Noise level



Maximum number of votes
Number of noise points
12
11
10
9
8
7
6
5
4
3
20
40
60
80
100
120
140
160
180
200



Line fitting can be max. likelihood - but choice of model is important



Who came from which line?

- Assume we know how many lines there are - but which lines are they?
  - easy, if we know who came from which line
- Three strategies
  - Incremental line fitting
  - K-means
  - Probabilistic (later!)

Algorithm 15.1: Incremental line fitting by walking along a curve, fitting a line to runs of pixels along the curve, and breaking the curve when the residual is too large

```
Put all points on curve list, in order along the curve
Empty the line point list
Empty the line list
Until there are too few points on the curve
   Transfer first few points on the curve to the line point list
   Fit line to line point list
   While fitted line is good enough
      Transfer the next point on the curve
         to the line point list and refit the line
   end
   Transfer last point(s) back to curve
   Refit line
   Attach line to line list
end
```

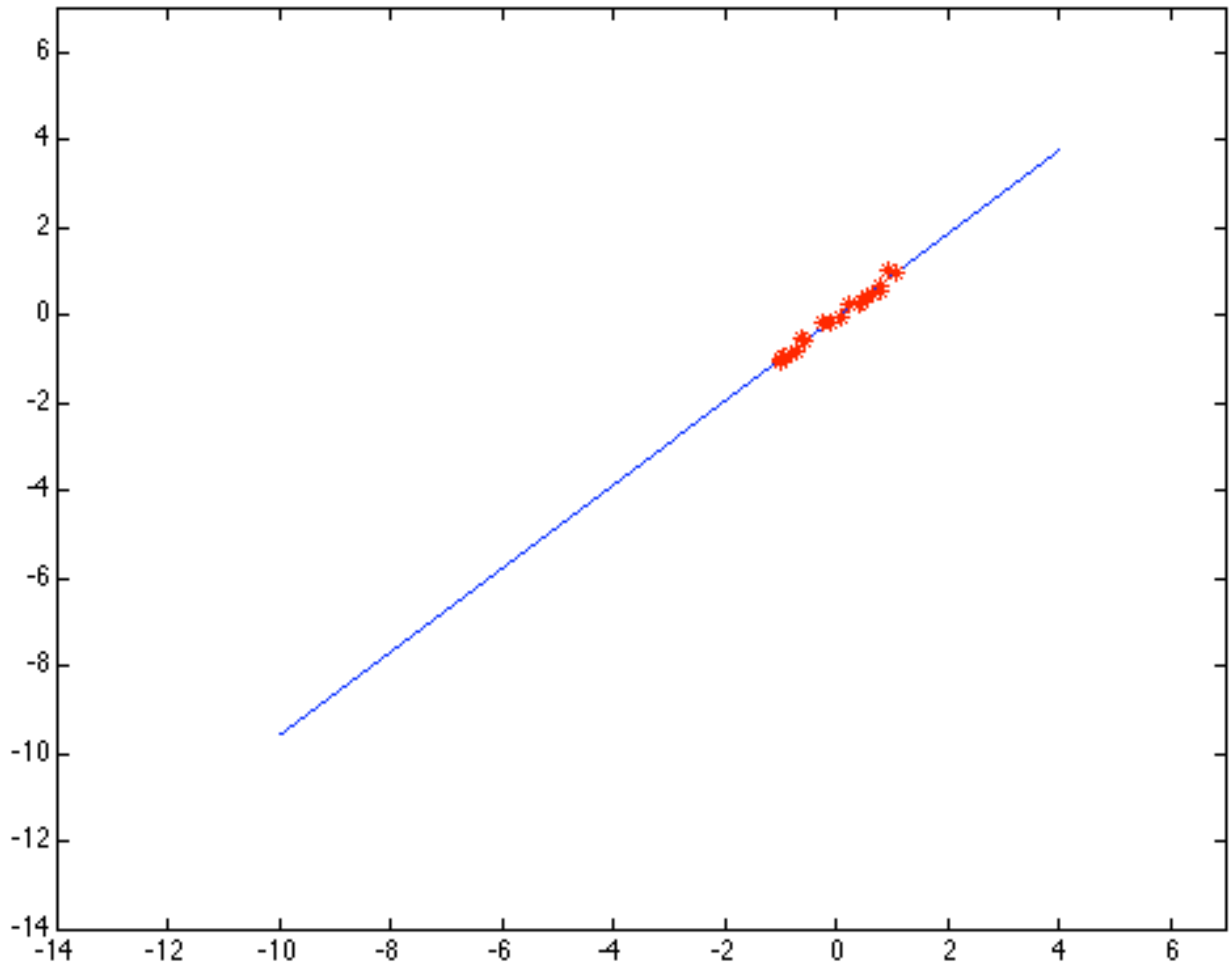
Algorithm 15.2: K-means line fitting by allocating points to the closest line and then refitting.

Hypothesize k lines (perhaps uniformly at random)
or
Hypothesize an assignment of lines to points
  and then fit lines using this assignment

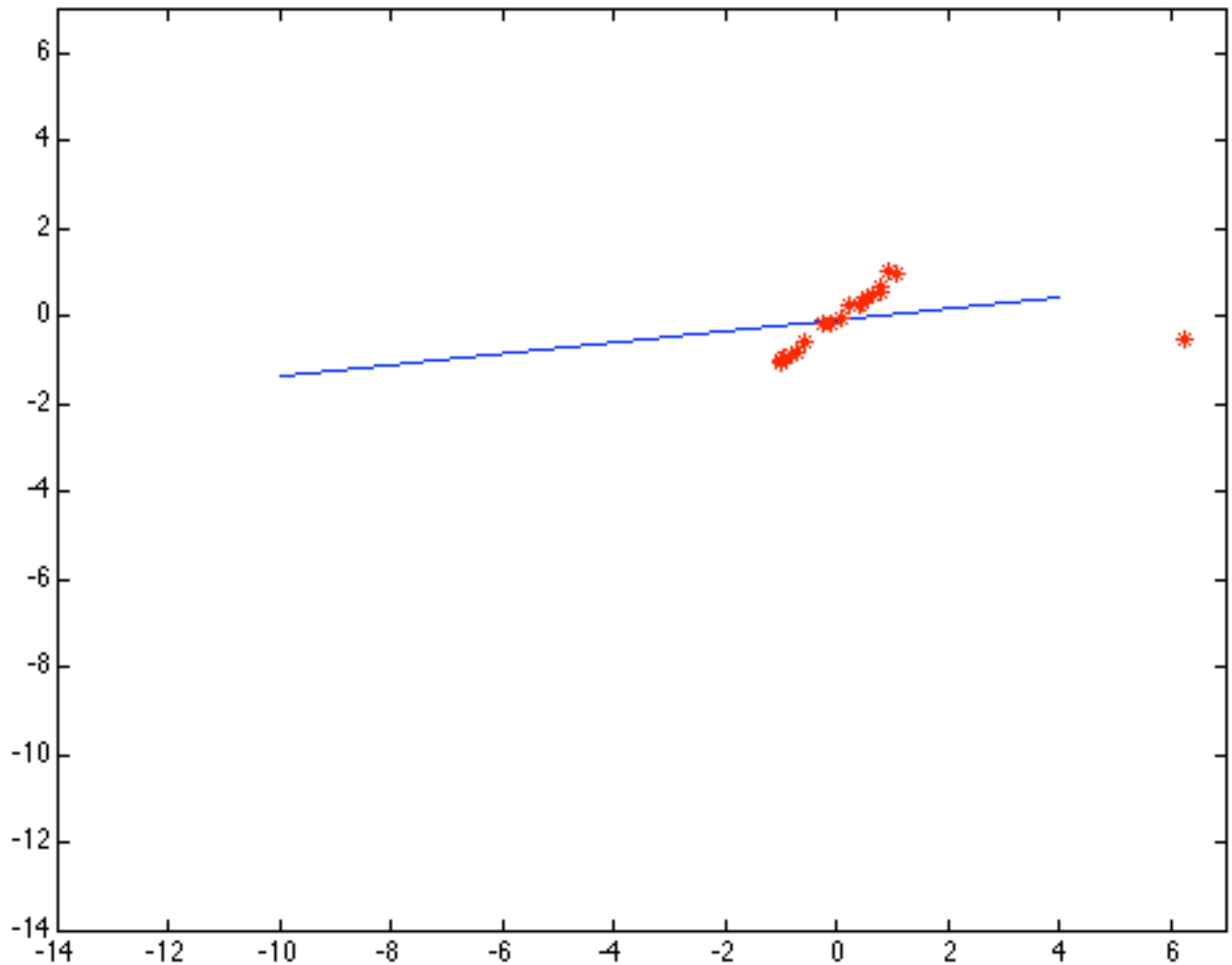
Until convergence
  Allocate each point to the closest line
  Refit lines
end

