# Generalized Hough Transform and Chamfer Matching



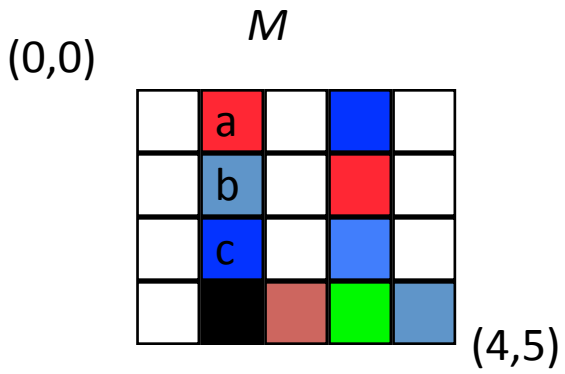
A lot of slides from Larry Davis

# Generalized Hough Transform

- Line and circle finders look for parametric shapes, a line has 2 parameters, a circle has 3, an ellipse 5 and so on
- The Generalized Hough transform can be used to find non-parametric shapes
  - similar to the Hough transform, the outermost loop of the algorithm will perform computations when encountering an edge pixels
- Let *H(i,j)* be an array of counters. This array will be used to accumulate the evidence for the template center/ origin
  - Whenever we encounter an edge pixel we will efficiently determine all placements of the template (*M*) in the edge image (*N*) that would cause an edge point of *M* to be aligned with this point of *N*. These placement will generate indices in *H* to be incremented

# Template representation for the generalized Hough transform

Represent the template *M* as a list of coordinates, *M'*.



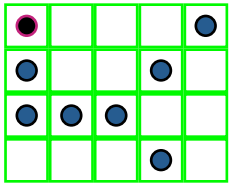

| | *M'* |
|---|---|
| a | (0,-1) |
| b | (-1,-1) |
| c | (-2,-1) |
| | (-3,-1) |
| | (-3,-2) |
| | (-3,-3) |
| | (-2,-3) |
| | (-1,-3) |
| | (0,-3) |

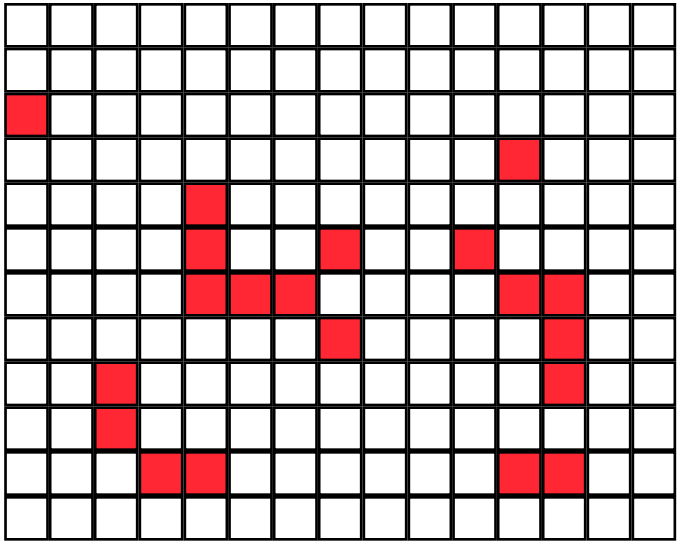
- If we place pixel a over location *(i,j)* in *N*, then the (0,0) location of the template will be at position *(i,j-1)*
- If we place pixel c over location *(i,j)* in *N*, then the (0,0) location of the template will be at position *(i-2,j-1)*

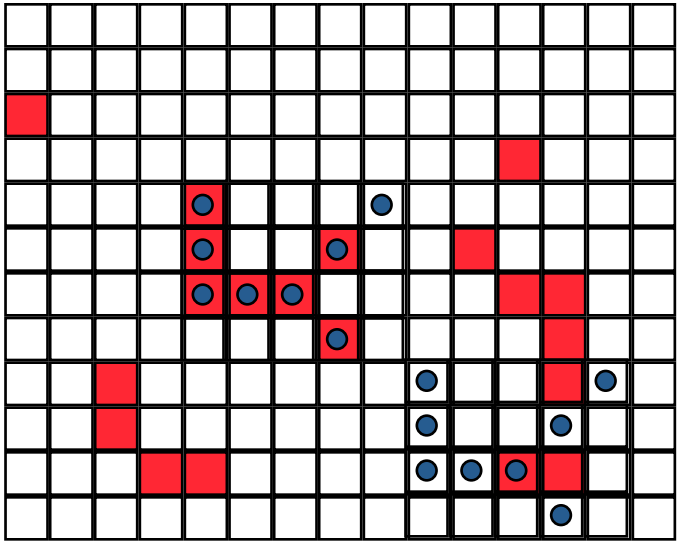
# GHT - basic algorithm

- Scan *N* until a 1 is encountered at position *(x,y)*
  - Iterate through each element *(i,j)* from *M'*
    - The placement of *M* over *N* that would have brought *M(i,j)* over *N(x,y)* is the one for which the origin of *M* is placed at position *(x+i, y+j)*.
    - Therefore, we increment *H(x+i, y+j)* by 1
  - And move on to the next element of *M'*
- And move on to the next 1 in *N*
- When the algorithm completes *H(i,j)* counts the number of template points that would overlay a "1" in *N* if the template were placed at position *(i,j)* in *N*.

