ECE5554 – Computer Vision
Lecture 6a – Contours

Creed Jones, PhD

Course Update

- HW3 is due this Wednesday at 11:59 PM
- Quiz 3 is TOMORROW
  - 6 PM to 3 AM (Wednesday morning) Eastern time
  - Eight questions this time, twenty minutes
  - Lectures 5 and 6
- SPOT surveys on this course will open soon
  - open from August 6 through August 12
  - participation is completely anonymous and completely voluntary
  - I would appreciate your responses – especially comments that I can act on!

Today's Objectives

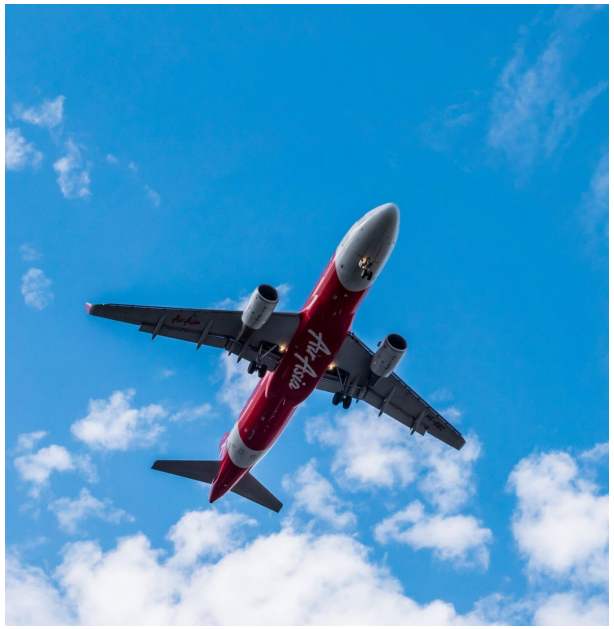
Contours in Binary Images

- Contour tracing
  - the Theo Pavlidis algorithm
- Chain code
- Convex Hull

CONTOURS IN BINARY IMAGES

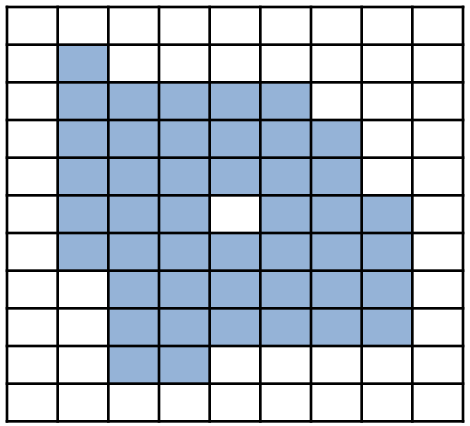
In a binary image, the only values are 0 and 1; We often represent this as 0 and 255, so the "background" is total black and the foreground is total white

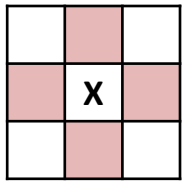
- From a general grayscale image, there are a variety of ways to assign each pixel as 1 (foreground) or 0 (background)
  - We will talk about this next week
- Conceptually, objects of interest are white (1) and the background is black (0)

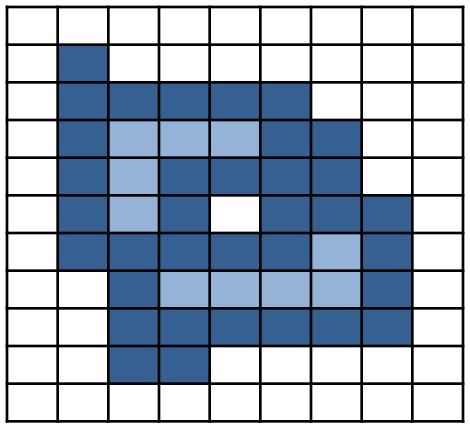


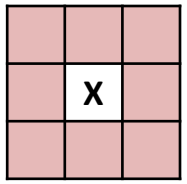
The *contour* of a binary region is the set of pixels around the exterior, considered or stored in sequence; there are many variations on the technique

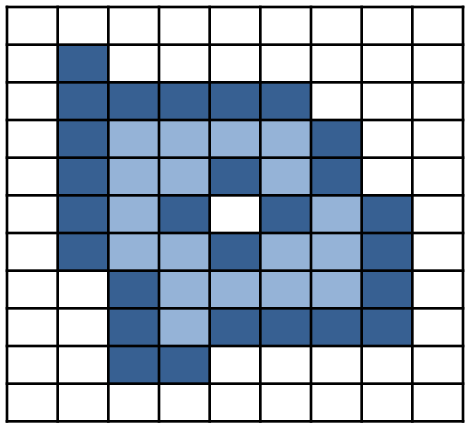
- The *outer contour* is the ordered set of pixels around the outer boundary of the object
- The *inner contour* is the collection of the contours of all enclosed holes in the object
- We can consider 4-connection or 8-connection (more common) when defining the contours





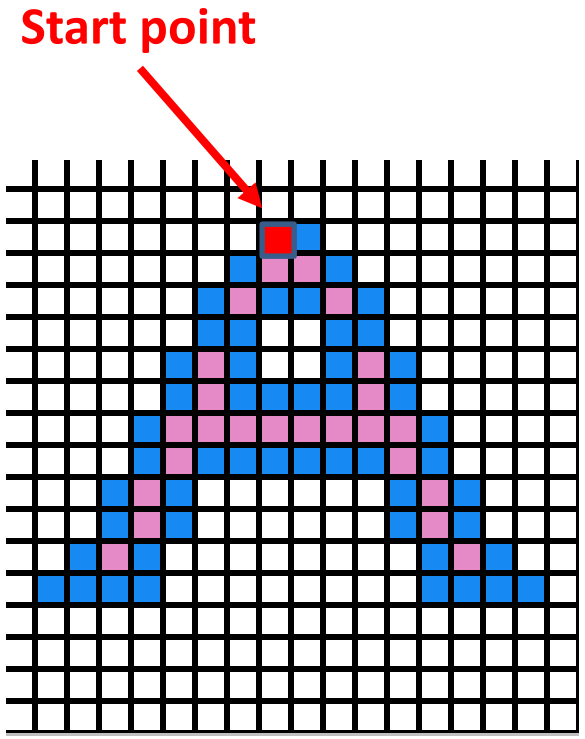






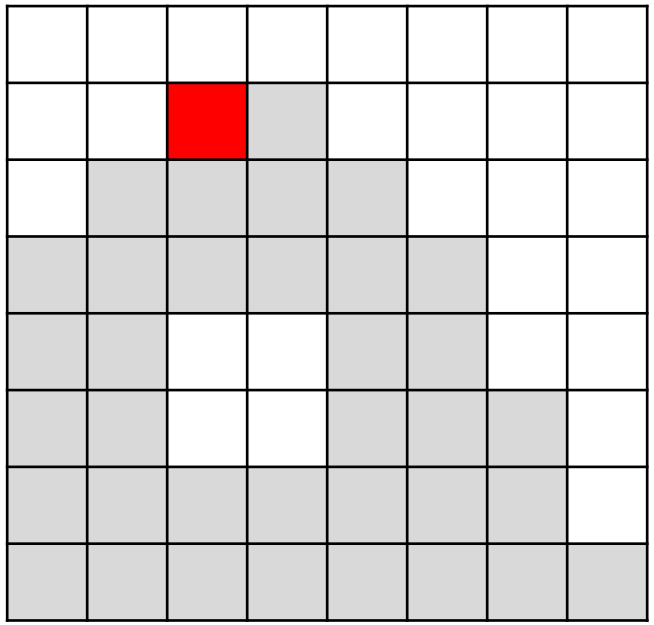
It's important to realize that the contour is an ordered sequence – we don't just want an image of the boundary pixels, but the sequence of moving between them

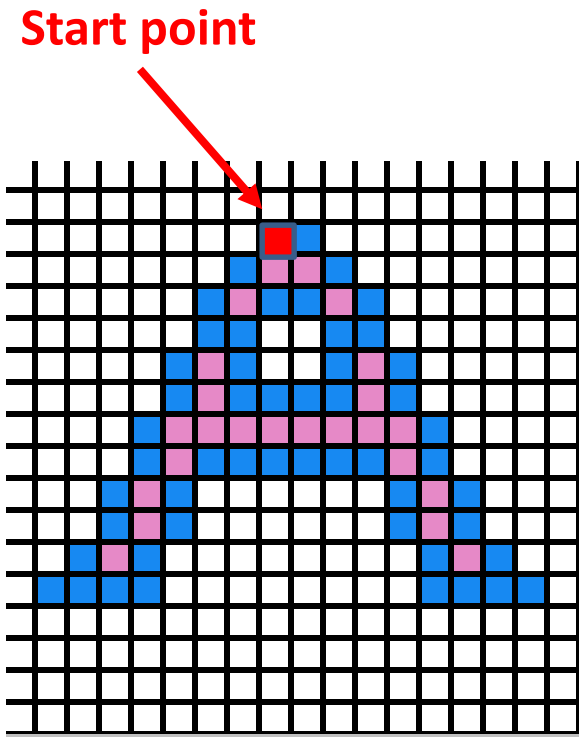
- What I want is something like:
- (1, 0)
- (1, 1)
- (1, 1)
- (0, 1)
- (1, 1)
- (0, 1)
- (1, 1)
- etc…



It's important to realize that the contour is an ordered sequence – we don't just want an image of the boundary pixels, but the sequence of moving between them

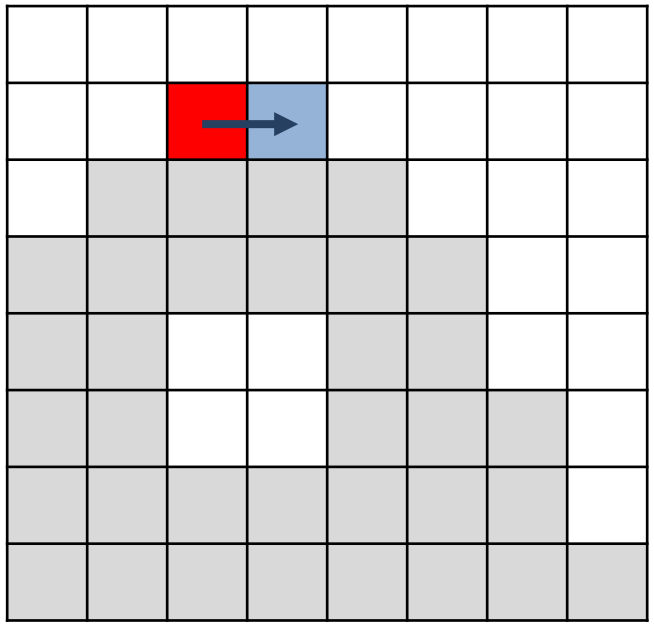
- What I want is something like:
- (1, 0)
- (1, 1)
- (1, 1)
- (0, 1)
- (1, 1)
- (0, 1)
- (1, 1)
- etc…