Robustness

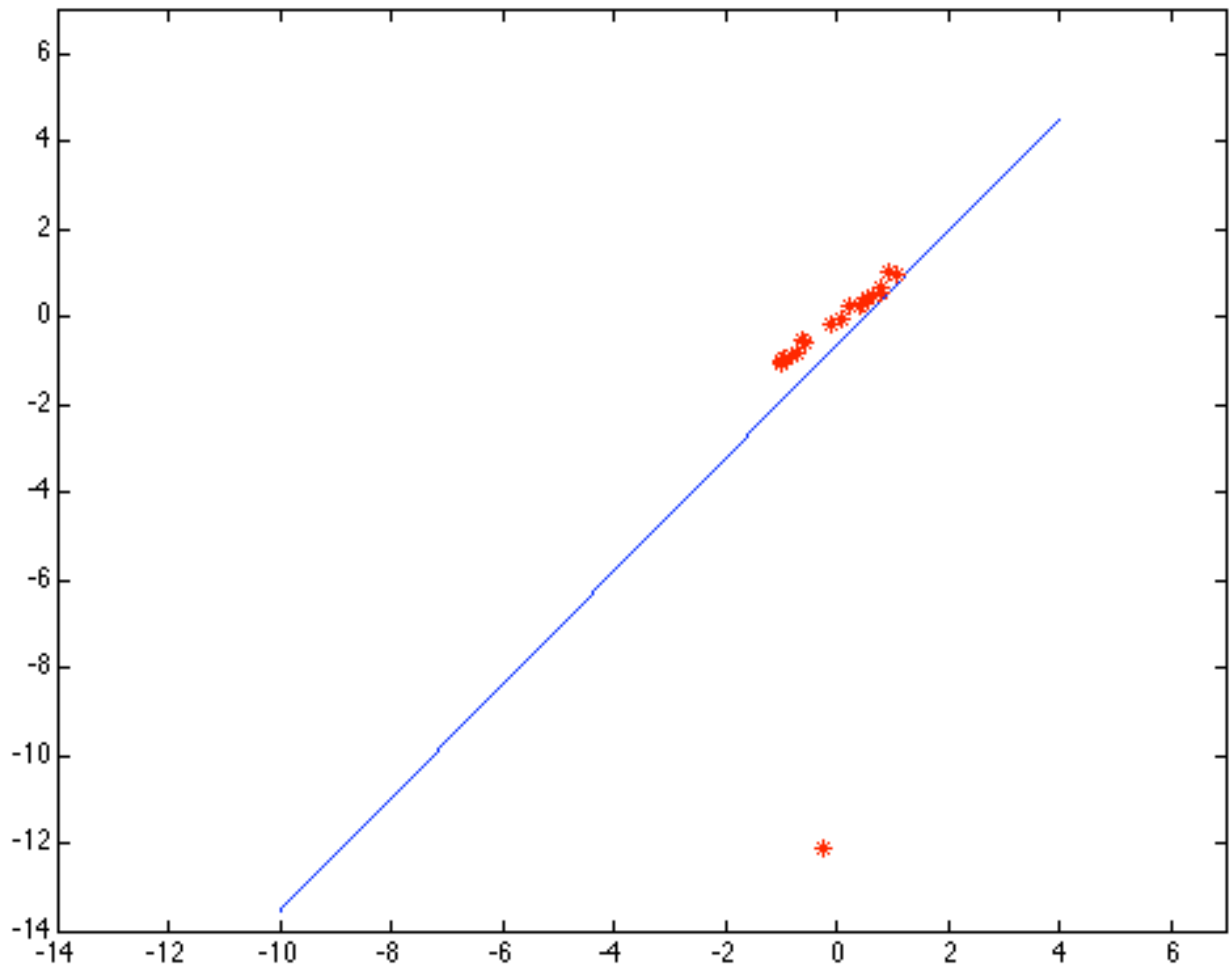
- As we have seen, squared error can be a source of bias in the presence of noise points
  - One fix is EM - we'll do this shortly
  - Another is an M-estimator
    - Square nearby, threshold far away
  - A third is RANSAC
    - Search for good points