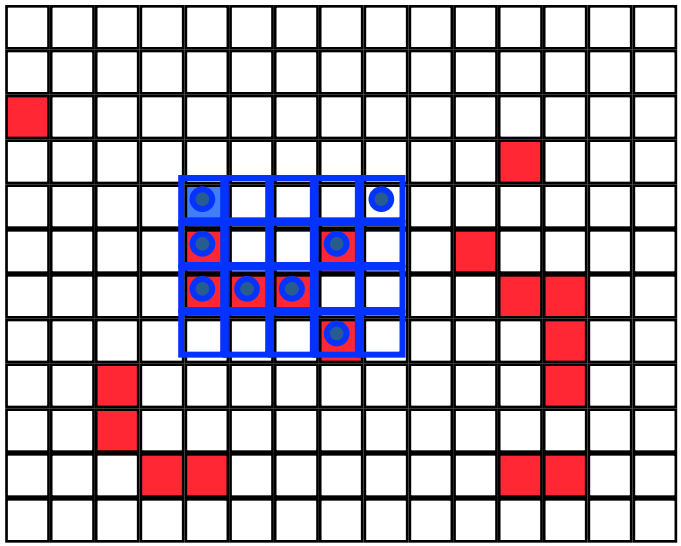
# Example



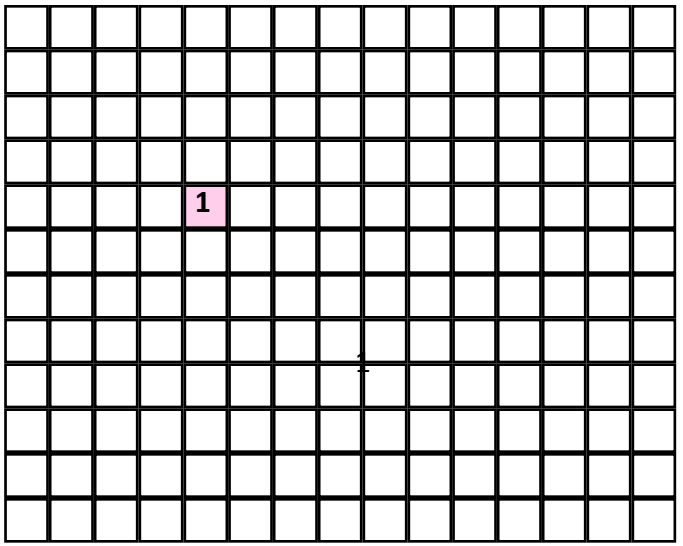




# Example

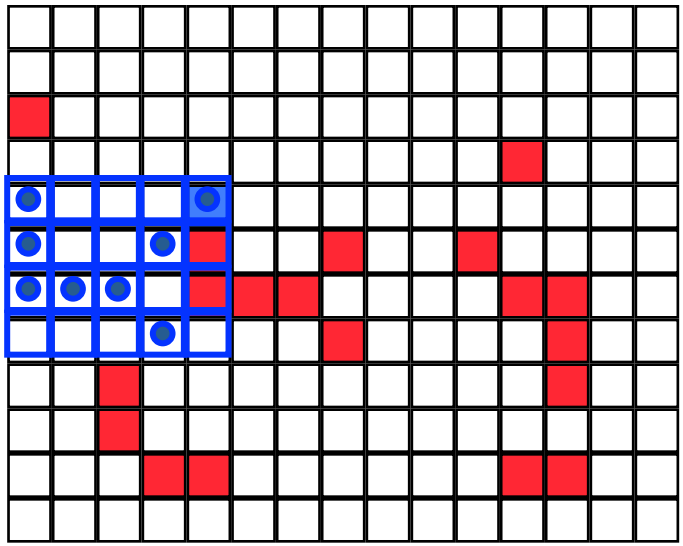


Image

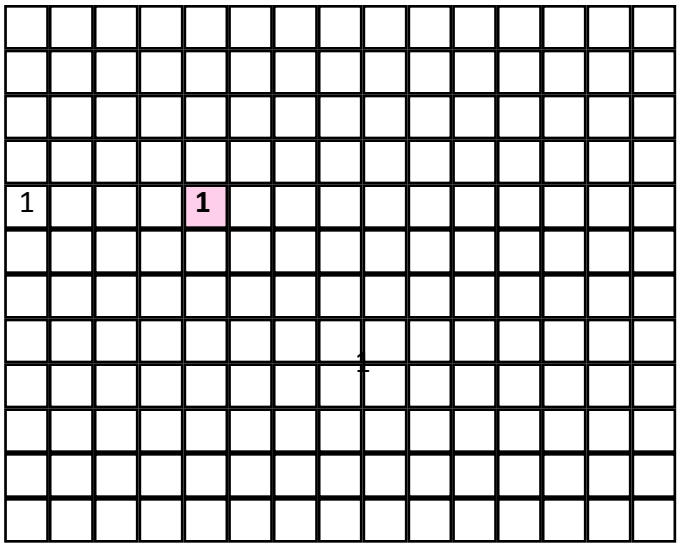



Hough array

# Example

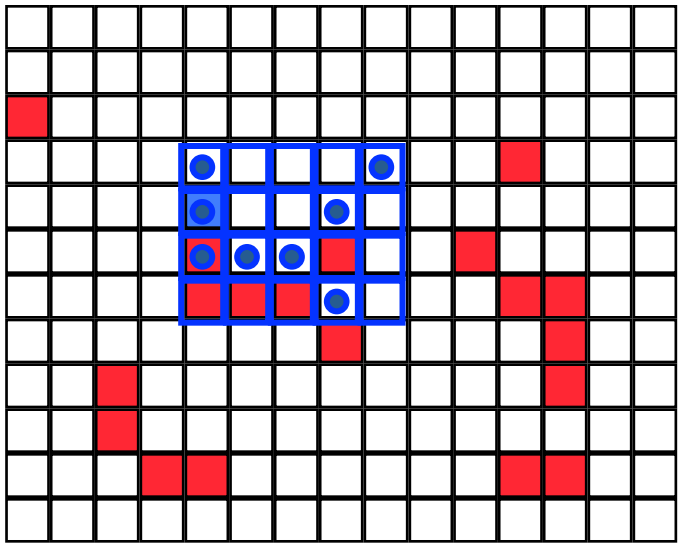


Image

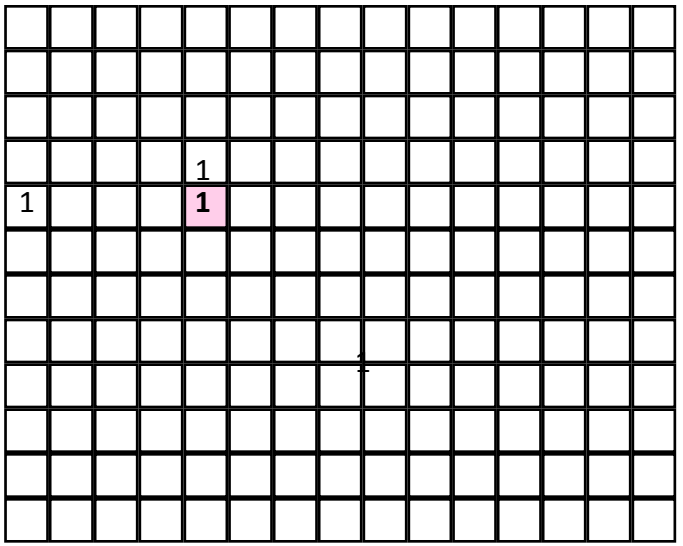



Hough array

# Example

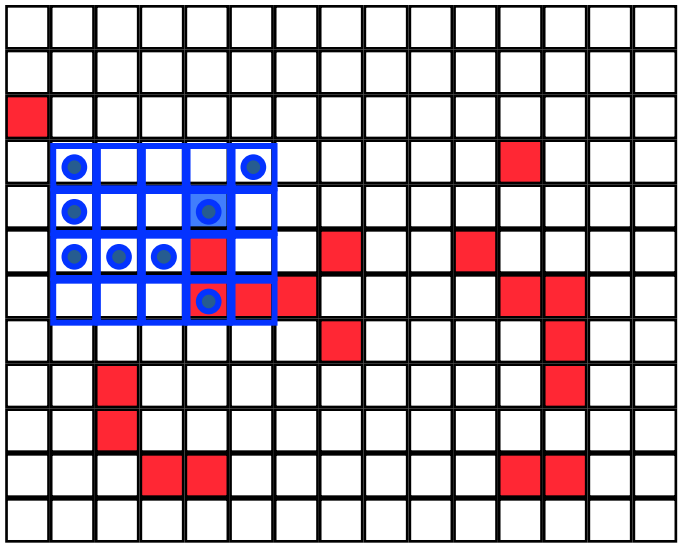


Image

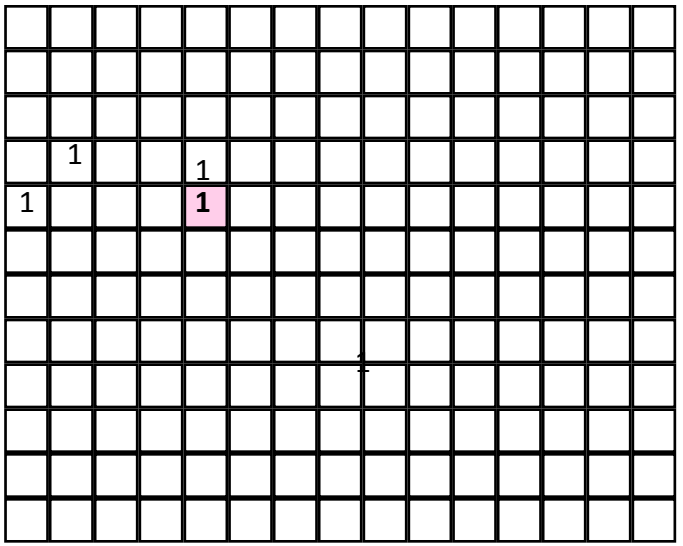



Hough array

# Example

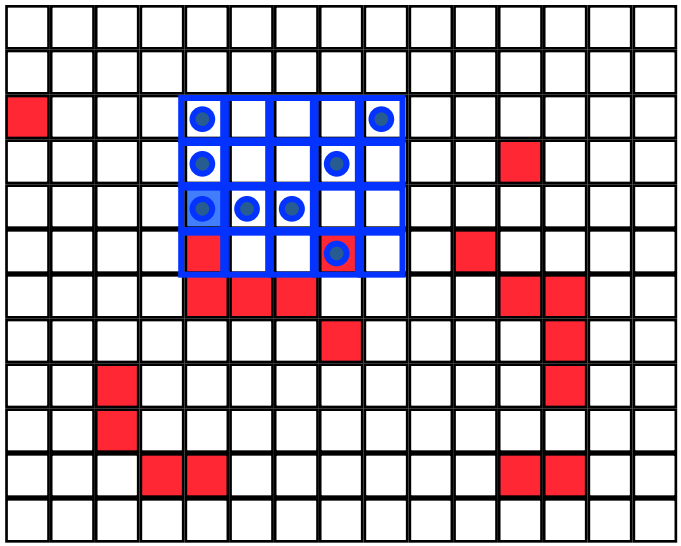


Image

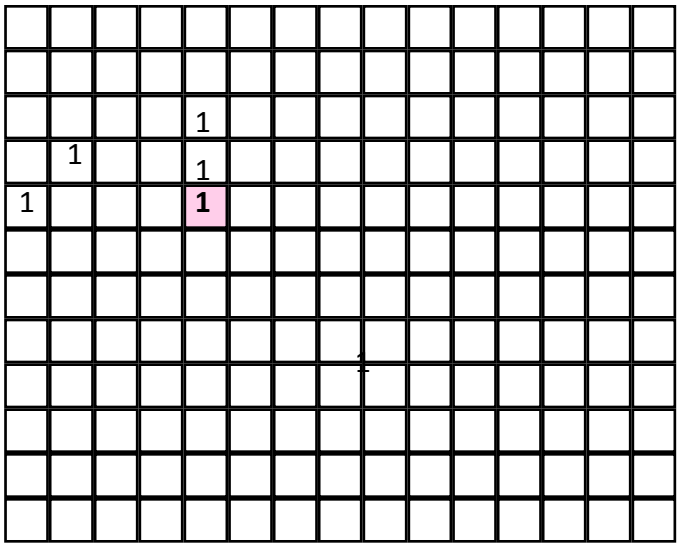



Hough array

# Example

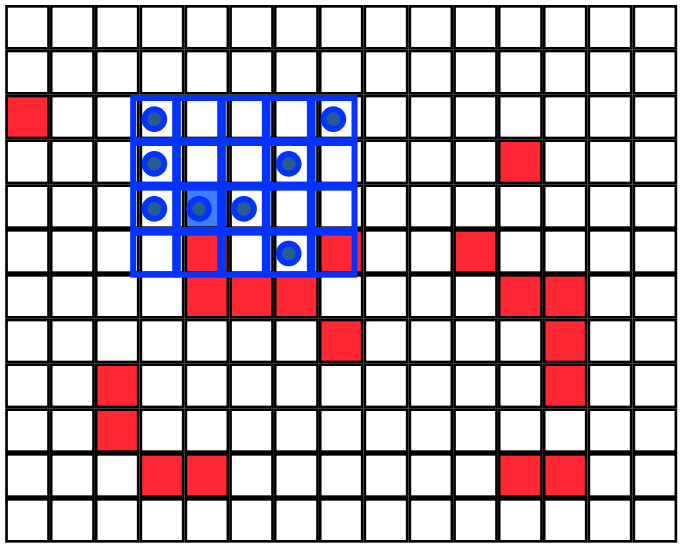


Image

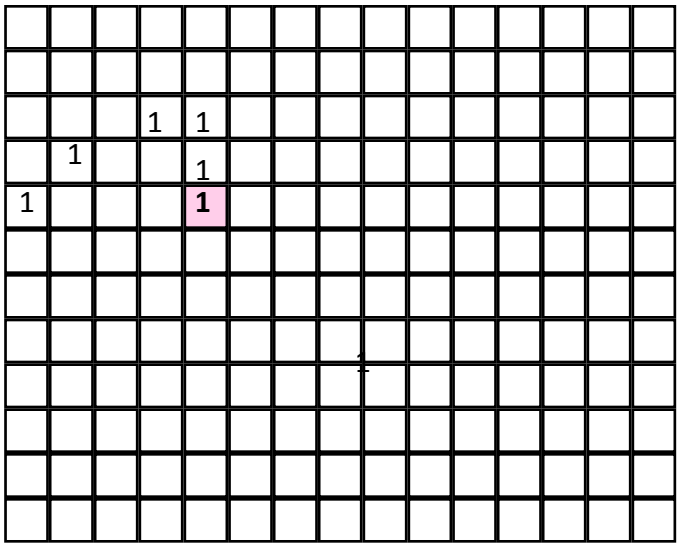



Hough array

# Example

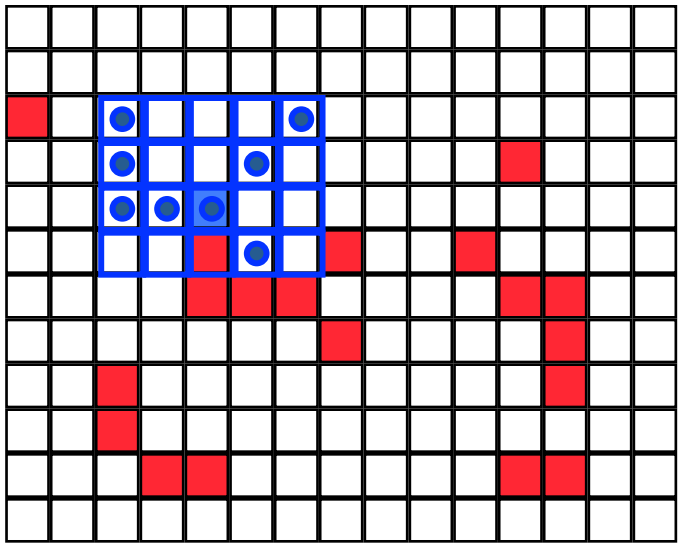