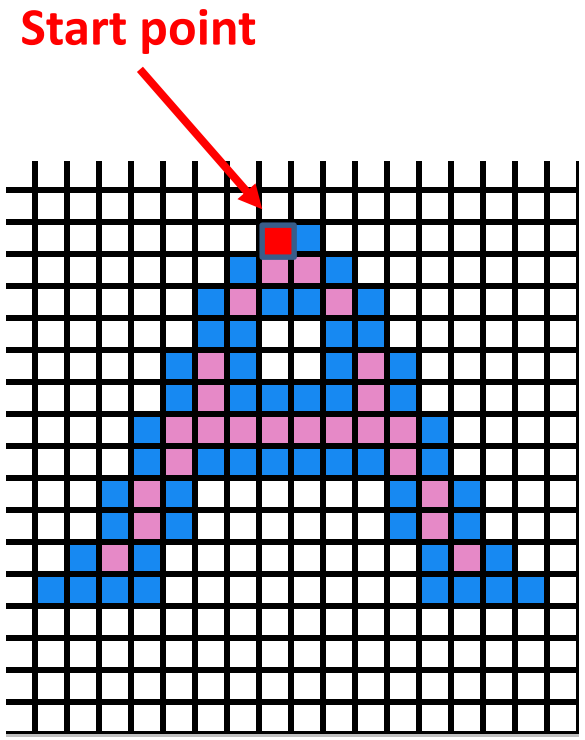




It's important to realize that the contour is an ordered sequence – we don't just want an image of the boundary pixels, but the sequence of moving between them

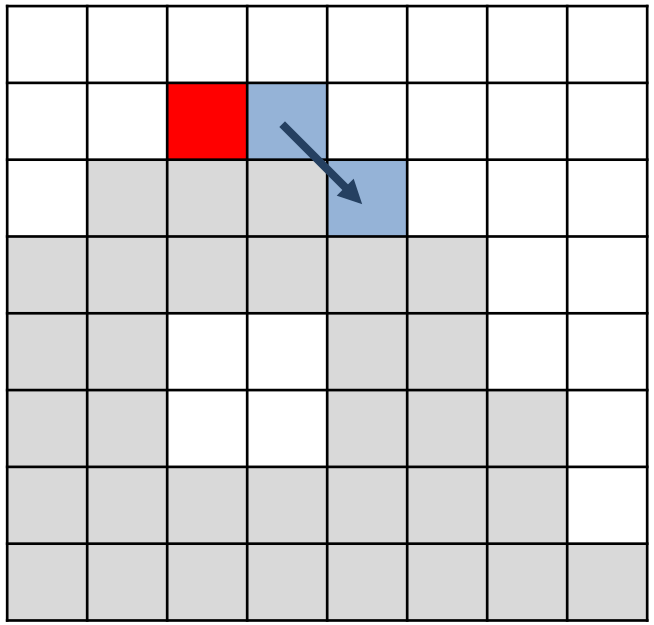
- What I want is something like:
- (1, 0) ✓
- (1, 1)
- (1, 1)
- (0, 1)
- (1, 1)
- (0, 1)
- (1, 1)
- etc…

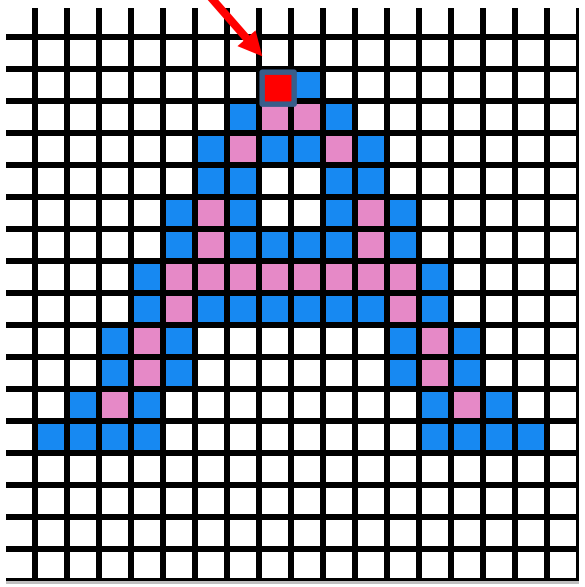




It's important to realize that the contour is an ordered sequence – we don't just want an image of the boundary pixels, but the sequence of moving between them

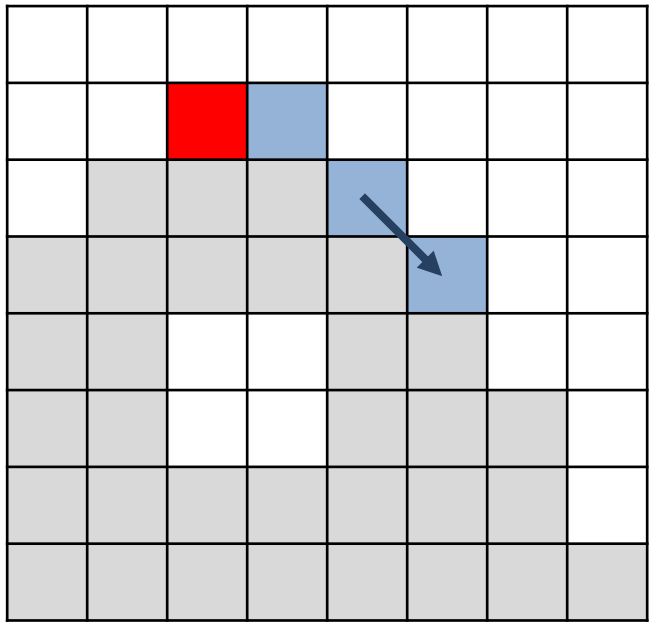
- What I want is something like:
- (1, 0) ✓
- (1, 1) ✓
- (1, 1)
- (0, 1)
- (1, 1)
- (0, 1)
- (1, 1)
- etc…

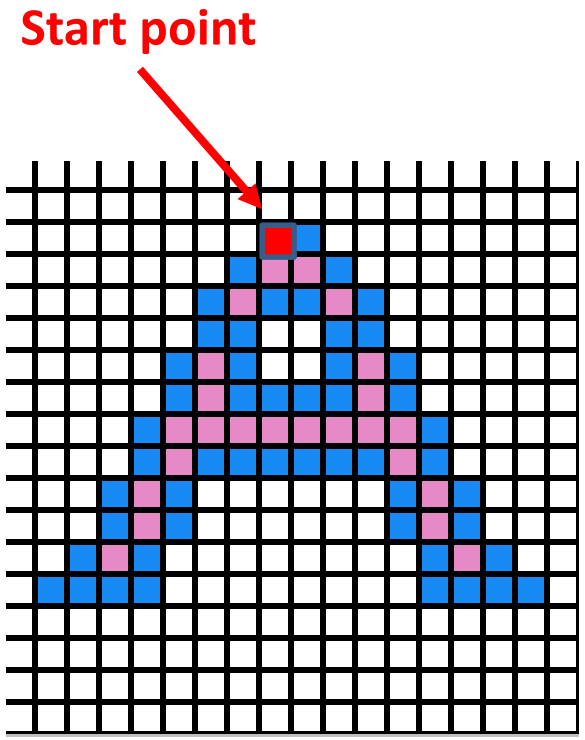




It's important to realize that the contour is an ordered sequence – we don't just want an image of the boundary pixels, but the sequence of moving between them

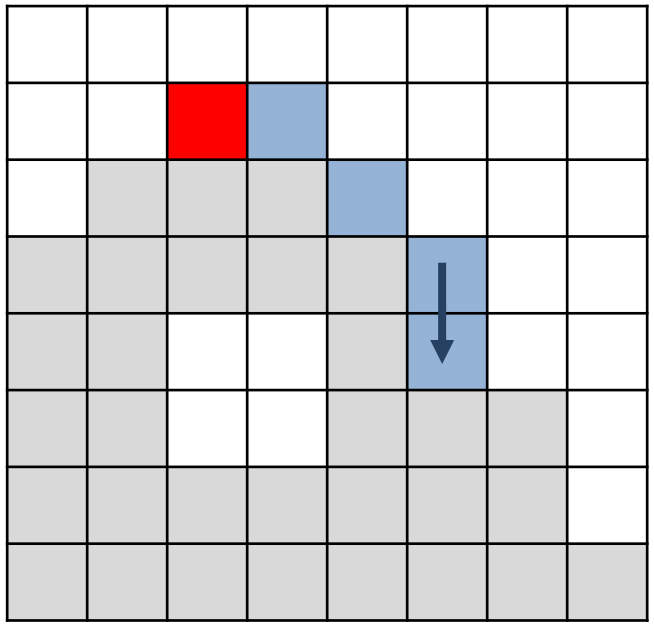
- What I want is something like:
- (1, 0) ✓
- (1, 1) ✓
- (1, 1) ✓
- (0, 1)
- (1, 1)
- (0, 1)
- (1, 1)
- etc…

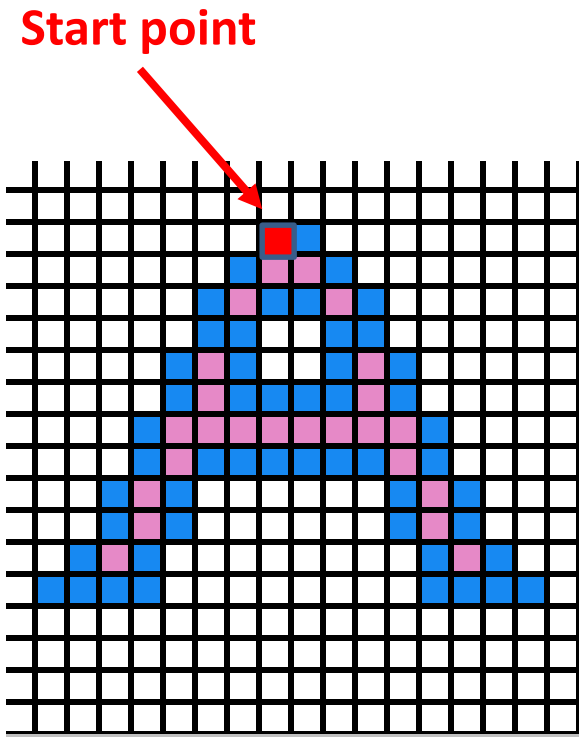




It's important to realize that the contour is an ordered sequence – we don't just want an image of the boundary pixels, but the sequence of moving between them