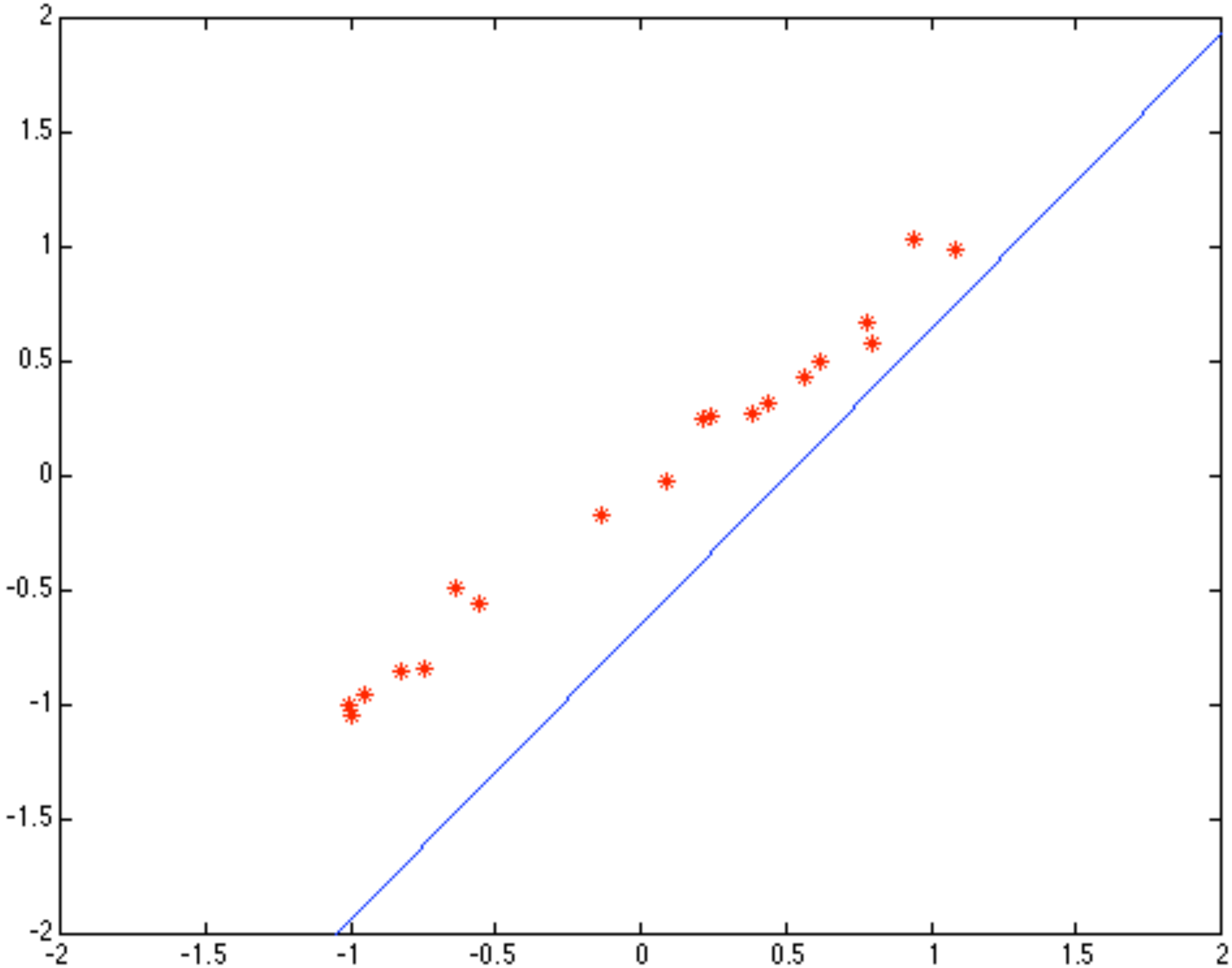
6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6



6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6



6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6



2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

RANSAC

- Algorithm
  - Choose a small subset uniformly at random
  - Fit to that
  - Anything that is close to result is signal; all others are noise
  - Refit
  - Do this many times and choose the best
- Issues
  - How many times?
    - Often enough that we are likely to have a good line
  - How big a subset?
    - Smallest possible
  - What does close mean?
    - Depends on the problem
  - What is a good line?
    - The number of nearby points is so big it is unlikely to be all outliers

Algorithm 15.4: RANSAC: fitting lines using random sample consensus

```
Determine:
    n — the smallest number of points required
    k — the number of iterations required
    t — the threshold used to identify a point that fits well
    d — the number of nearby points required
      to assert a model fits well
Until k iterations have occurred
    Draw a sample of n points from the data
      uniformly and at random
    Fit to that set of n points
    For each data point outside the sample
      Test the distance from the point to the line
        against t; if the distance from the point to the line
        is less than t, the point is close
    end
    If there are d or more points close to the line
      then there is a good fit. Refit the line using all
      these points.
end
Use the best fit from this collection, using the
  fitting error as a criterion
```

Fitting curves other than lines

- In principle, an easy generalisation
  - The probability of obtaining a point, given a curve, is given by a negative exponential of distance squared
- In practice, rather hard
  - It is generally difficult to compute the distance between a point and a curve

Missing variable problems

- In many vision problems, if some variables were known the maximum likelihood inference problem would be easy
  - fitting; if we knew which line each token came from, it would be easy to determine line parameters
  - segmentation; if we knew the segment each pixel came from, it would be easy to determine the segment parameters
  - fundamental matrix estimation; if we knew which feature corresponded to which, it would be easy to determine the fundamental matrix
  - etc.
- This sort of thing happens in statistics, too

Missing variable problems

- Strategy
  - estimate appropriate values for the missing variables
  - plug these in, now estimate parameters
  - re-estimate appropriate values for missing variables, continue
- eg
  - guess which line gets which point
  - now fit the lines
  - now reallocate points to lines, using our knowledge of the lines
  - now refit, etc.
- We’ve seen this line of thought before (k means)

Missing variables - strategy

- We have a problem with parameters, missing variables
- This suggests:
- Iterate until convergence
  - replace missing variable with expected values, given fixed values of parameters
  - fix missing variables, choose parameters to maximise likelihood given fixed values of missing variable
- e.g., iterate till convergence
  - allocate each point to a line with a weight, which is the probability of the point given the line
  - refit lines to the weighted set of points
  - Converges to local extremum
  - Somewhat more general form is available

Lines and robustness

- We have one line, and n points
- Some come from the line, some from "noise"
- This is a mixture model:

$$
\begin{aligned}
P(\text{point} \mid \text{line and noise params}) &= P(\text{point} \mid \text{line})P(\text{comes from line}) + \\
&\quad P(\text{point} \mid \text{noise})P(\text{comes from noise}) \\
&= P(\text{point} \mid \text{line})\lambda + P(\text{point} \mid \text{noise})(1-\lambda)
\end{aligned}
$$

- We wish to determine
  - line parameters
  - p(comes from line)

Complete data

- Introduce a set of hidden variables,
  - δ, one for each point.
  - They are one when the point is on the line, and zero when off.
- If these were known, we would have:

The complete data log-likelihood

- Log-likelihood if delta's were known
  - quite easy to work with

But we don’t know delta

- Solution:
  - Iterate
    - estimate delta’s with fixed parameters
    - estimate parameters with new estimates of delta
- Formally
  - obtain a starting theta
  - now at each step, maximize

Substituting for delta

- Notice:
  - points close to current line means P(delta=1|x, theta) will be big
  - far -> small
  - like k-means but now we are averaging
  - sometimes called soft assignment

