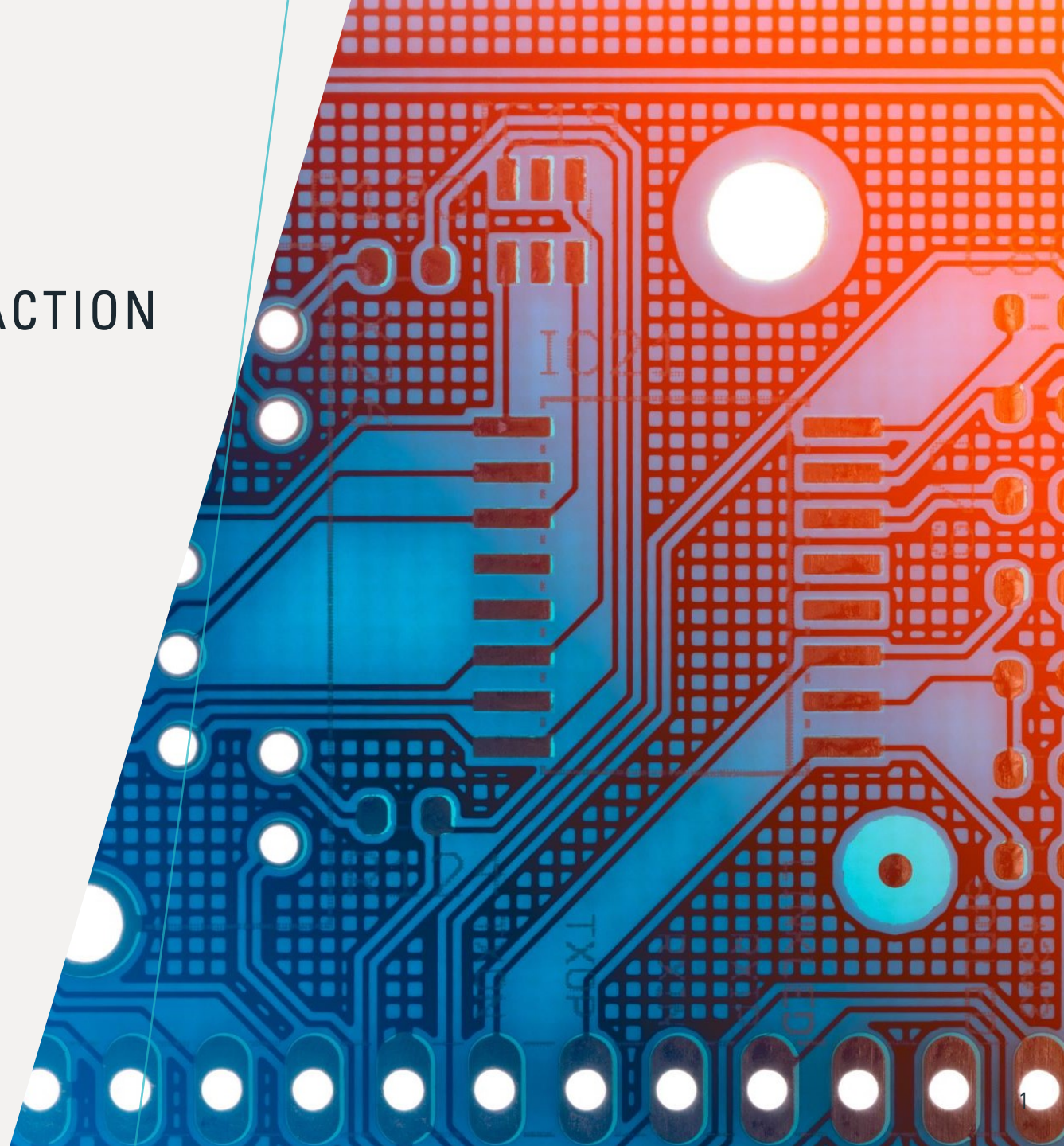
CHAPTER 11

ARC SEGMENTATION AND EXTRACTION



PRESENTED BY: RYANDHIMAS E. ZEZARIO

RYANDHIMAS@CITI.SINICA.EDU.TW

授課教授：傅楸善 博士

國立臺灣大學 National Taiwan University

DC&CV LAB
CSIE NTU 2020

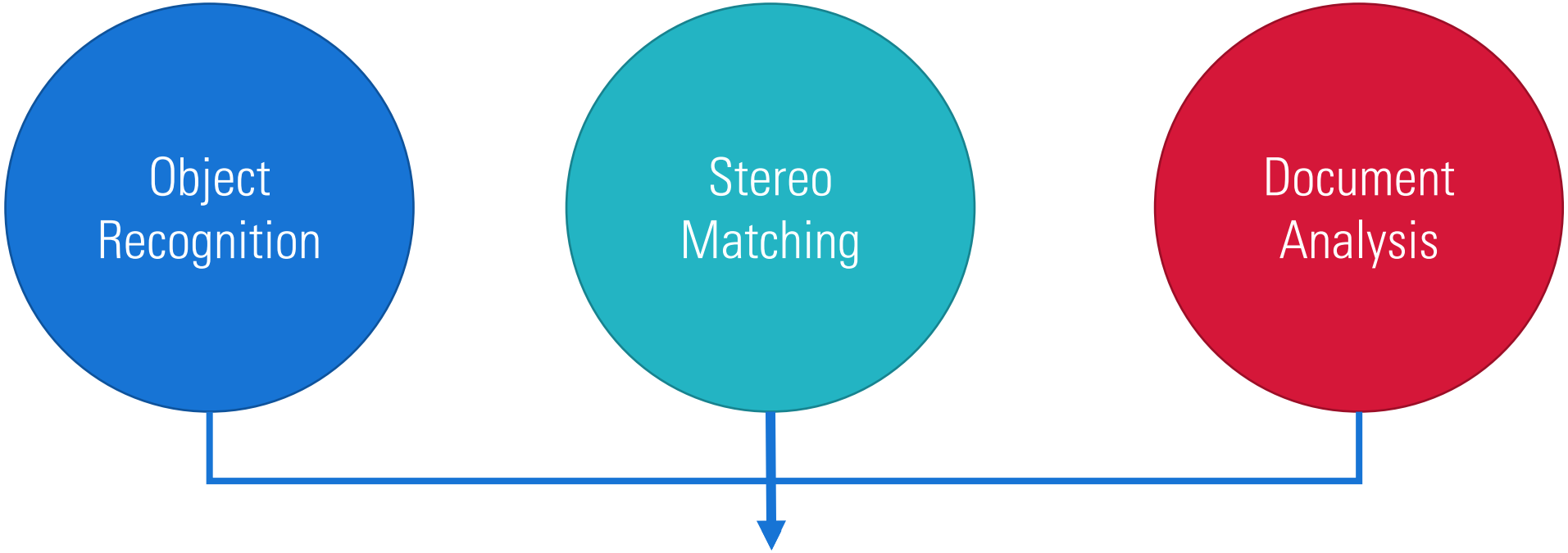
OUTLINES

- 11.1 Introduction
- 11.2 Extracting Boundary Pixels from a Segmented Image
- 11.3 Linking One-Pixels-Wide Edges or Lines
- 11.4 Edge and Line Linking Using Directional Information
- 11.5 Segmentation of Arcs into Simple Segments

OUTLINES

- 11.6 Hough Transform
- 11.7 Line Fitting
- 11.8 Region-of-Support Determination