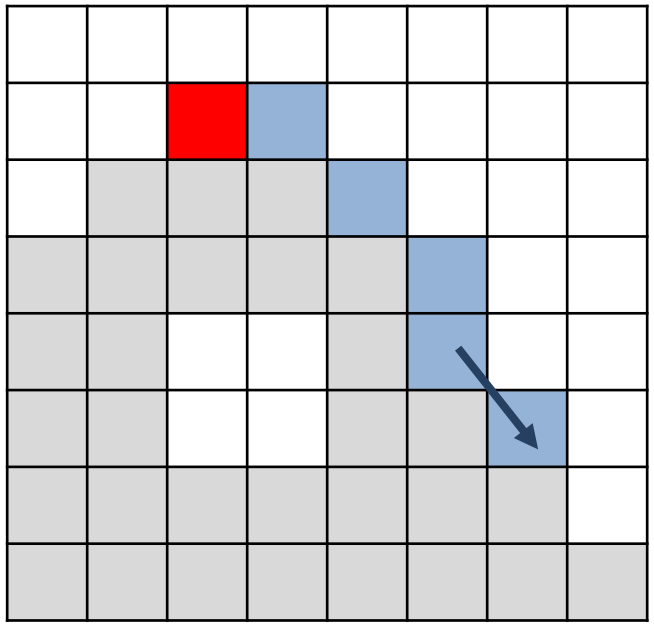
- What I want is something like:
- (1, 0) ✓
- (1, 1) ✓
- (1, 1) ✓
- (0, 1) ✓
- (1, 1)
- (0, 1)
- (1, 1)
- etc…

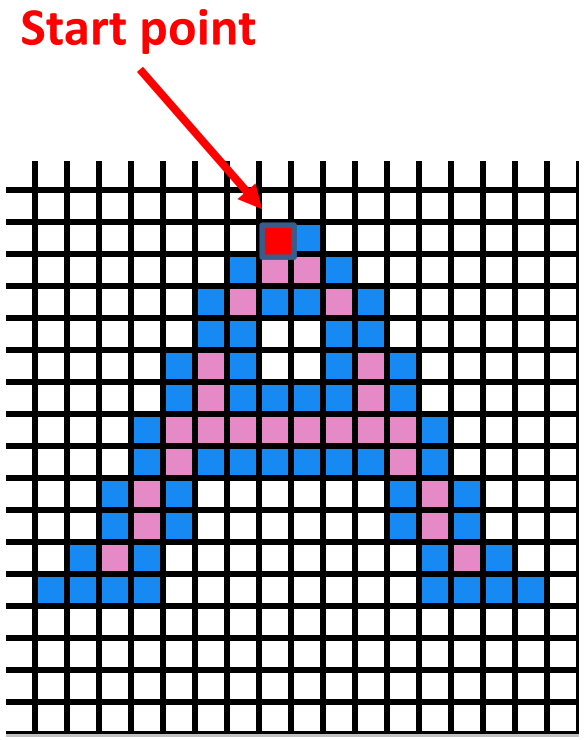




It's important to realize that the contour is an ordered sequence – we don't just want an image of the boundary pixels, but the sequence of moving between them

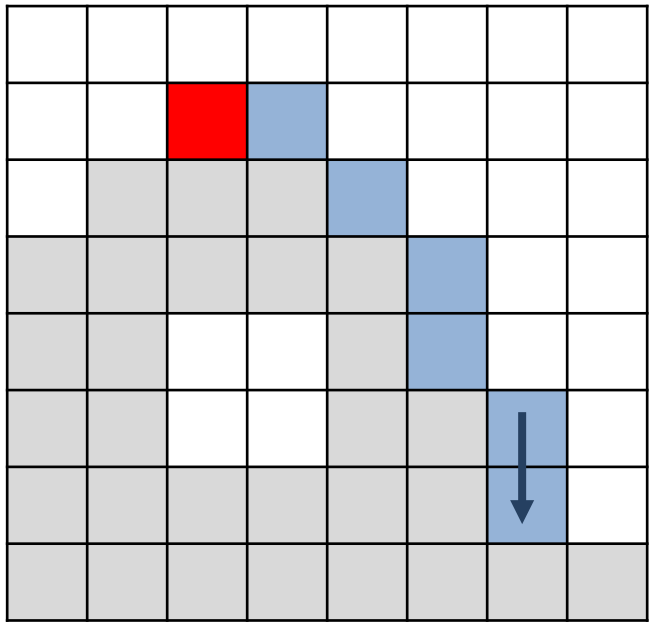
- What I want is something like:
- (1, 0) ✓
- (1, 1) ✓
- (1, 1) ✓
- (0, 1) ✓
- (1, 1) ✓
- (0, 1)
- (1, 1)
- etc…

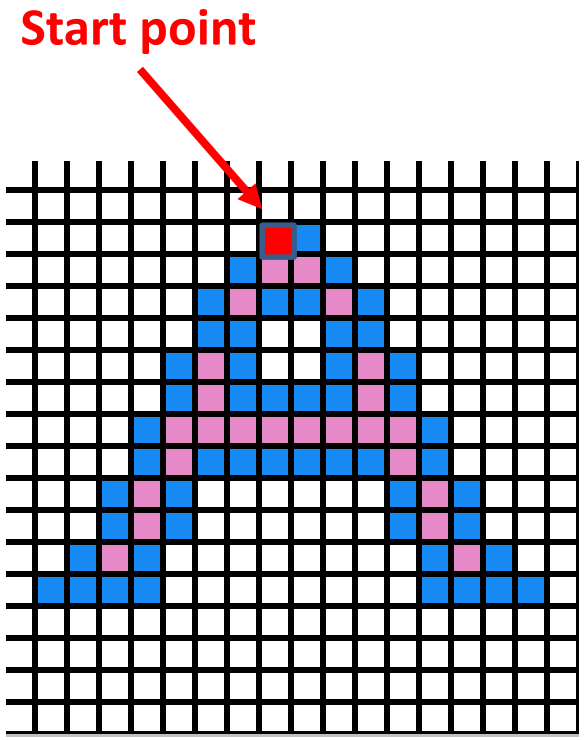




It's important to realize that the contour is an ordered sequence – we don't just want an image of the boundary pixels, but the sequence of moving between them

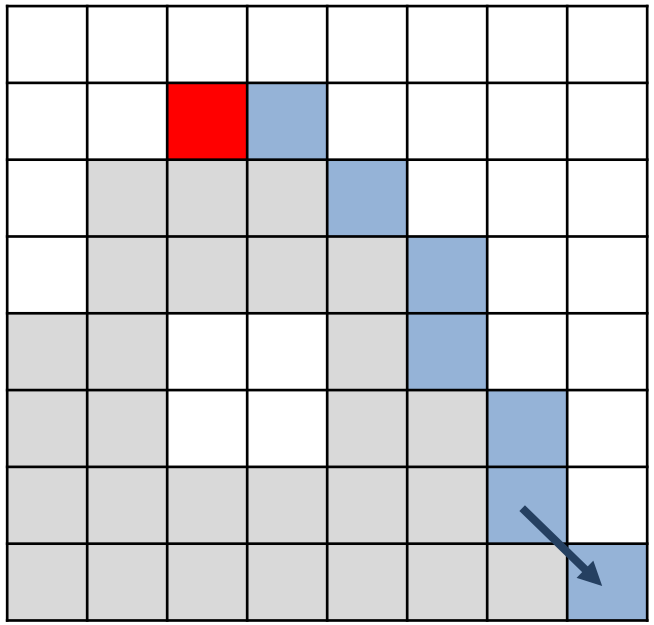
- What I want is something like:
- (1, 0) ✓
- (1, 1) ✓
- (1, 1) ✓
- (0, 1) ✓
- (1, 1) ✓
- (0, 1) ✓
- (1, 1)
- etc…