Algorithm for line fitting

- Obtain some start point

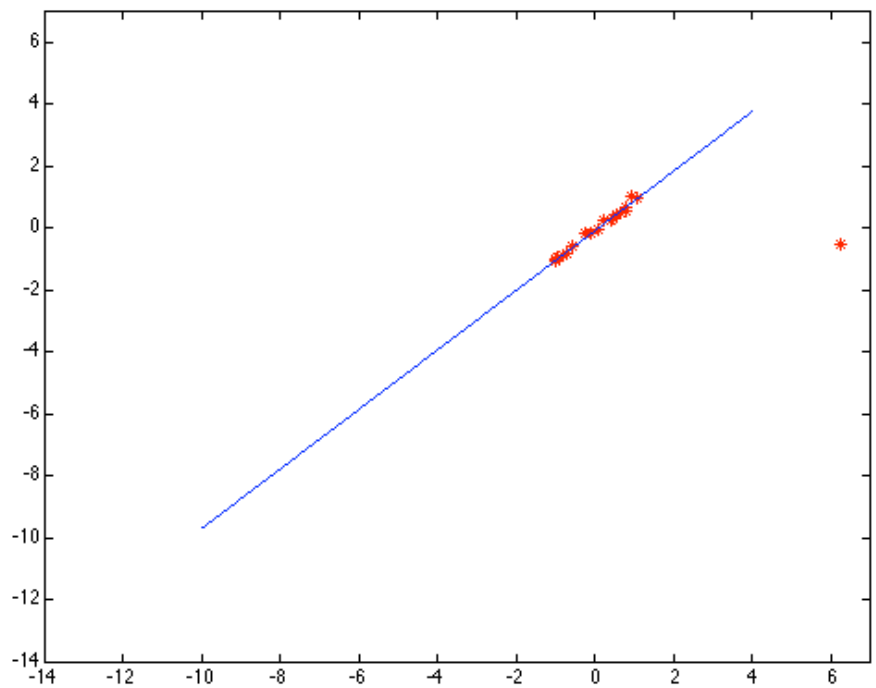
$$\boxed{\theta^{(0)} = \left(\phi^{(0)}, c^{(0)}, \lambda^{(0)}\right)}$$

- Now compute δ's using formula above

- Now compute maximum likelihood estimate of

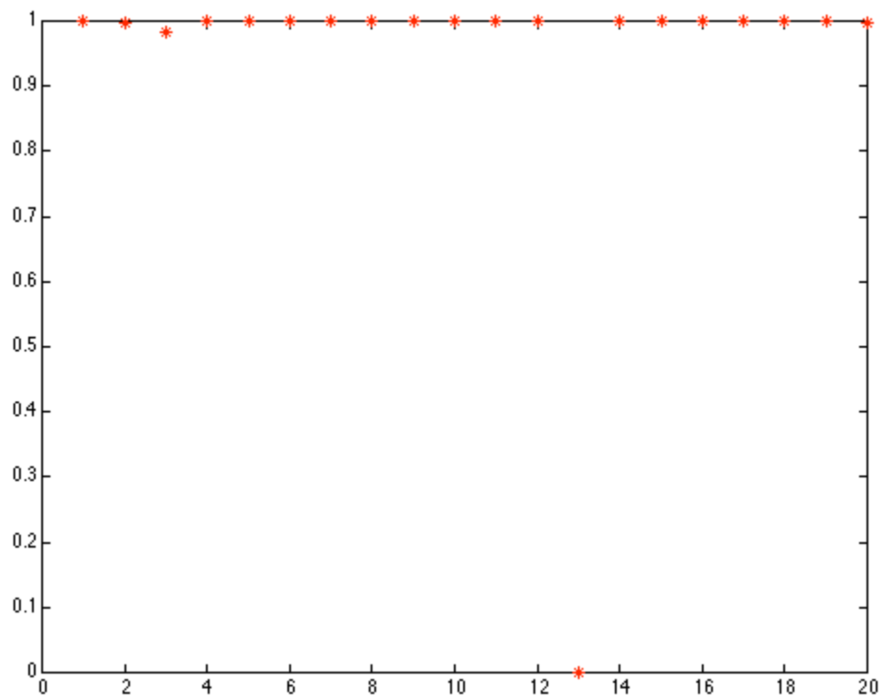
$$\boxed{\theta^{(1)}}$$

  - ϕ, c come from fitting to weighted points
  - λ comes by counting
  - • Iterate to convergence

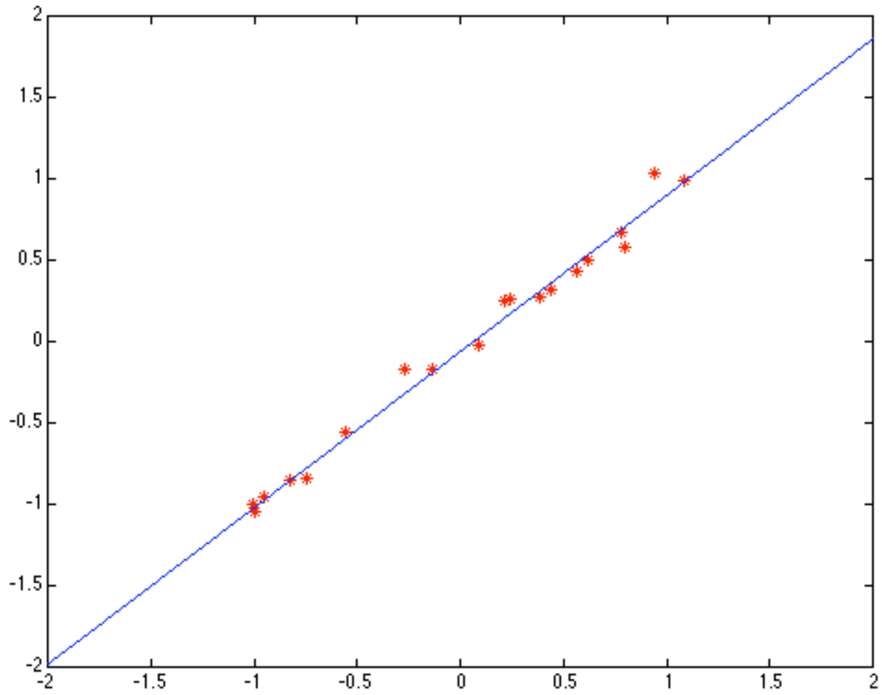


6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-14
-12
-10
-8
-6
-4
-2
0
2
4
6

The expected values of the deltas at the maximum
(notice the one value close to zero).