- 11.9 Robust Line Fitting
- 11.10 Least-Squares Curve Fitting

INTRODUCTIONS



In some image sets, **lines, curves, and circular arcs are more useful** than regions

11.1 INTRODUCTIONS

- In this Chapter, we will discuss techniques for **extracting sequences of pixels** that belong to the same curve

  - Source: Segmented or Labeled Images

11.1 INTRODUCTIONS

- Grouping Operations:

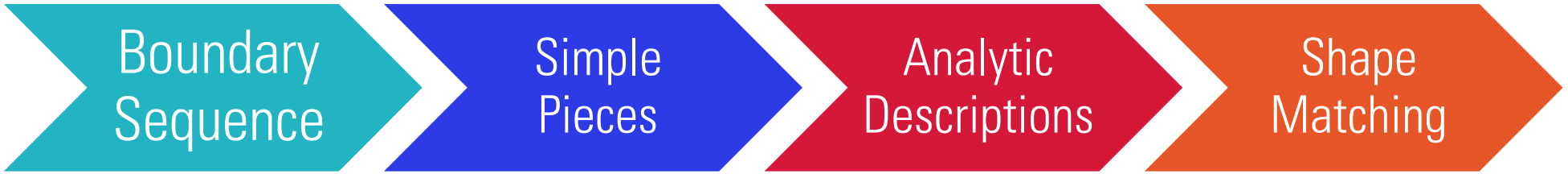
  - It aims to **extract sets or sequences** of labeled or border pixel positions.
  - It performs after the edge labeling or image segmentation.