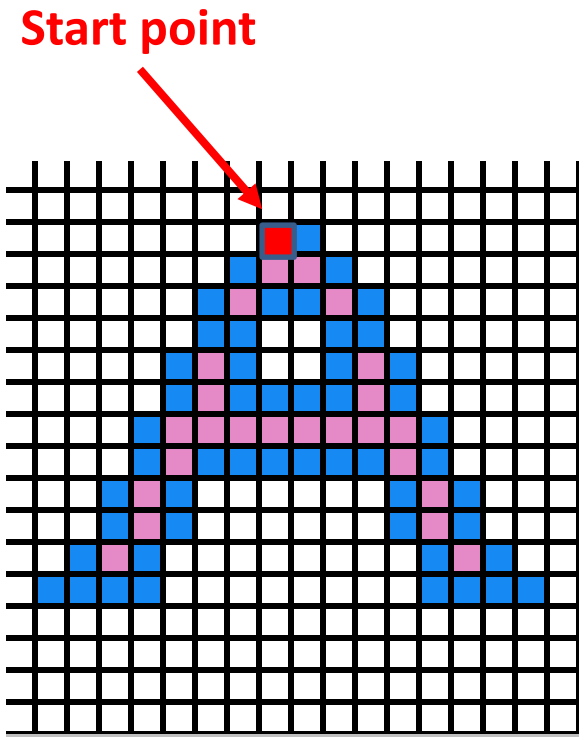




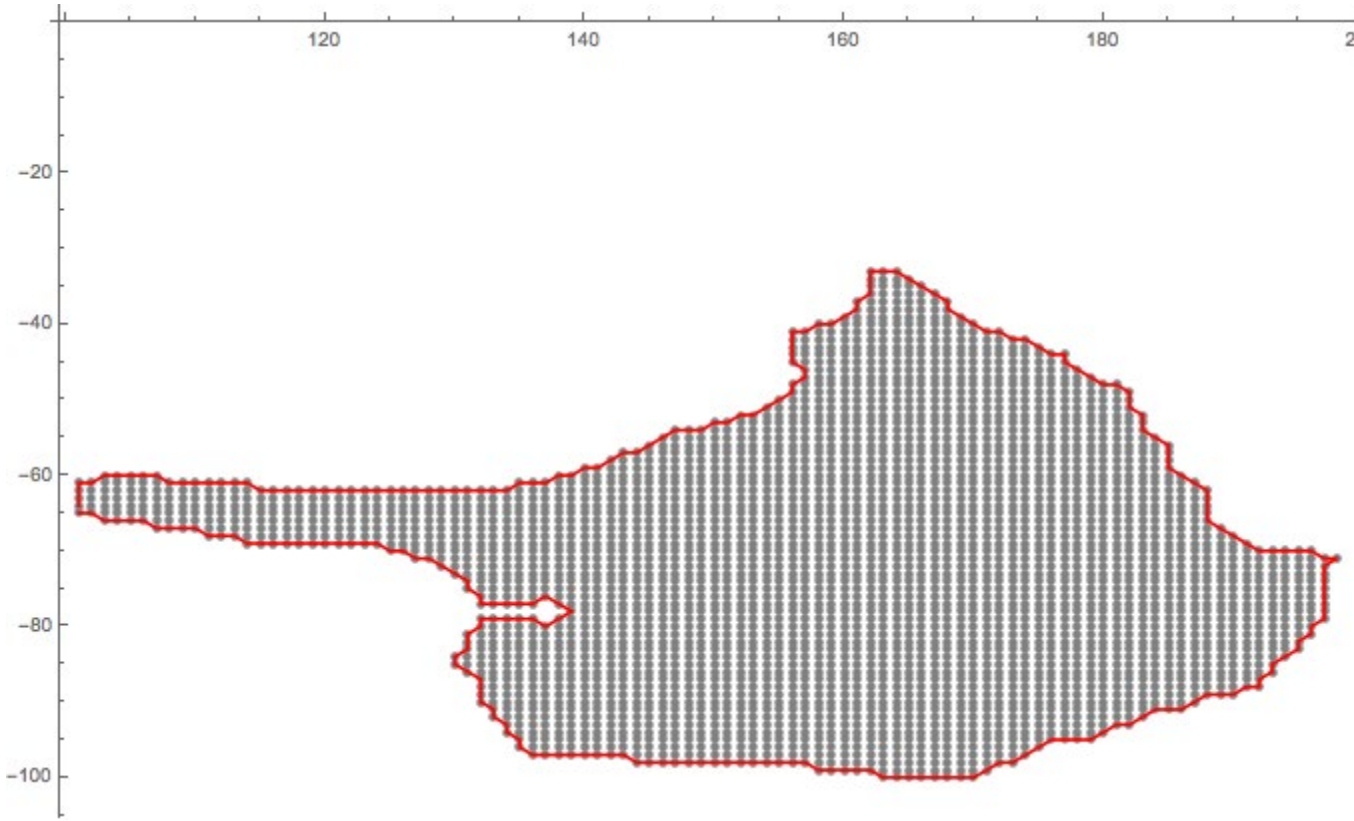
It's important to realize that the contour is an ordered sequence – we don't just want an image of the boundary pixels, but the sequence of moving between them

- What I want is something like:
- (1, 0) ✓
- (1, 1) ✓
- (1, 1) ✓
- (0, 1) ✓
- (1, 1) ✓
- (0, 1) ✓
- (1, 1) ✓
- etc…

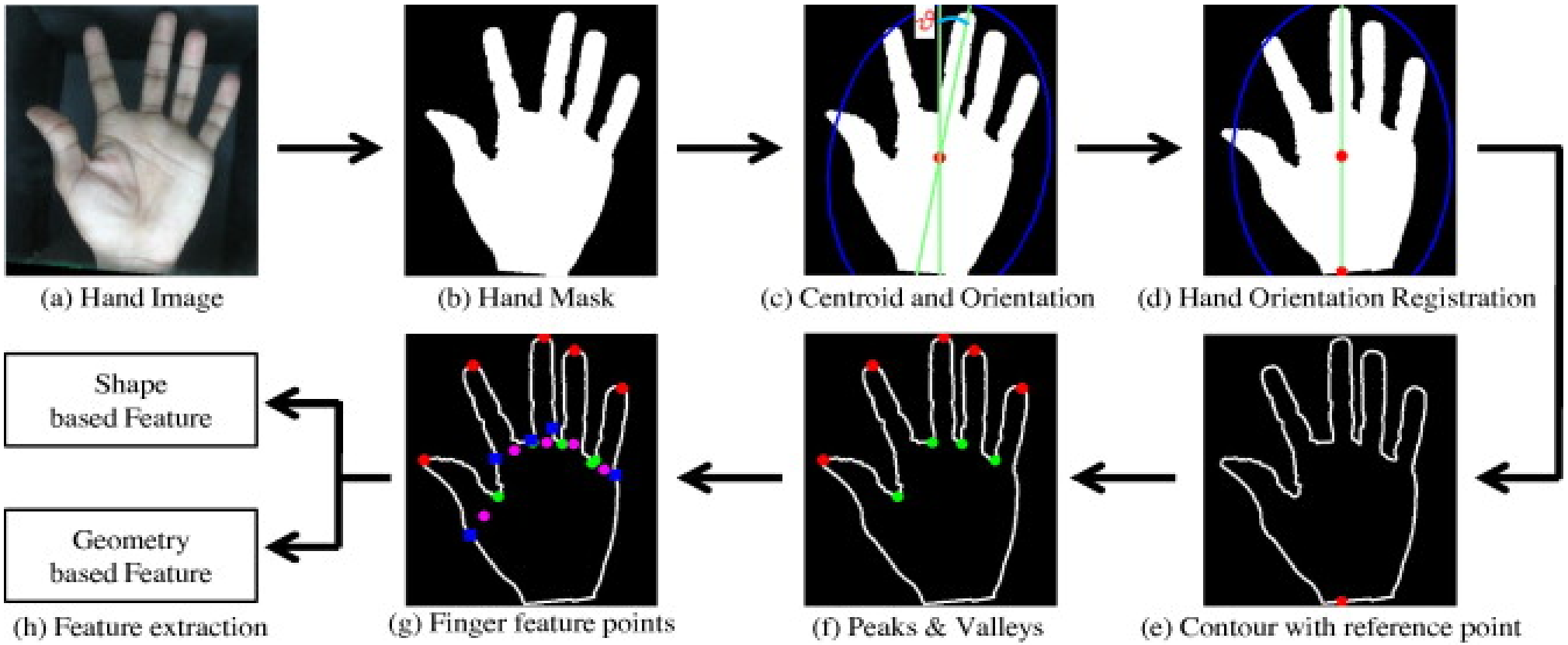




Contour information can be accumulated while the connected components are being identified – contour points have at least one 4-neighbor that's background – but the contour pieces may need to be assembled

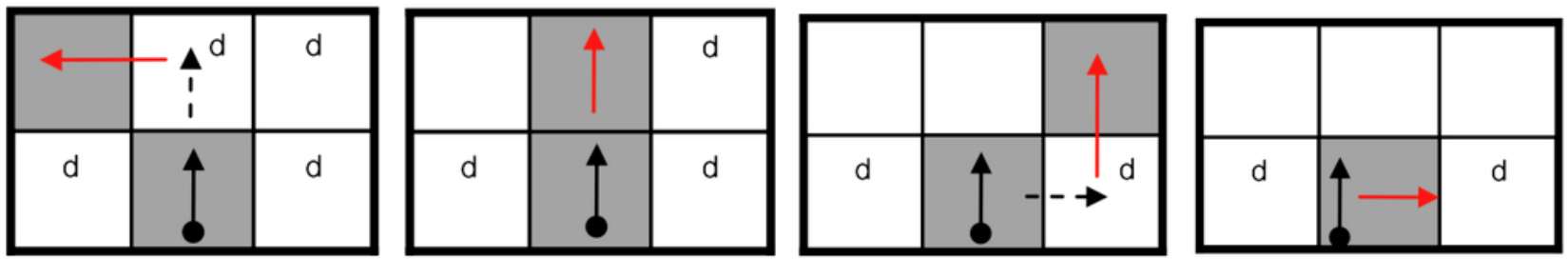


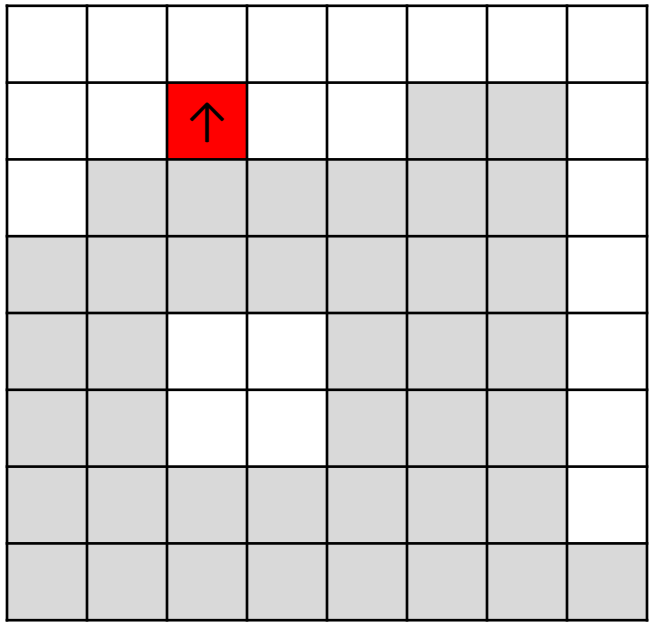
The *hand geometry* biometric modality is based on key points extracted from a contour of the hand, placed on an imaging surface



Let's look at the Theo Pavlidis algorithm – it does pretty well on contours of all sorts

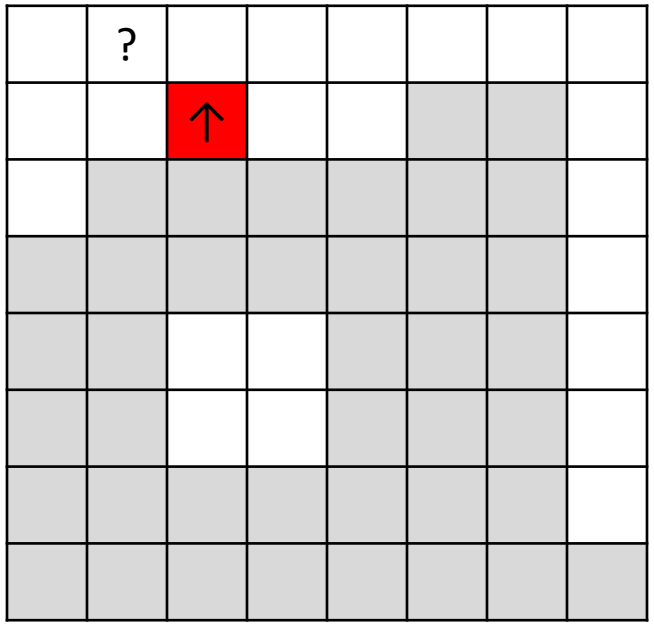
- We consider a current direction for the contour, based on the last movement that was made
- Look at the three neighboring pixels in the current direction
  - front-left, front and front-right pixels
  - consider the three pixels in this order
- If any of the three is foreground, then move in that direction
  - if we choose the front-left pixel, turn the direction 90º to the left
- If none are foreground, then turn the direction 90º to the right and repeat (examine the new front three pixels)