Closeup of the fit



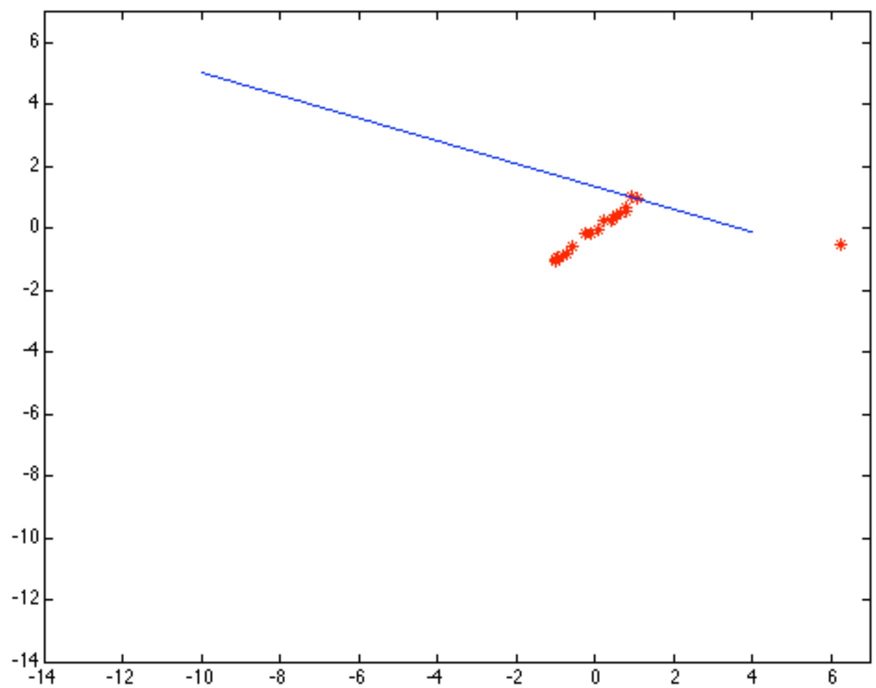
Choosing parameters

- What about the noise parameter, and the sigma for the line?
  - several methods
    - from first principles knowledge of the problem (seldom really possible)
    - play around with a few examples and choose (usually quite effective, as precise choice doesn't matter much)
  - notice that if kn is large, this says that points very seldom come from noise, however far from the line they lie
    - usually biases the fit, by pushing outliers into the line
    - rule of thumb; its better to fit to the better fitting points, within reason; if this is hard to do, then the model could be a problem

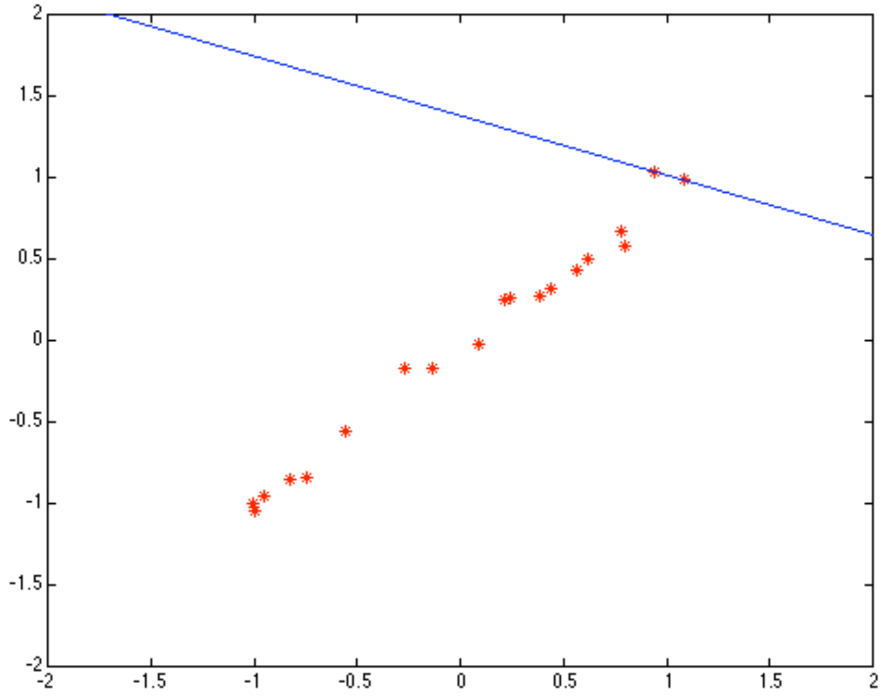
Issues with EM

- Local maxima
  - can be a serious nuisance in some problems
  - no guarantee that we have reached the “right” maximum
- Starting
  - k means to cluster the points is often a good idea

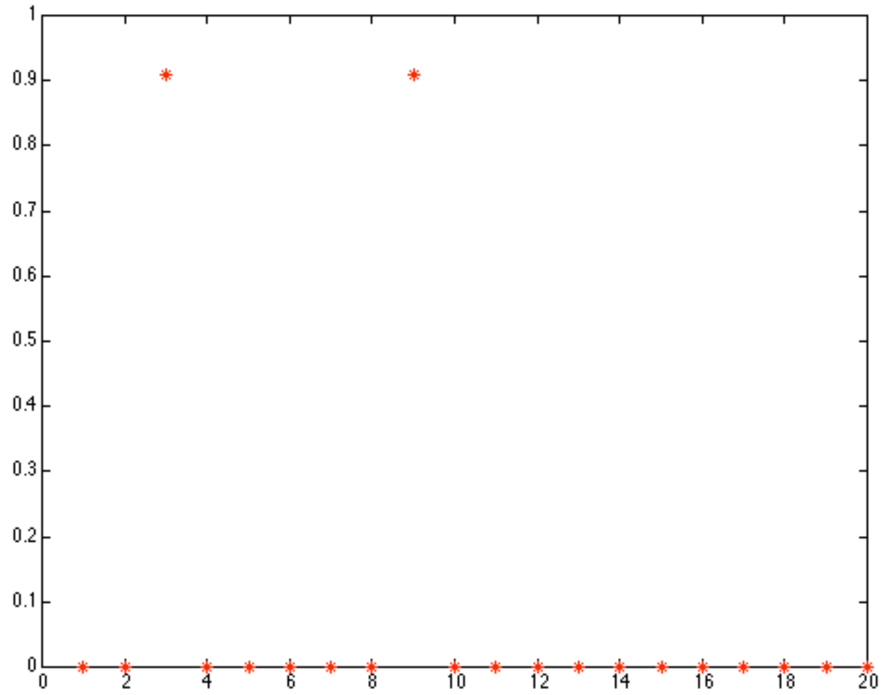
Local maximum



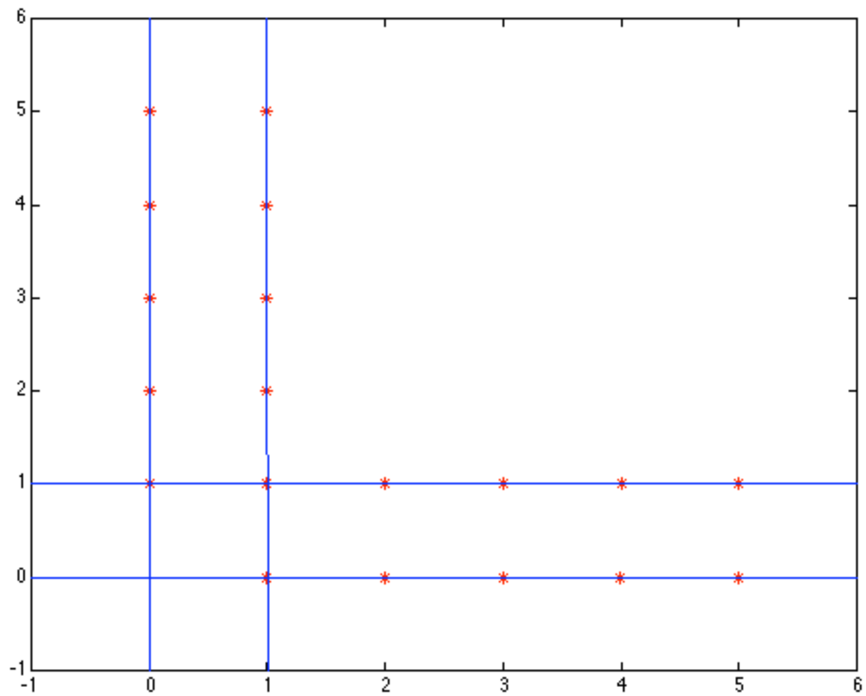
which is an excellent fit to some points