11.1 INTRODUCTIONS

- Labeling
  - ➢ Check whether it is edge or non-edge?
  - ➢ Additional Properties: edge direction, gradient magnitude, edge contrast

11.1 INTRODUCTIONS

- Grouping
  - ➢Edge pixels participating in **the same region boundary** are grouped together into a sequence.



11.2 EXTRACTING BOUNDARY PIXELS FROM A SEGMENTED IMAGE

- Once the regions have been **determined by segmentation or connected components**
- Therefore, boundary of each region can be extracted

- Boundary extraction for small-sized images
  - **Scan through the image** and make a list of the first border pixel for each connected component
  - In each region, **begin at first border of each region**, follow the border of the connected component around in a **clockwise direction until reach itself**

11.2 EXTRACTING BOUNDARY PIXELS FROM A SEGMENTED IMAGE

- Boundary extraction for large-sized images
  - **Memory problems**; for large-sized one, simple boarder-tacking algo. may result in excessive I/O to storage
  - 4K UHD – (3840 × 2160) ~= 8.3M pixel
  - Many techniques like **down-sampling**
  - It may be a problem back to 1992; how about TODAY?

11.2.1 BORDER TRACKING ALGORITHM

- Border-tracking algorithm: *border*

  - Input: symbolic image
  - Output: a **clockwise-ordered list** of the coordinates of its border pixels
  - In one left-right, top-bottom scan through the image
  - During execution, there are 3 sets of regions: current, past, future

11.2.1 BORDER TRACKING ALGORITHM

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

(a) A symbolic image with two regions.

Past region: null
Current region: 2
Future region: 1

11.2.1 BORDER TRACKING ALGORITHM

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

(a) A symbolic image with two regions.

Past region: 1
Current region: 2
Future region: null

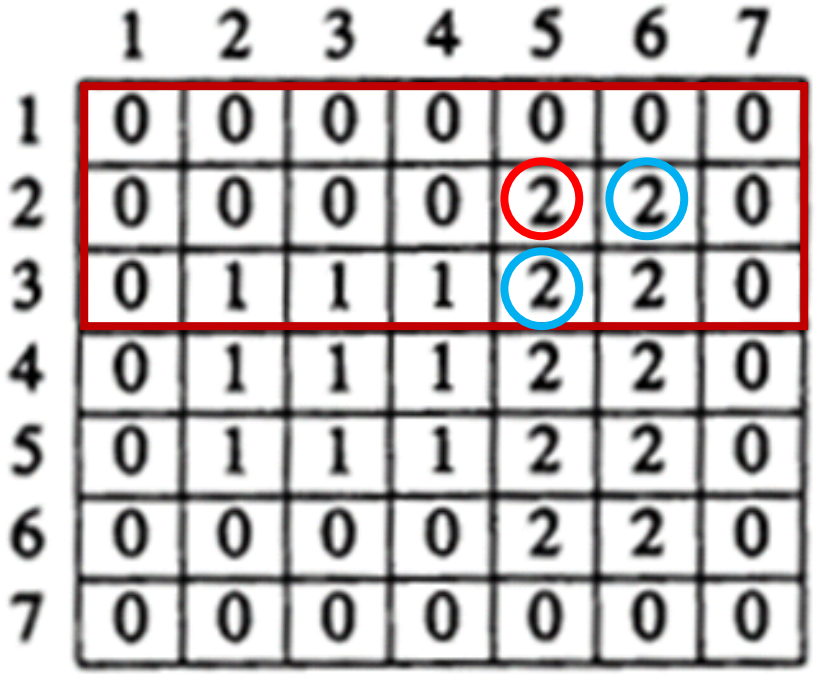
```
procedure border;
  for R:= 1 to NLINES do
    begin
      for C:= 1 to NPIXELS do
        begin
          LABEL:= S(R,C);
          if new_region(LABEL) then add (CURRENT,LABEL);
          NEIGHB:=neighbors(R,C,LABEL);
          T:= pixeltype(R,C,NEIGHB);
          if T == 'border'
          then for each pixel N in NEIGHB do
            begin
              CHAINSET:=chainlist(LABEL);
              NEWCHAIN:=true;
              for each chain X in CHAINSET while NEWCHAIN do
                if N==rear(X)
                then begin add(X,(R,C)); NEWCHAIN:= false end
              end for
              if NEWCHAIN
              then make_new_chain(CHAINSET,(R,C),LABEL)
            end
          end for
        end
      end for;
      for each region REG in CURRENT
        if complete(REG)
        then begin connect_chains(REG); output(REG); free(REG) end
      end for
    end
  end for
end border;
```