Image

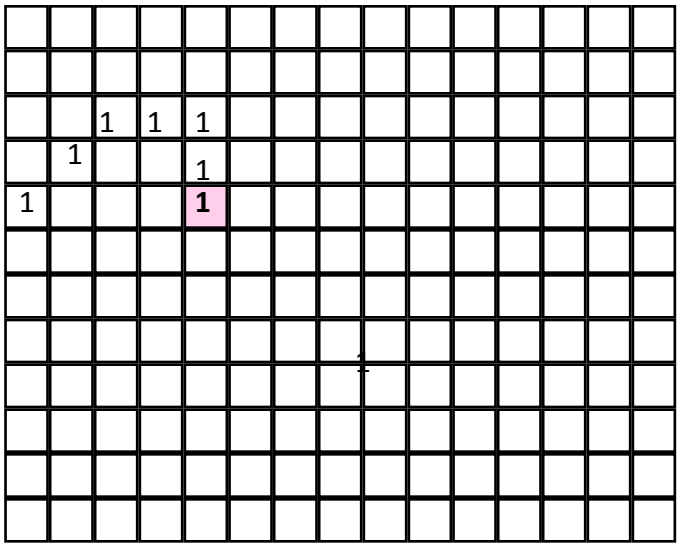



Hough array

# Example

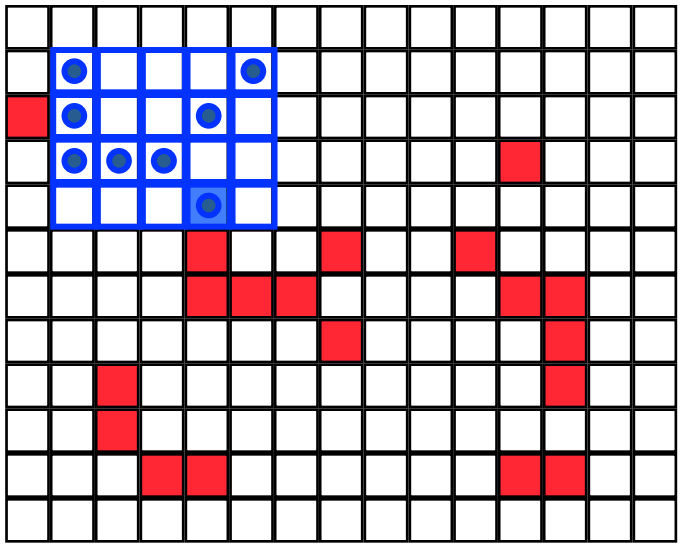


Image

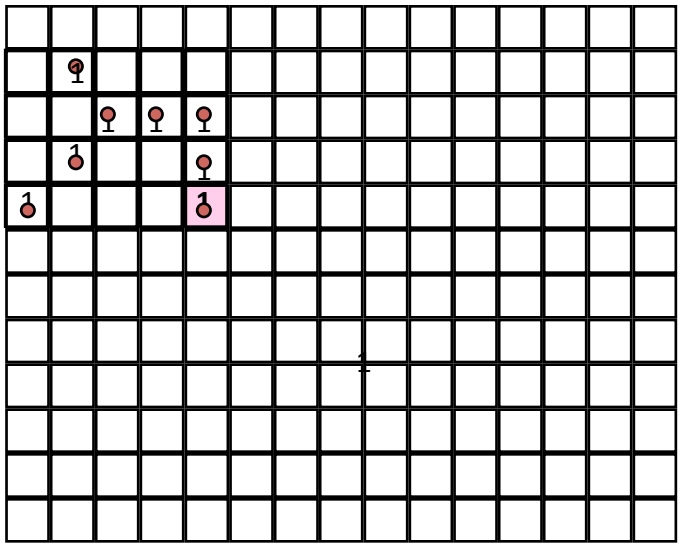



Hough array

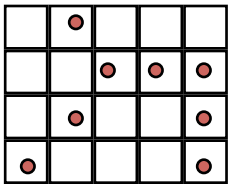
# Example



Image



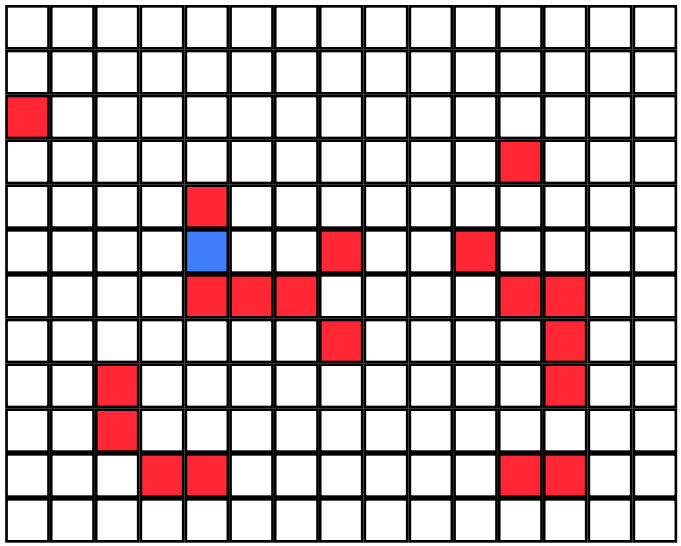
Hough array



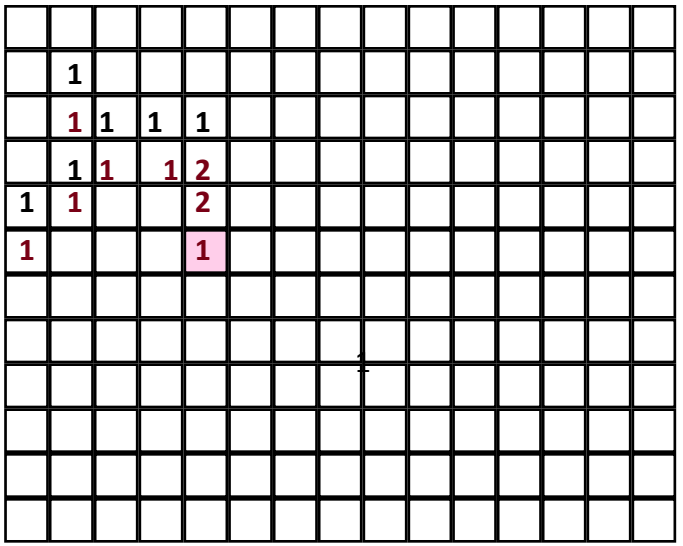
Template

Increment pattern

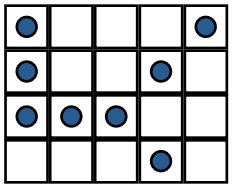
# Example



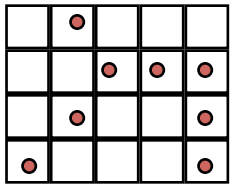
Image




Hough array

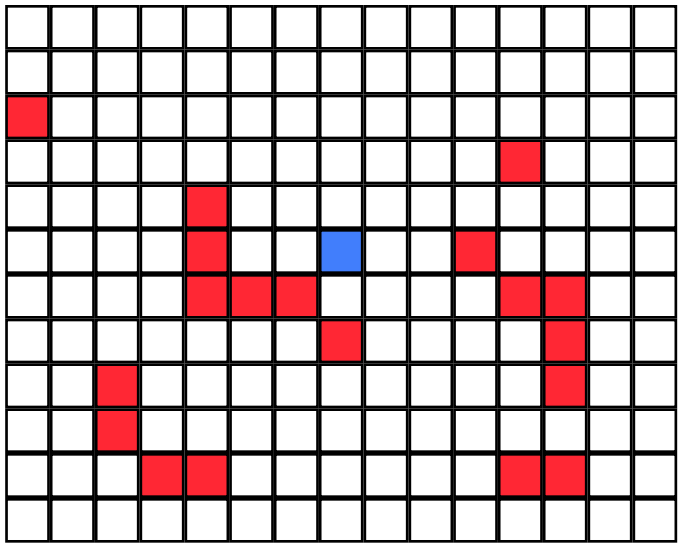


Template

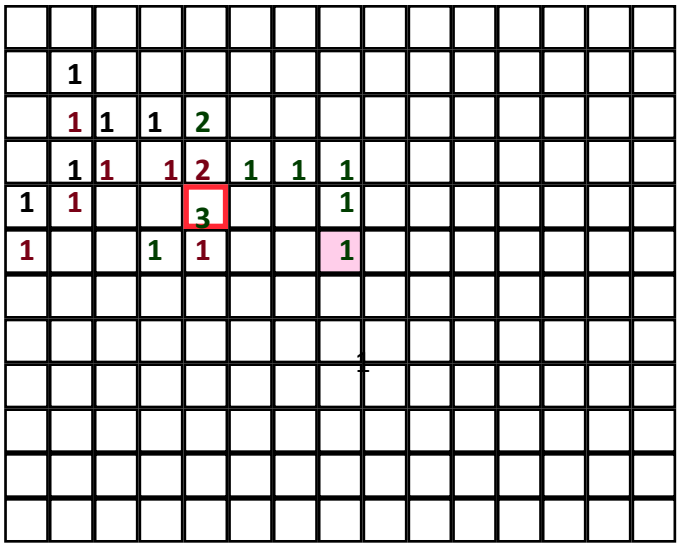


Increment pattern

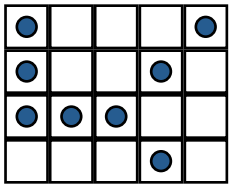
# Example



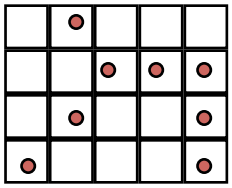