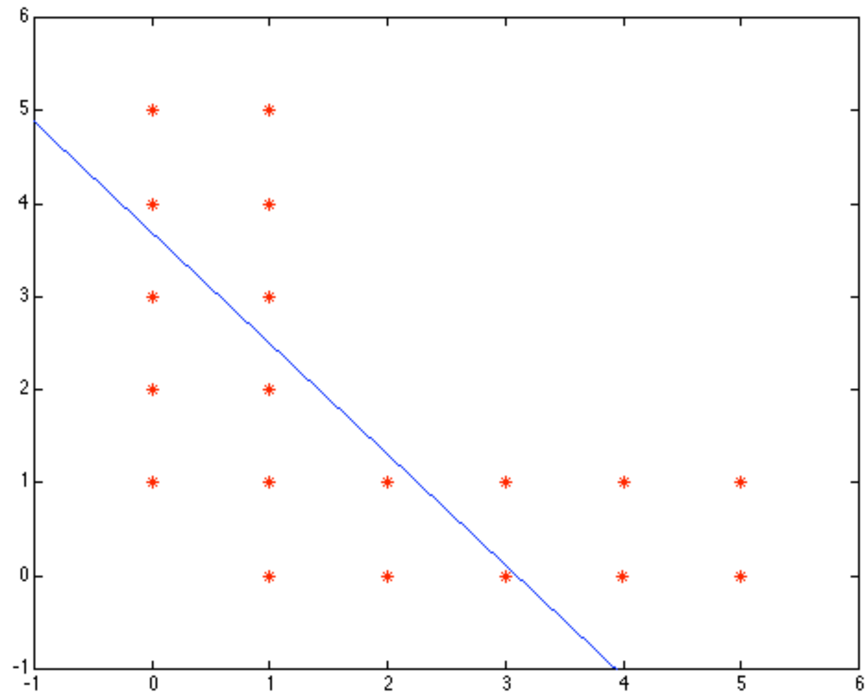
and the deltas for this maximum



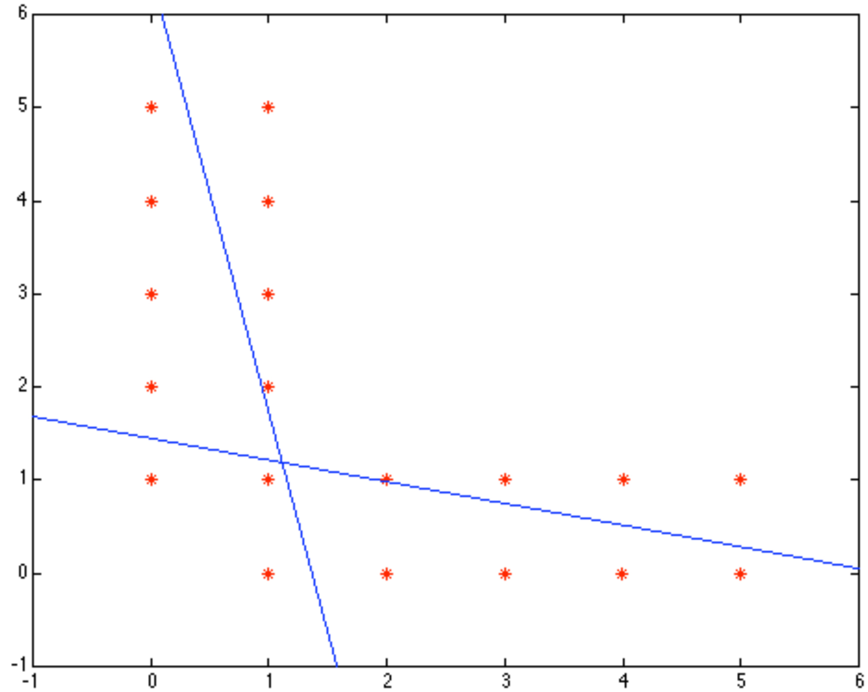
A dataset that is well fitted by four lines



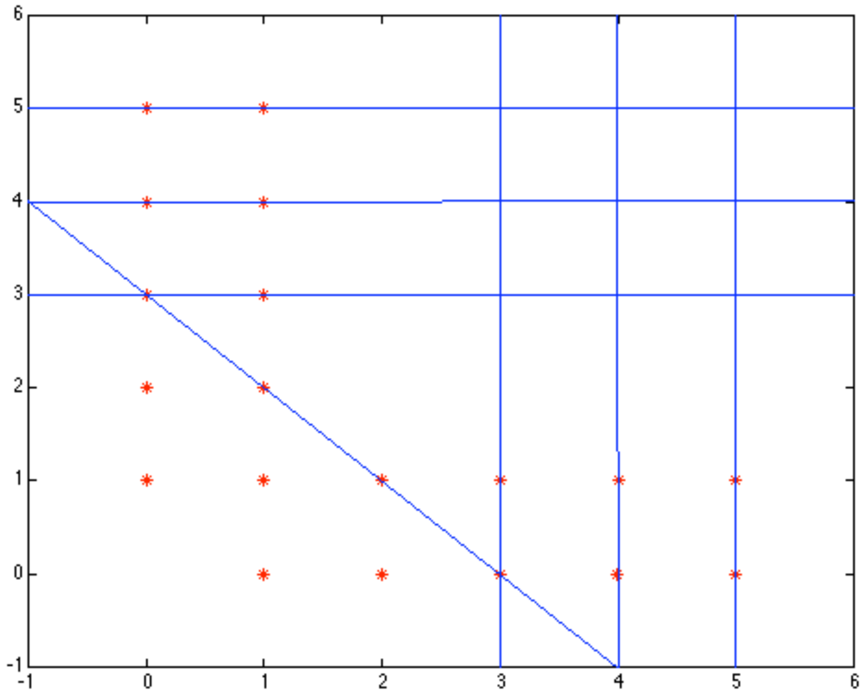
Result of EM fitting, with one line (or at least, one available local maximum).



Result of EM fitting, with two lines (or at least,
one available local maximum).