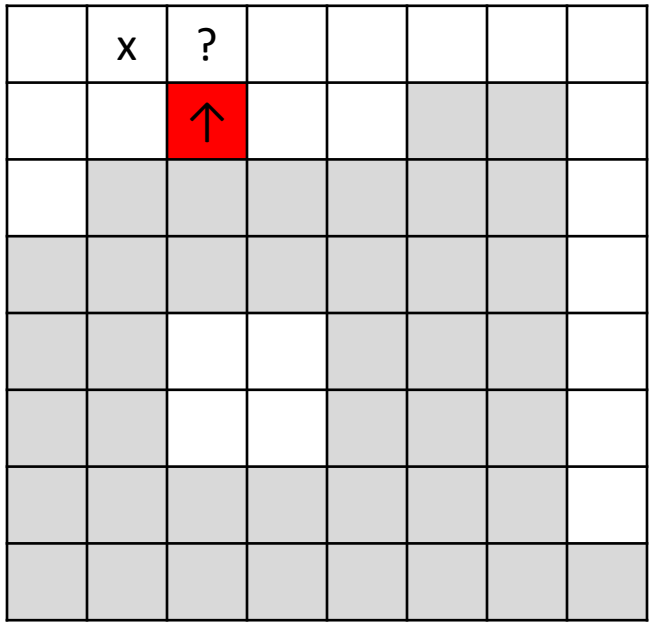
Absolute encoding: (2, 1) …

Differential encoding: (2, 1) …



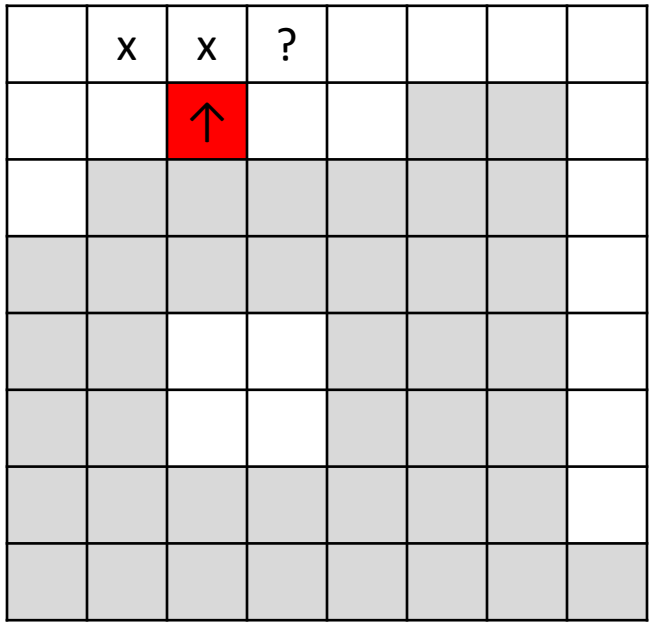
Absolute encoding: (2, 1) …

Differential encoding: (2, 1) …



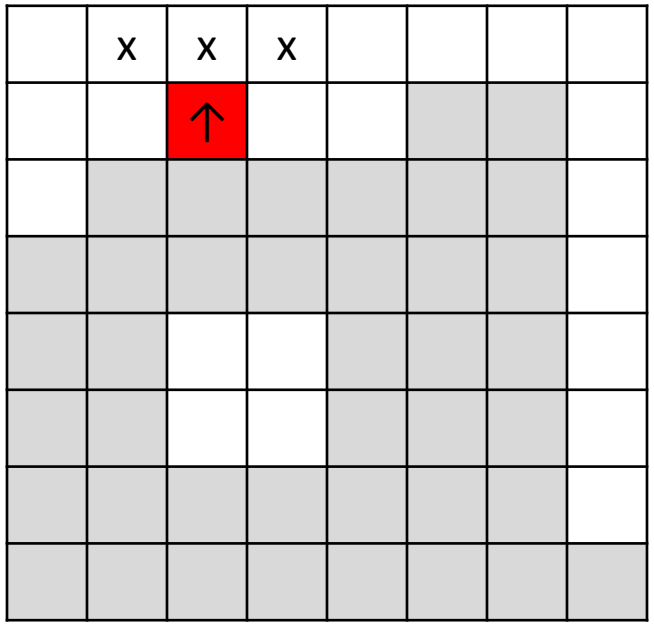
Absolute encoding: (2, 1) …

Differential encoding: (2, 1) …



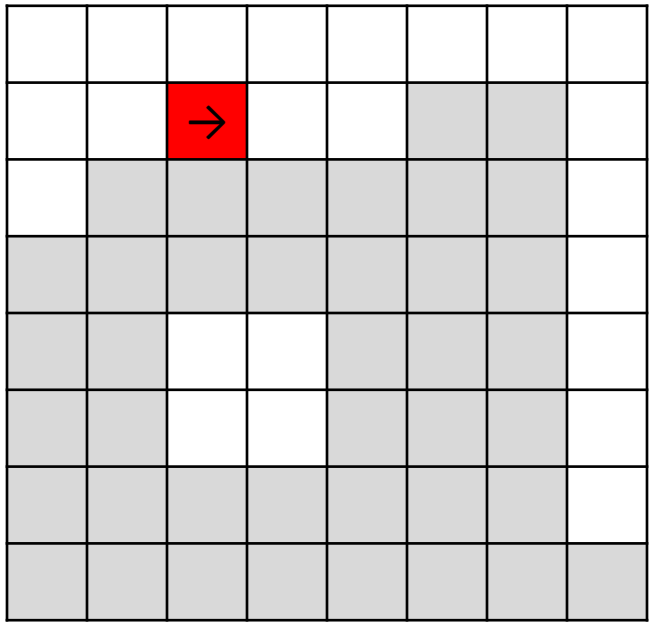
Absolute encoding: (2, 1) …

Differential encoding: (2, 1) …



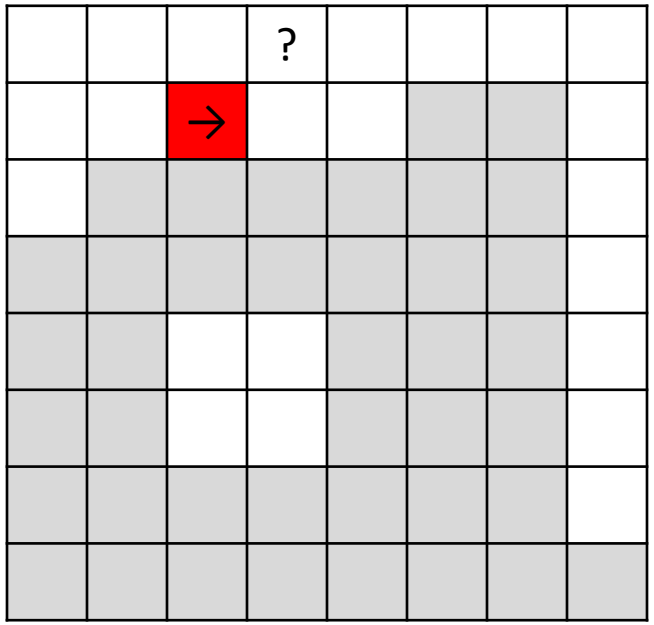
Absolute encoding: (2, 1) …

Differential encoding: (2, 1) …



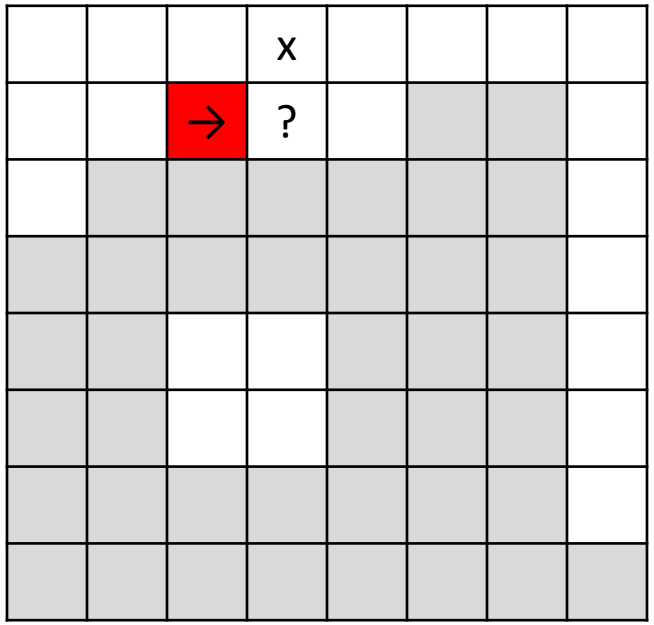
Absolute encoding: (2, 1) …

Differential encoding: (2, 1) …



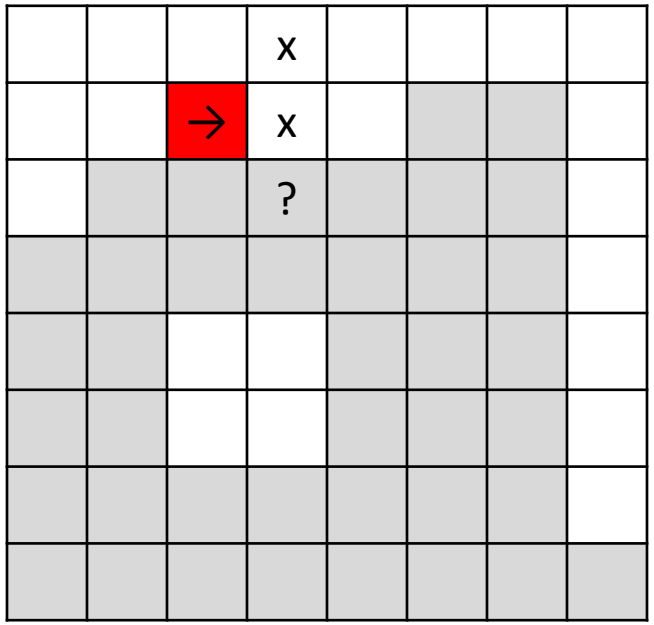
Absolute encoding: (2, 1) …

Differential encoding: (2, 1) …



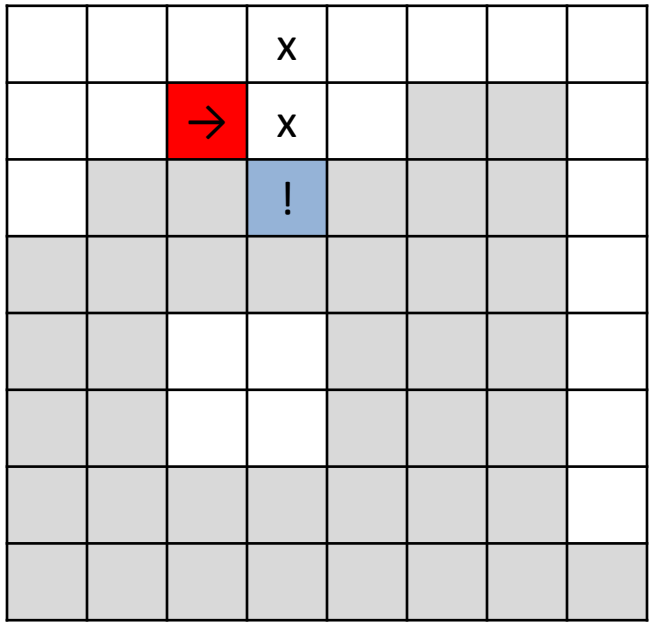
Absolute encoding: (2, 1) …