Image




Hough array

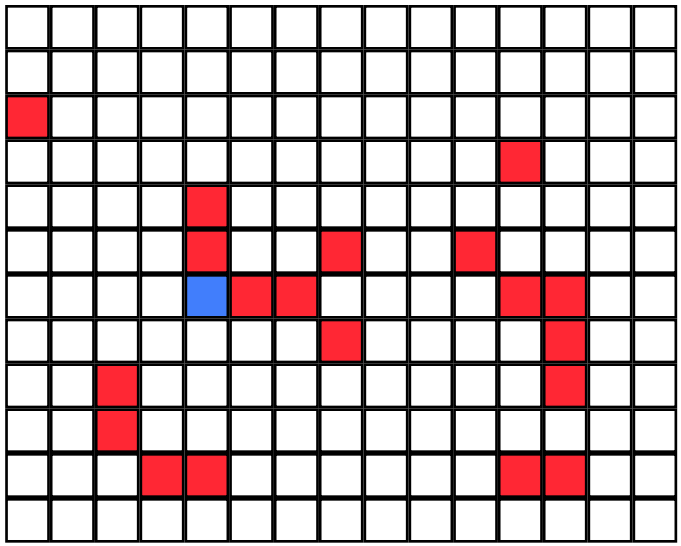


Template

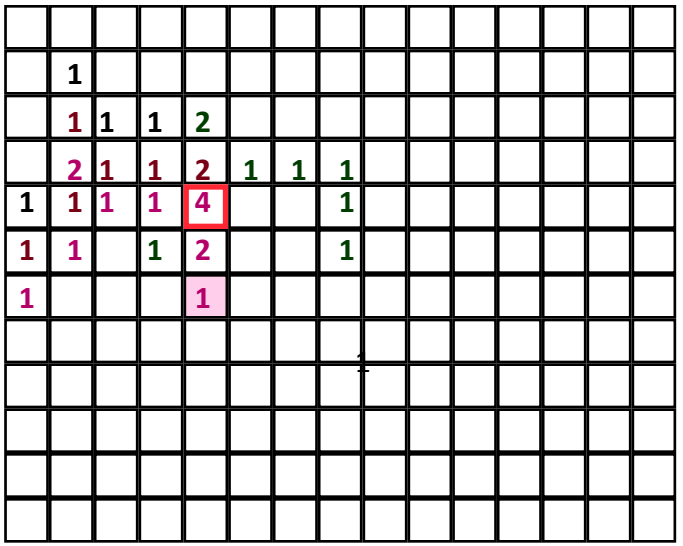


Increment pattern

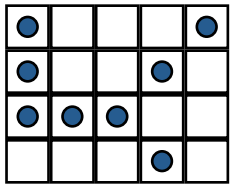
# Example



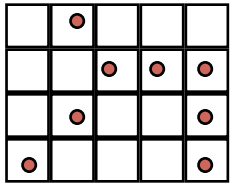
Image




Hough array



Template



Increment pattern

# GHT - generalizations

- Suppose we want to detect instances of *M* that vary in orientation in the image
  - need to increase the dimensionality of *H* by adding a dimension, $\theta$, for orientation
  - Now, each time we encounter a 1 during the scan of *N* we must consider all possible rotations of *M* wrt *N* - will result in incrementing one counter in each $\theta$ plane of *H* for each point in *M*.