Seven lines can produce a rather logical answer



Motion segmentation with EM

- Model image pair (or video sequence) as consisting of regions of parametric motion
  - affine motion is popular

$$\begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

  - Now we need to
  - determine which pixels belong to which region
  - estimate parameters
- Likelihood
  - assume

$$\begin{aligned} I(x, y, t) &= I\left(x + v_x, y + v_y, t + 1\right) \\ &\quad + noise \end{aligned}$$

- Straightforward missing variable problem, rest is calculation

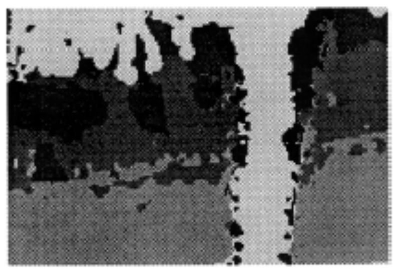
Sequence

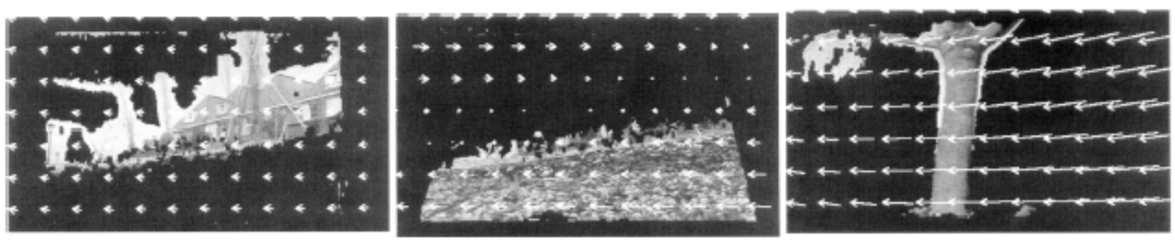
http://www-bcs.mit.edu/people/jyawang/demos/garden-layer/layer-demo.html

Three frames from the MPEG “flower garden” sequence

Figure from “Representing Images with layers,”, by J. Wang and E.H. Adelson, IEEE Transactions on Image Processing, 1994, c 1994, IEEE

Grey level shows region no. with highest probability





Segments and motion fields associated with them

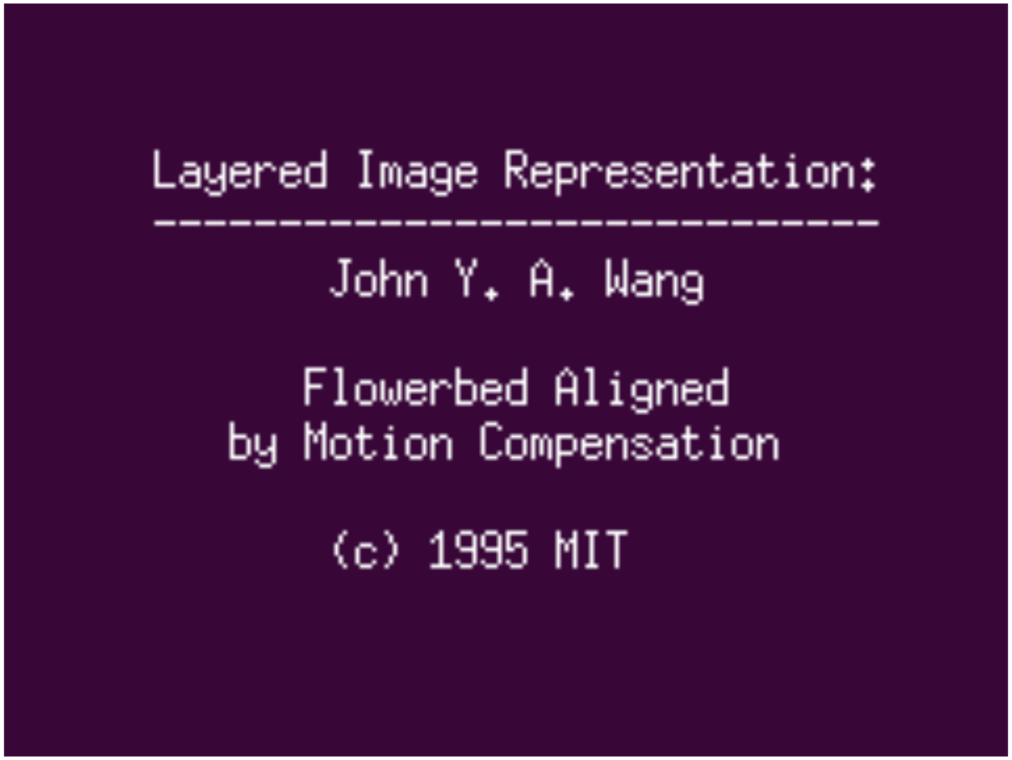
Figure from "Representing Images with layers,", by J. Wang and E.H. Adelson, IEEE Transactions on Image Processing, 1994, c 1994, IEEE

Segments

http://www-bcs.mit.edu/people/jyawang/demos/garden-layer/layer-demo.html

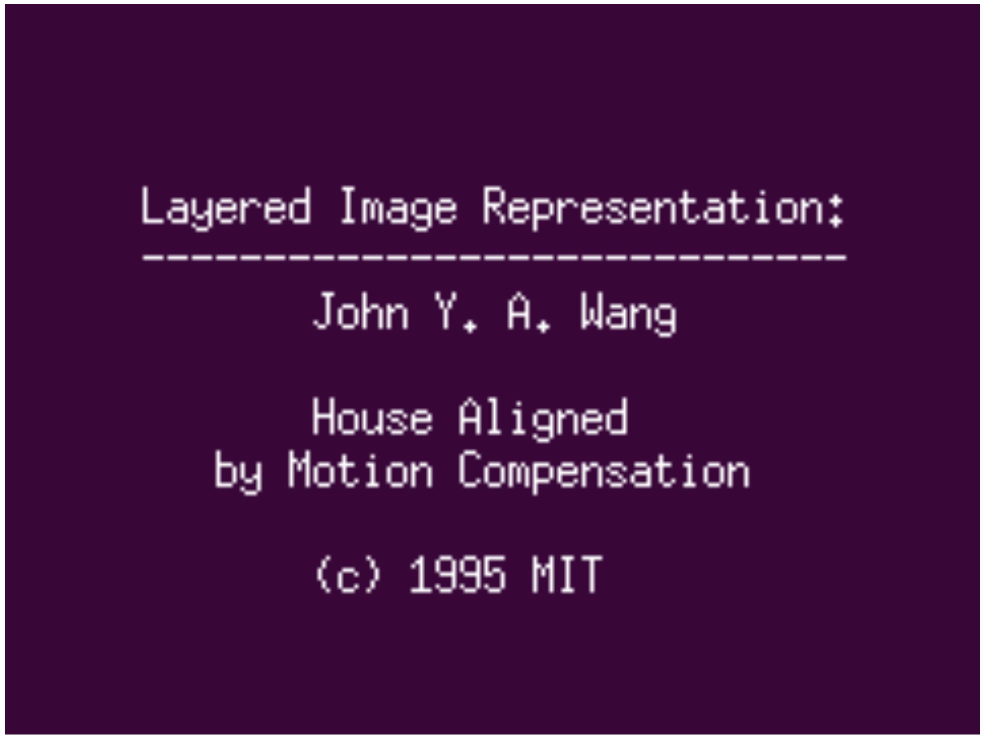
Alignment (motion compensation)

- For each segment in each frame, know a motion model
- We could add a motion to all pixels in the frame that stabilizes the segment