Differential encoding: (2, 1) …



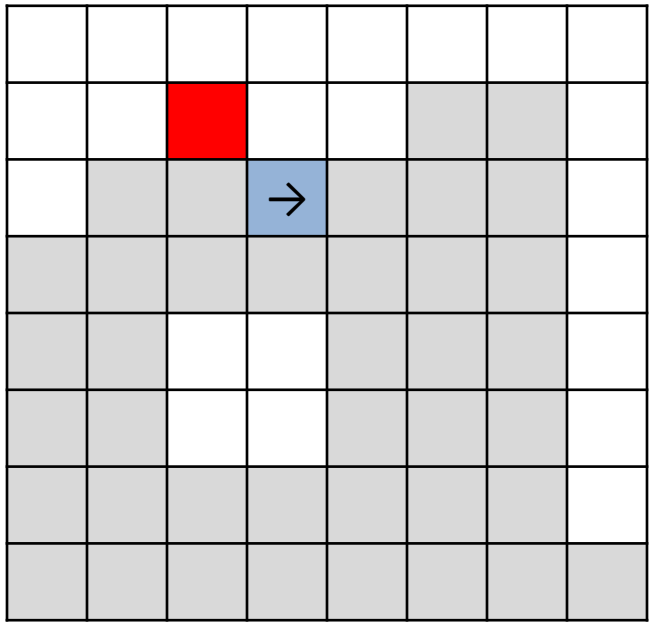
Absolute encoding: (2, 1) …

Differential encoding: (2, 1) …



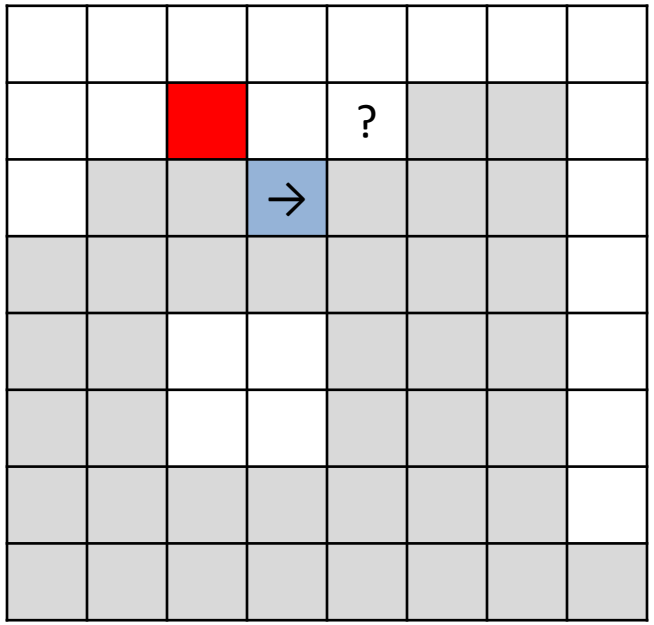
Absolute encoding: (2, 1) …

Differential encoding: (2, 1) …



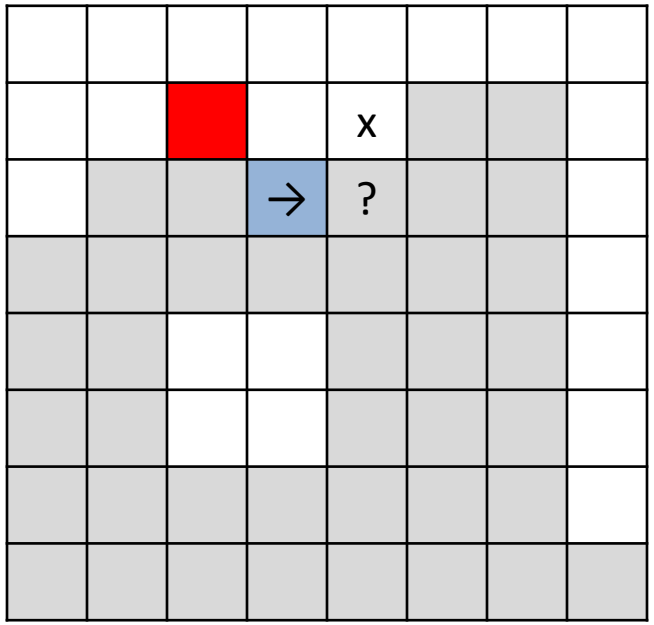
Absolute encoding: (2, 1)**, (3, 2)**…

Differential encoding: (2, 1)***, [1, 1]***…



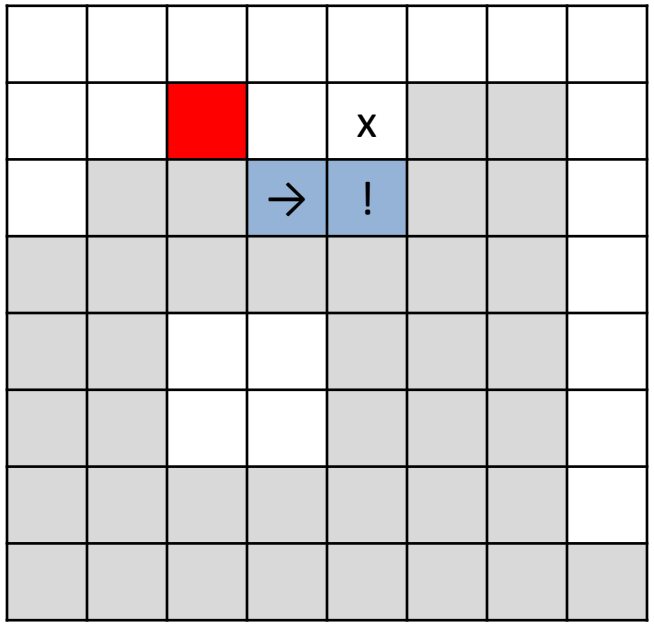
Absolute encoding: (2, 1), (3, 2)…

Differential encoding: (2, 1), *[1, 1]*…



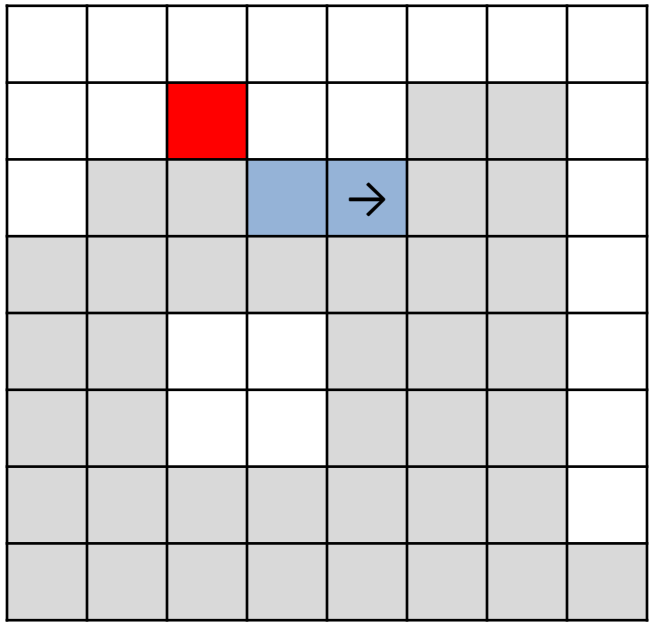
Absolute encoding: (2, 1), (3, 2)…

Differential encoding: (2, 1), *[1, 1]*…



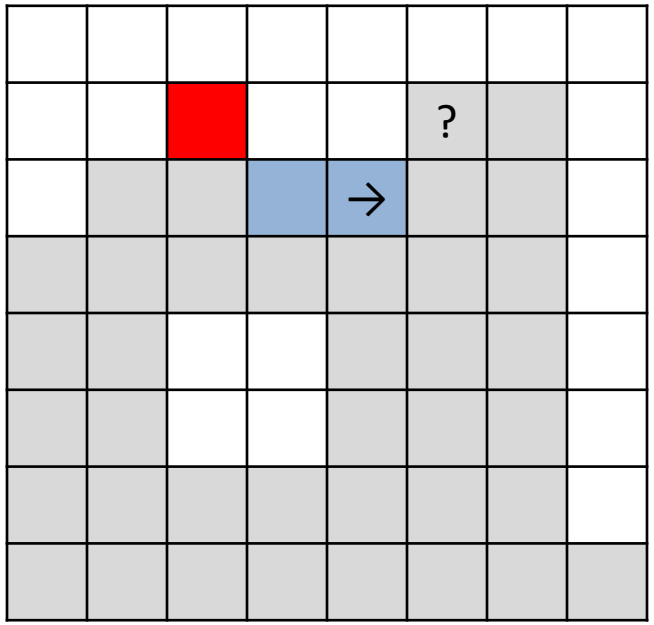
Absolute encoding: (2, 1), (3, 2)…

Differential encoding: (2, 1), *[1, 1]*…



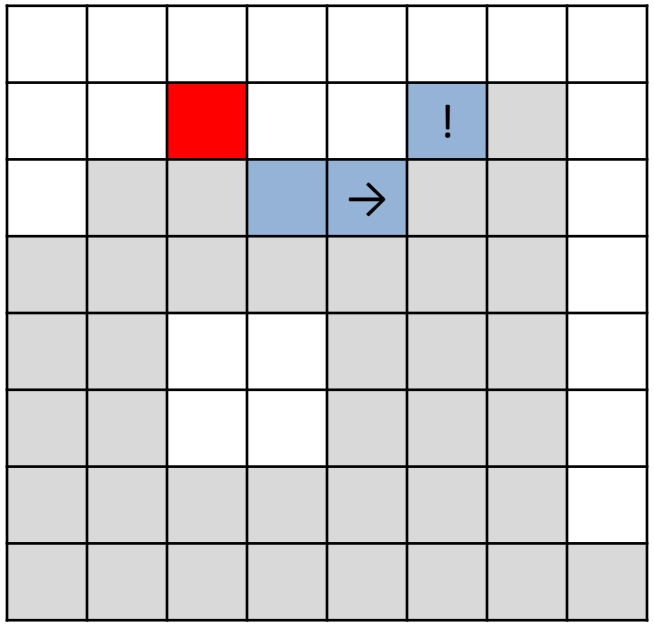
Absolute encoding: (2, 1), (3, 2)**, (4, 2)**…