11.2.1 BORDER TRACKING ALGORITHM

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

(a) A symbolic image with two regions.

11.2.1 BORDER TRACKING ALGORITHM



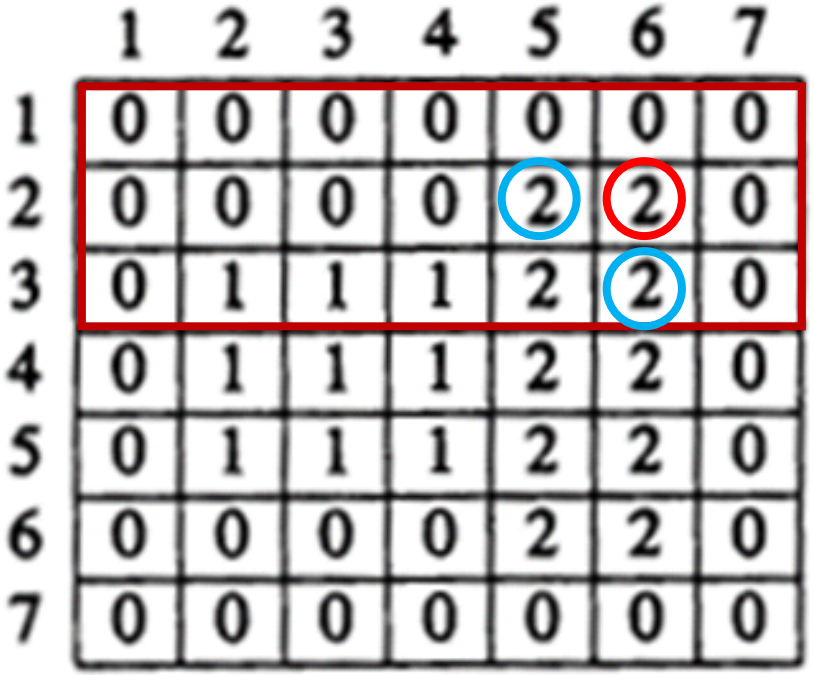
	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	0	2	2	0
3	0	1	1	1	2	2	0
4	0	1	1	1	2	2	0
5	0	1	1	1	2	2	0
6	0	0	0	0	2	2	0
7	0	0	0	0	0	0	0

CHAINSET : NULL

(2,5) NEIGHB (2,6),(3,5)

CHAINSET (2)→(2,5)

11.2.1 BORDER TRACKING ALGORITHM



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

(active)

CHAINSET (2)→(2,5)

(2,6) NEIGHB (2,5),(3,6)

CHAINSET (2)→(2,5)→(2,6)

11.2.1 BORDER TRACKING ALGORITHM

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

(Inactive) CHAINSET (2) -> (2,5) -> (2,6)

(active) CHAINSET (1) -> (3,2)
(3,2) NEIGHB (3,3), (4,2)

11.2.1 BORDER TRACKING ALGORITHM

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

(Inactive) CHAINSET (2) -> (2,5) -> (2,6)
(active) CHAINSET (1) -> (3,2)

- (3,3) NEIGHB (3,2),(3,4),(4,3)
- (active) CHAINSET (1) -> (3,2) -> (3,3)