Compensation

http://www-bcs.mit.edu/people/jyawang/demos/garden-layer/layer-demo.html

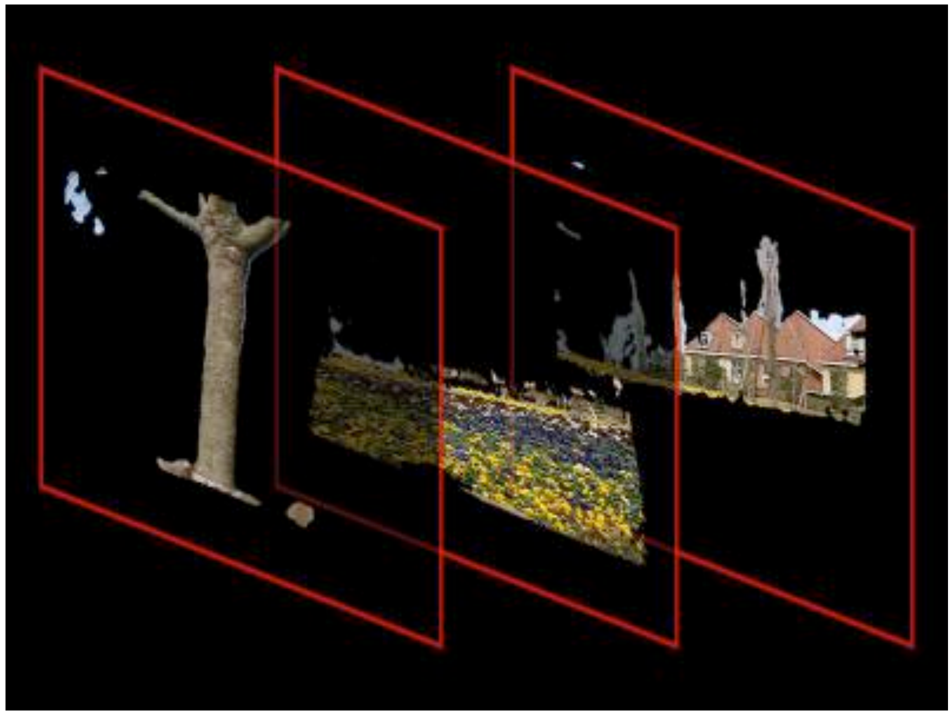


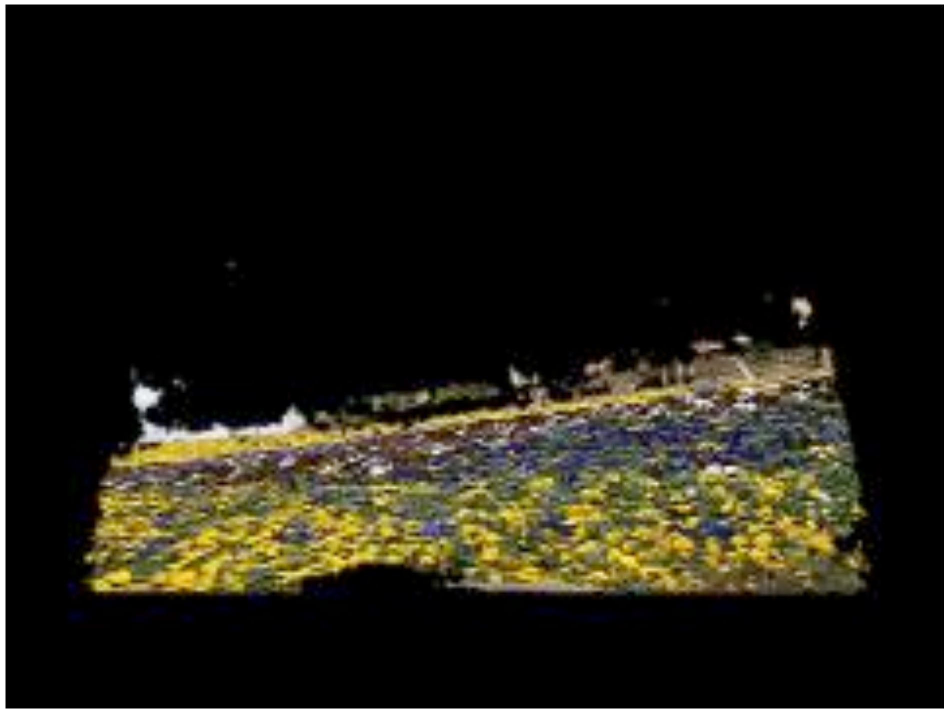
Compensation

http://www-bcs.mit.edu/people/jyawang/demos/garden-layer/layer-demo.html

Compensation

http://www-bcs.mit.edu/people/jyawang/demos/garden-layer/layer-demo.html



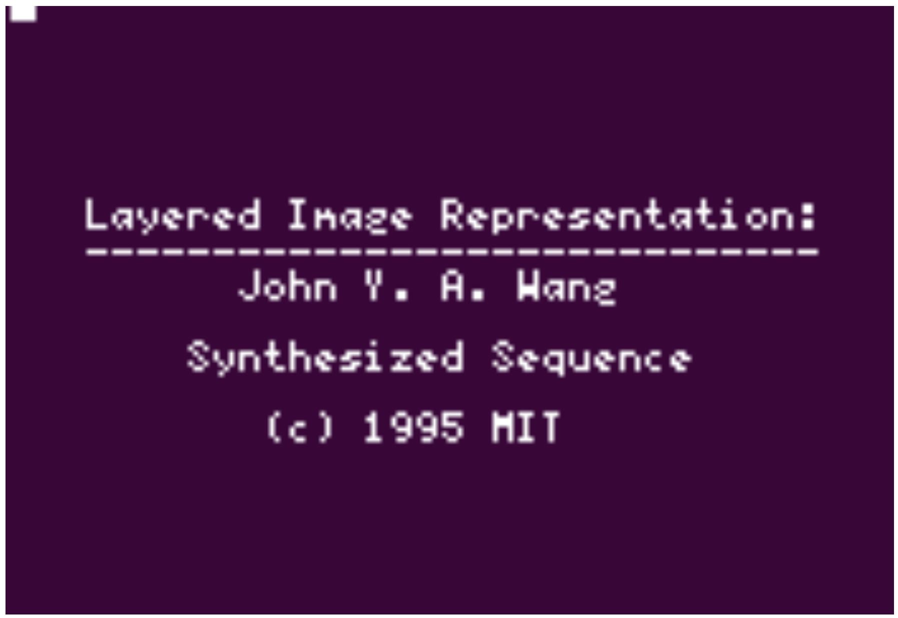


Segment appearance

http://www-bcs.mit.edu/people/jyawang/demos/garden-layer/layer-demo.html

Segment appearance

http://www-bcs.mit.edu/people/jyawang/demos/garden-layer/layer-demo.html



Reassembling segments

http://www-bcs.mit.edu/people/jyawang/demos/garden-layer/layer-demo.html

If we use multiple frames to estimate the appearance
of a segment, we can fill in occlusions; so we can
re-render the sequence with some segments removed.

Figure from “Representing Images with layers,”, by J. Wang and E.H. Adelson, IEEE
Transactions on Image Processing, 1994, c 1994, IEEE

Reassemble w/o tree

http://www-bcs.mit.edu/people/jyawang/demos/garden-layer/layer-demo.html