Differential encoding: (2, 1), *[1, 1]****, [1, 0]***…



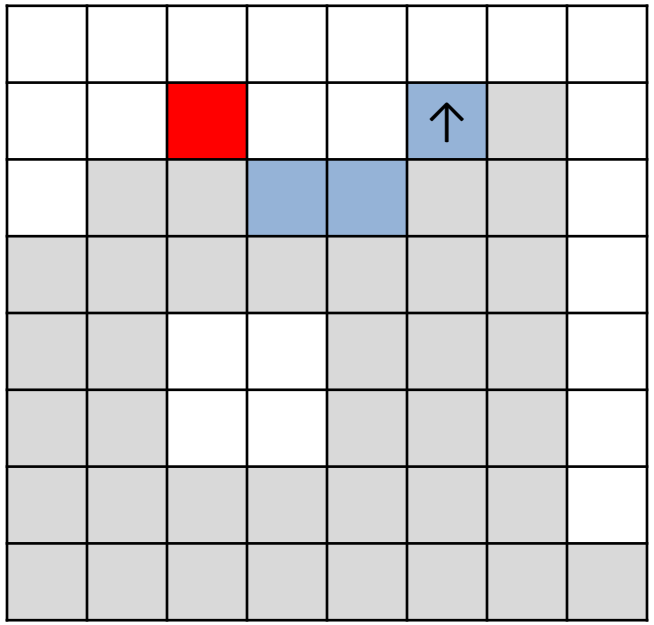
Absolute encoding: (2, 1), (3, 2), (4, 2)…

Differential encoding: (2, 1), *[1, 1], [1, 0]*…



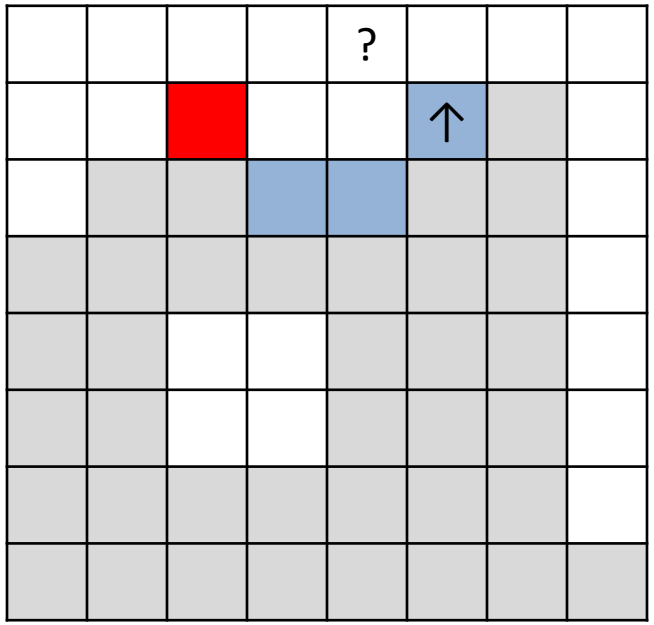
Absolute encoding: (2, 1), (3, 2), (4, 2)…

Differential encoding: (2, 1), *[1, 1], [1, 0]*…



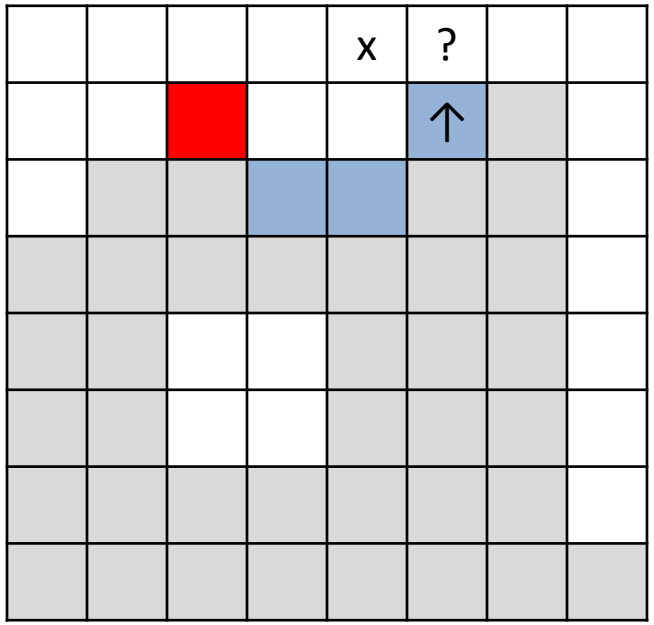
Absolute encoding: (2, 1), (3, 2), (4, 2)**, (5, 1)**…

Differential encoding: (2, 1), *[1, 1], [1, 0]****, [1, -1]*** …



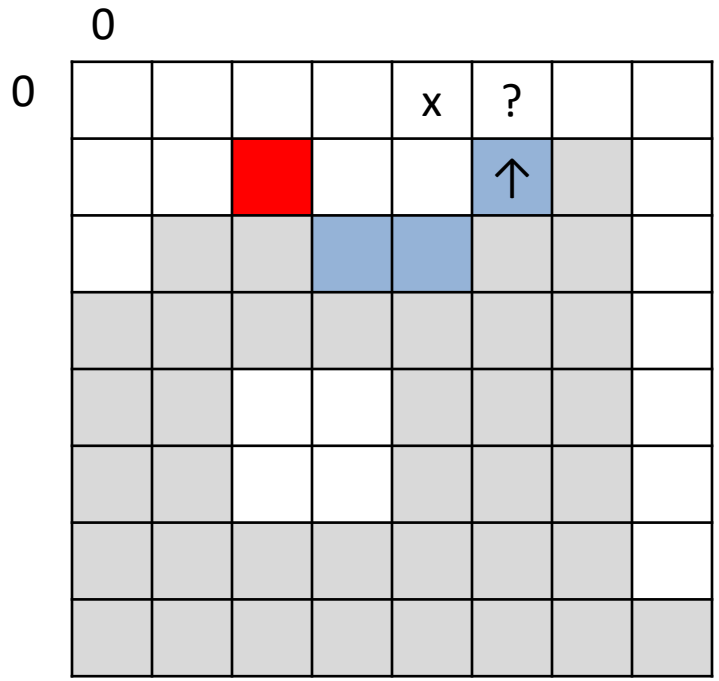
Absolute encoding: (2, 1), (3, 2), (4, 2), (5, 1)…

Differential encoding: (2, 1), *[1, 1], [1, 0], [1, -1]* …



Absolute encoding: (2, 1), (3, 2), (4, 2), (5, 1)…

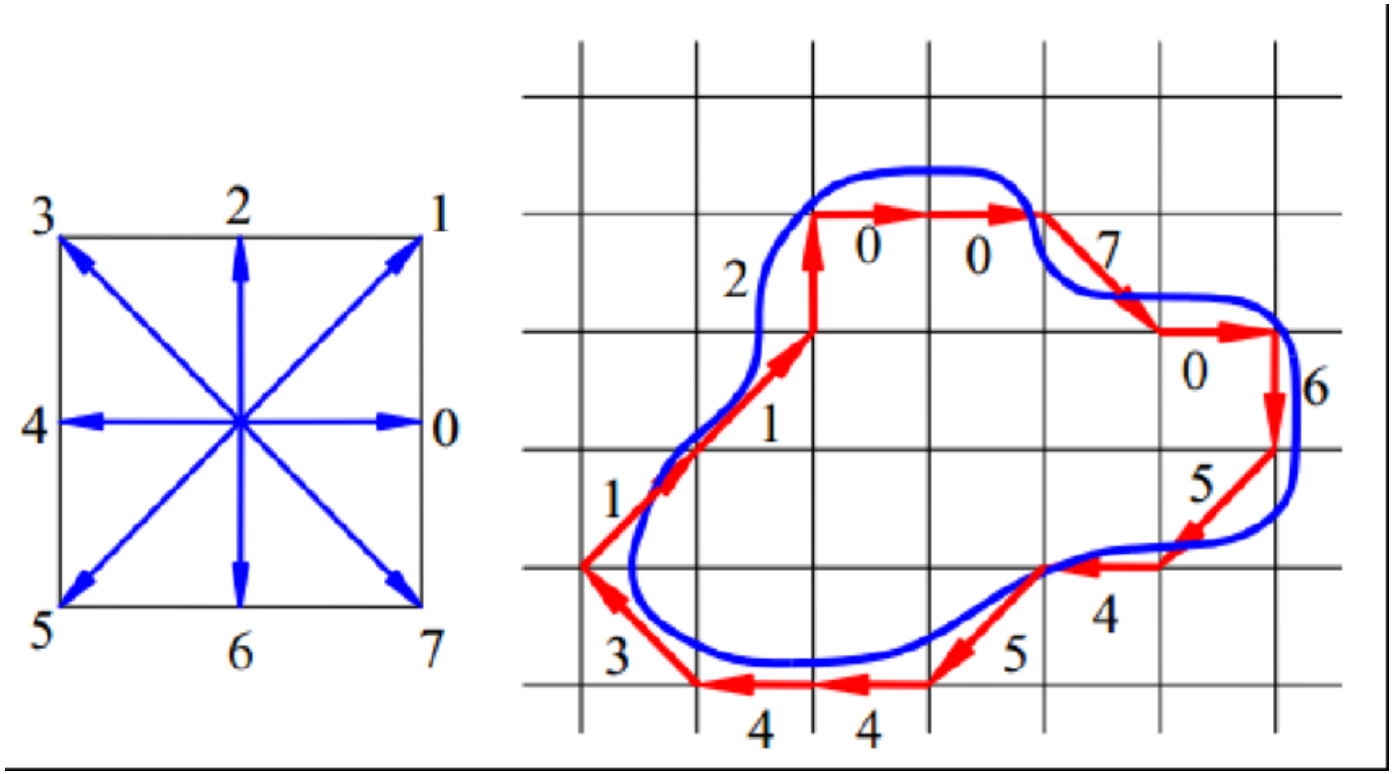
Differential encoding: (2, 1), *[1, 1], [1, 0], [1, -1]* …