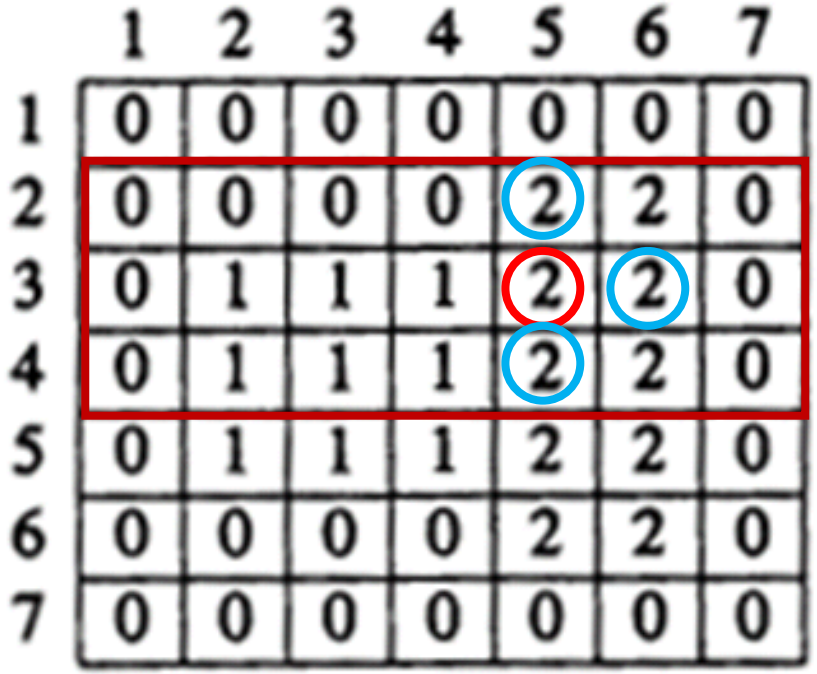
11.2.1 BORDER TRACKING ALGORITHM

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

(Inactive) CHAINSET (2) -> (2,5) -> (2, 6)
(active) CHAINSET (1) -> (3,2) -> (3,3)

- (3,4) NEIGHB (3,3),(4,4)
- (active) CHAINSET (1) -> (3,2) -> (3,3) -> (3,4)

11.2.1 BORDER TRACKING ALGORITHM



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

(Inactive) CHAINSET (2) -> (2,5) -> (2,6)
(Inactive) CHAINSET (1) -> (3,2) -> (3,3) -> (3,4)

(active) CHAINSET (2) -> (3,5)
(3,5) NEIGHB (2,5), (3,6), (4,5)

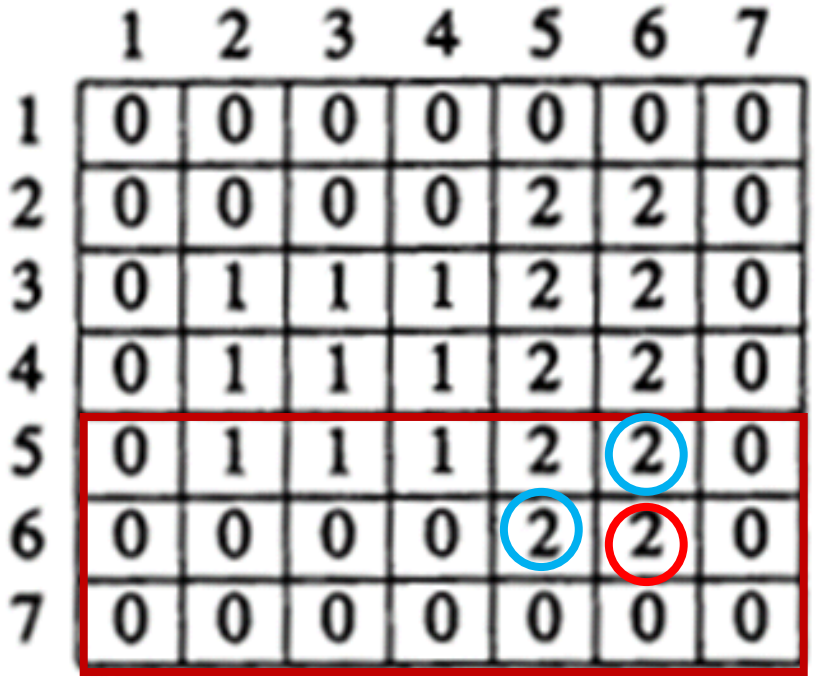
11.2.1 BORDER TRACKING ALGORITHM

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

(inactive) CHAINSET (2) -> (2,5) -> (2,6)
(inactive) CHAINSET (1) -> (3,2) -> (3,3) -> (3,4)
(active) CHAINSET (2) -> (3,5)