    - For each *(i,j)* from *M*
      - For each quantized $\theta$
      - Determine the placement *(r,s)* of the rotated template in *N* that would bring *(i,j)* onto *(x,y)* and increment *H(r,s,*$\theta$*)*
- For scale we would have to add one more dimension to *H* and another loop that considers possible scales of *M*.

# Representing high dimensional Hough arrays

- Problems with high dimensional arrays:
  - storage
  - initialization and searching for high values after algorithm
- Possible solutions
  - Hierarchical representations
    - first match using a coarse resolution Hough array
    - then selectively expand parts of the array having high matches
  - Projections
    - Instead of having one high dimensional array store a few two dimensional projections with common coordinates (e.g., store *(x,y), (y,*$\theta$*), (*$\theta$*,s)* and *(s,x)*)
    - Finds consistent peaks in these lower dimensional arrays

# GHT generate and test

- Peaks in Hough array do not reflect spatial distribution of points underlying match
  - typical to "test" the quality of peak by explicitly matching template against image at the peak
- Controlling the generate and test framework
  - construct the complete Hough array, find peaks, and test them
  - test as soon as a point in the Hough space passes a threshold
    - if the match succeeds, points in I that matched can be eliminated from further testing
  - test as soon as a point in the Hough space is incremented **even once**

# Chamfer matching

- Given:
  - binary image, *B*, of edge and local feature locations
  - binary "edge template", *T*, of shape we want to match
- Let *D* be an array in registration with *B* such that *D(i,j)* is the distance to the nearest "1" in *B*.
  - this array is called the **distance transform** of *B*
- Goal: Find placement of *T* in *D* that minimizes the sum, *M*, of the distance transform multiplied by the pixel values in *T*
  - if *T* is an exact match to *B* at location *(i,j)* then *M(i,j) = 0*
  - but if the edges in *B* are slightly displaced from their ideal locations in *T*, we still get a good match using the distance transform technique