Absolute encoding: (2, 1), (3, 2), (4, 2), (5, 1)…

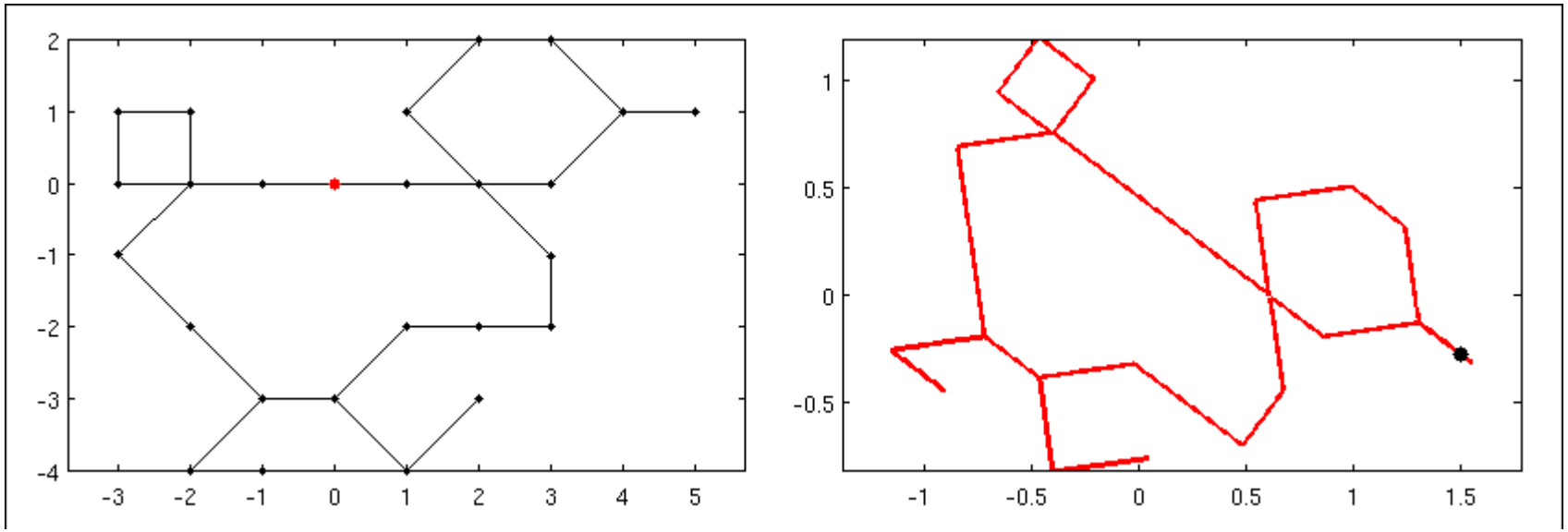
Differential encoding: (2, 1), *[1, 1], [1, 0], [1, -1]* …

The directions of change in movement between contour points is often encoded by the 8 orthogonal and diagonal directions – this results in the *Freeman chain code*



https://www.researchgate.net/publication/305791915_Patient_Condition_Monitoring_Modular_Hospital_Robot

The Freeman chain code is only mildly affected by rotation of the object – a constant is added (modulo 8) to each entry



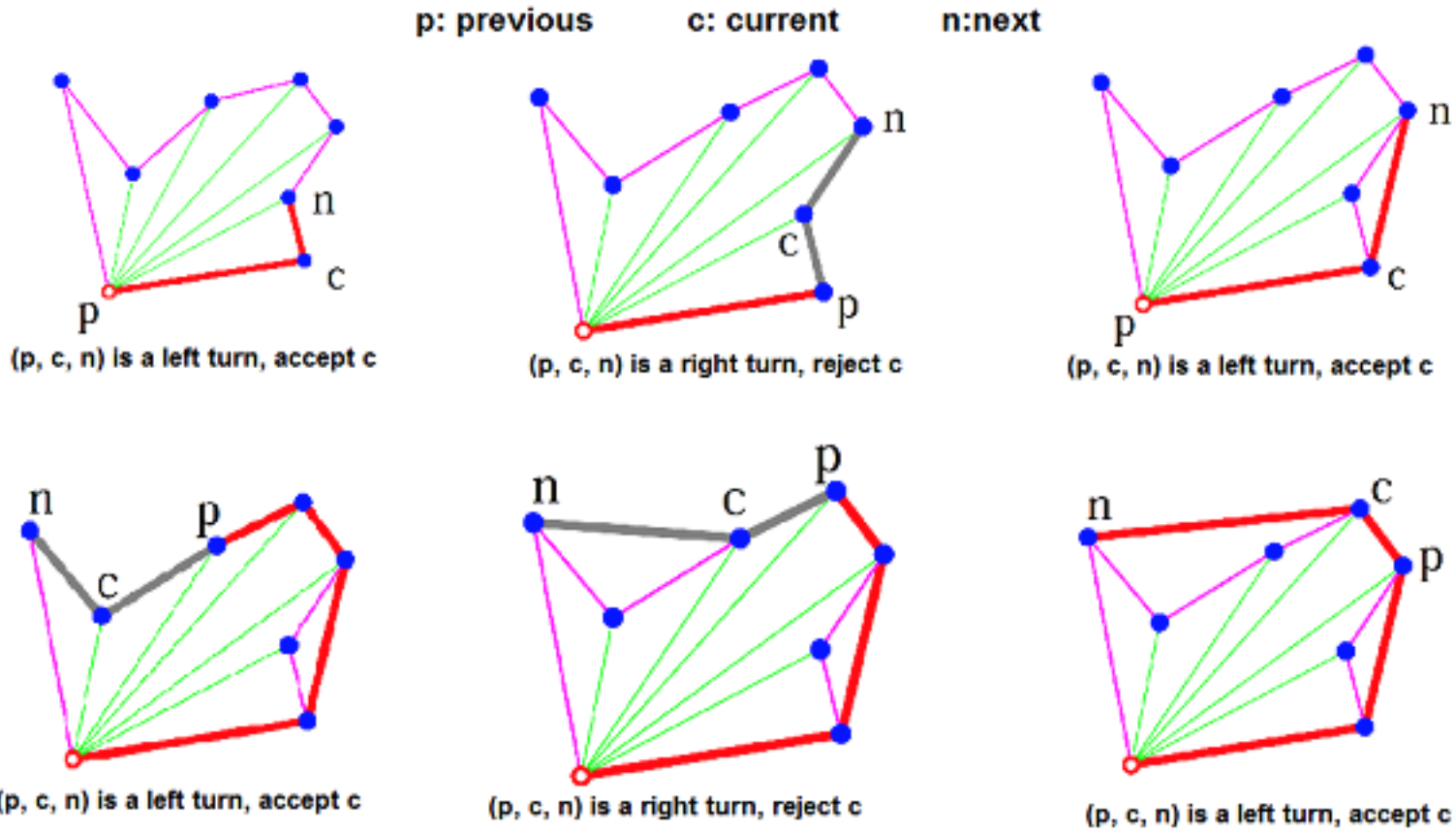
- The chain code of the figure on the left, starting on the beak, is

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

- For the figure on the right, it's

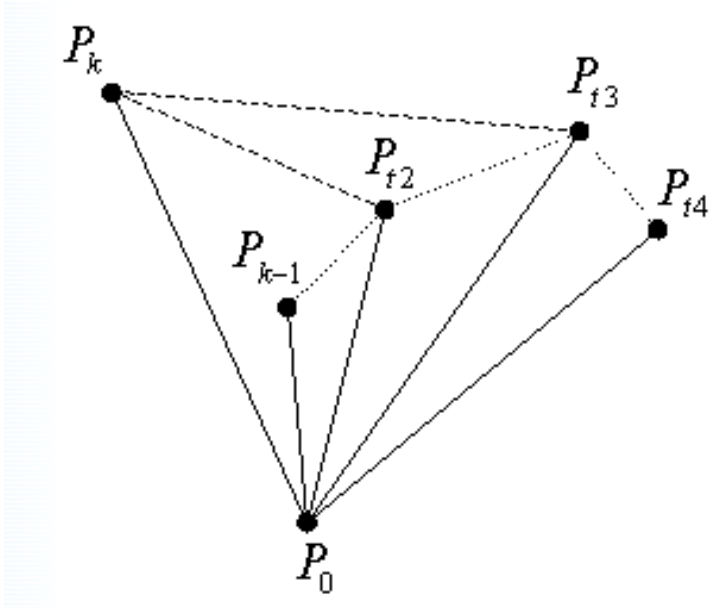
  5 4 5 2 3 5 5 4 2 0 4 6 5 4 0 6 6 0 5 7 1 3 1 1 1 1 6 0 1 2 1

A useful subset of the contour is the *convex hull* – points form the smallest convex set enclosing the contour (and the object)



In the above algorithm and below code, a stack of points is used to store convex hull points. With reference to the code, p is next-to-top in stack, c is top of stack and n is points[i].

One of the classic algorithms for extracting the convex hull is the Graham scan; it uses a stack to store points that may end up being on the hull



Old Stack $= S_{k-1} = \{P_0, \ldots, P_{t3}, P_{t2}, P_{k-1}\}$

P_k right of line $P_{t2}P_{k-1} \Rightarrow$ pop P_{k-1} off stack

P_k right of line $P_{t3}P_{t2} \Rightarrow$ pop P_{t2} off stack

P_k left of line $P_{t4}P_{t3} \Rightarrow$ push P_k onto stack

New Stack $= S_k = \{P_0, \ldots, P_{t3}, P_k\}$

See http://www.algomation.com/algorithm/graham-scan-convex-hull

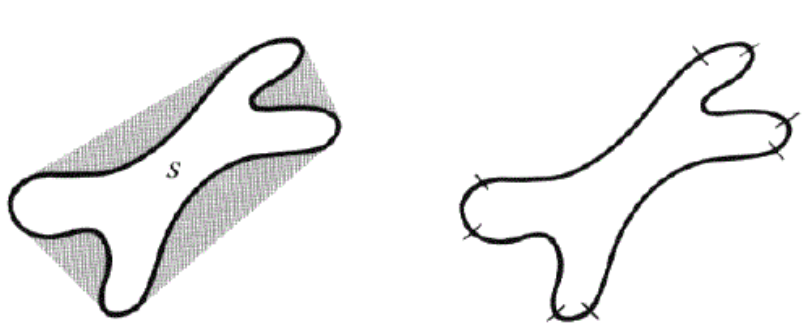
Here is one version of the Graham scan algorithm for finding the convex hull of a set of points in $\mathbb{R}^2$

1. Find the bottom-most point by comparing y coordinate of all points. If there are two points with same y value, then the point with smaller x coordinate value is considered. Let the bottom-most point be P_0. Put P_0 at first position in output hull.
2. Consider the remaining n-1 points and sort them by polar angle in counterclockwise order around points[0]. If polar angle of two points is same, then put the nearest point first.
3. After sorting, check if two or more points have same angle. If two more points have same angle, then remove all same angle points except the point farthest from P_0. Let the size of new array be m.
4. If m is less than 3, return (Convex Hull not possible)
5. Create an empty stack 'S' and push points[0], points[1] and points[2] to S.
6. Process remaining m-3 points one by one. Do following for every point 'points[i]'
    1. Keep removing points from stack while orientation of following 3 points is not counterclockwise (or they don't make a left turn).
        1. Point next to top in stack
        2. Point at the top of stack
        3. points[i]
    2. Push points[i] to S
7. S contains the points of the convex hull

https://www.geeksforgeeks.org/convex-hull-set-2-graham-scan/

What uses are there for the contour and/or convex hull of an object?

- Identifying shape
  - differences between the contour and the convex hull are called convex deficiencies or bays
- Rotationally invariant descriptions
- Compression
  - only store the outline of objects
- Sizes can be extracted
  - perimeter, bounding box, etc…



Today's Objectives

Contours in Binary Images

- Contour tracing
  - the Theo Pavlidis algorithm
- Chain code
- Convex Hull