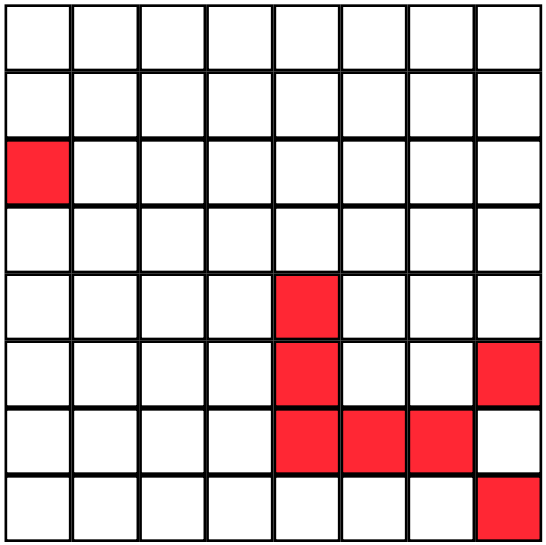
# Computing the distance transform

- Brute force, exact algorithm, is to scan *B* and find, for each "0", its closest "1" using the Euclidean distance.
  - expensive in time, and difficult to implement

# Computing the distance transform

- Two pass sequential algorithm
- Initialize: set $D(i,j) = 0$ where $B(i,j) = 1$, else set $D(i,j) = \infty$
- Forward pass
  - $D(i,j) = \min\{D(i-1,j-1) + 4, D(i-1,j) + 3, D(i-1, j+1) +4, D(i,j-1) + 3, D(i,j)\}$
- Backward pass
  - $D(i,j) = \min\{D(i,j+1) + 3, D(i+1,j-1) + 4, D(i+1, j) +3, D(i+1,j+1) + 4, D(i,j)\}$

# Distance transform example



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 2 | 2 | 3 | 0 | 1 | 2 | 3 |
| 3 | 3 | 3 | 1 | 0 | 1 | 2 | 0 |
| 4 | 4 | 2 | 1 | 0 | 0 | 0 | 1 |
| 5 | 3 | 2 | 1 | 1 | 1 | 1 | 0 |

| | | |
|---|---|---|
| f | f | f |
| f | | b |
| b | b | b |

# Distance transform example

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 2 | 2 | 3 | 0 | 1 | 2 | 3 |
| 3 | 3 | 3 | 1 | 0 | 1 | 2 | 0 |
| 4 | 4 | 2 | 1 | 0 | 0 | 0 | 1 |
| 5 | 3 | 2 | 1 | 1 | 1 | 1 | 0 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 |
| 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 |
| 0 | 1 | 2 | 2 | 2 | 2 | 2 | 3 |
| 1 | 1 | 2 | 1 | 1 | 1 | 2 | 2 |
| 2 | 2 | 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 3 | 2 | 1 | 0 | 1 | 1 | 0 |
| 4 | 3 | 2 | 1 | 0 | 0 | 0 | 1 |
| 4 | 3 | 2 | 1 | 1 | 1 | 1 | 0 |

*D(i,j)* = min{*D(i,j), D(i,j+1)+1, D(I+1, j-1)+1, D(i+1,j)+1, D(i+1,j+1)+1}*

| | | |
|---|---|---|
| f | f | f |
| f | | b |
| b | b | b |

# Distance Transforms

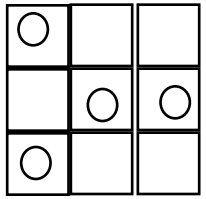
- Note that it is possible to computer Euclidean distances in linear time – available in OpenCV
- In the example presented here various update templates can be used - *(3,4)* and *(1,1)* are two examples used here

# Chamfer matching

- Chamfer matching is convolution of a binary edge template with the distance transform
  - for any placement of the template over the image, it sums up the distance transform values for all pixels that are "1's" (edges) in the template
  - if, at some position in the image, all of the edges in the template coincide with edges in the image (which are the points at which the distance transform is zero), then we have a perfect match with a match score of 0.

# Example

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 |
| 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 |
| 0 | 1 | 2 | 2 | 2 | 2 | 2 | 3 |
| 1 | 1 | 2 | 1 | 1 | 1 | 2 | 2 |
| 2 | 2 | 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 3 | 2 | 1 | 0 | 1 | 1 | 0 |
| 4 | 3 | 2 | 1 | 0 | 0 | 0 | 1 |
| 4 | 3 | 2 | 1 | 1 | 1 | 1 | 0 |



Template

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 |
| 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 |
| 0 | 1 | 2 | 2 | 2 | 2 | 2 | 3 |
| 1 | 1 | 2 | 1 | 1 | 1 | 2 | 2 |
| 2 | 2 | 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 3 | 2 | 1 | 0 | 1 | 1 | 0 |
| 4 | 3 | 2 | 1 | 0 | 0 | 0 | 1 |
| 4 | 3 | 2 | 1 | 1 | 1 | 1 | 0 |

Match score is $$\sum_{k=1}^{n}\sum_{l=1}^{n} T(i,j)D(i+k,j+l)$$