- (3,6) NEIGHB (2,6), (3,5), (4,6)
- active CHAINSET change! (clockwise-ordered)
- (active) CHAINSET (2) -> (2,5) -> (2,6) -> (3,6)

11.2.1 BORDER TRACKING ALGORITHM



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

CHAINSET (1)→(3,2)→(3,3)→(3,4)→(4,4)→(5,4)

(1)→(4,2)→(5,2)→(5,3)

(2)→(2,5)→(2,6)→(3,6)→(4,6)→(5,6)

(2)→(3,5)→(4,5)→(5,5)→(6,5)

(6,6) NEIGHB (5,6),(6,5)

CHAINSET (1)→(3,2)→(3,3)→(3,4)→(4,4)→(5,4)

(1)→(4,2)→(5,2)→(5,3)

(2)→(2,5)→(2,6)→(3,6)→(4,6)→(5,6)→(6,6)

(2)→(3,5)→(4,5)→(5,5)→(6,5)

11.2.1 BORDER TRACKING ALGORITHM

- CHAINSET

  (1)→(3,2)→(3,3)→(3,4)→(4,4)→(5,4)

  (1)→(4,2)→(5,2)→(5,3)

  (2)→(2,5)→(2,6)→(3,6)→(4,6)→(5,6)→(6,6)

  (2)→(3,5)→(4,5)→(5,5)→(6,5)

- CHAINSET

  (1)→(3,2)→(3,3)→(3,4)→(4,4)→(5,4)→(5,3)→(5,2)→(4,2)

  (2)→(2,5)→(2,6)→(3,6)→(4,6)→(5,6)→(6,6)→(6,5)→(5,5)→(4,5)→(3,5)

11.2.1 BORDER TRACKING ALGORITHM

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

(a) A symbolic image with two regions.

Region	Length	List
1	8	(3,2)(3,3)(3,4)(4,4)(5,4)(5,3)(5,2)(4,2)
2	10	(2,5)(2,6)(3,6)(4,6)(5,6)(6,6)(6,5)(5,5) (4,5)(3,5)

(b). The output of the border procedure for the symbolic image.

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

- Border tracking: it assumed that each border **bounded a closed region** → Therefore, NO point would be split into two or more segments.
- Tracking symbolic edge (line) segments: more complex

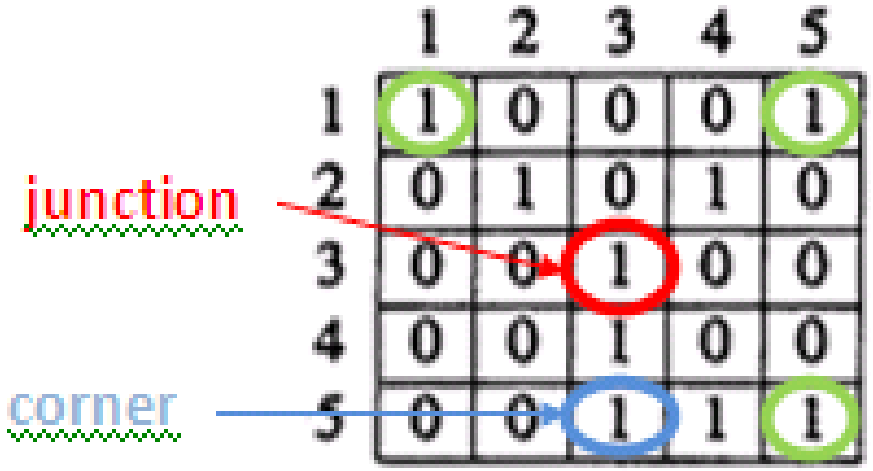
  → value:1 edge; 0: non-edge

  → not necessary for edge pixels to bound closed regions

  → segments consist of connected edge pixels that go from endpoint, corner, or junction to endpoint, corner, or junction.

  → pixeltype() is more complex

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES



	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

***INLIST, OUTLIST**

Figure 11.2 Symbolic edge image containing a junction of three line segments at pixel (3,3) and a potential corner at pixel (5,3).

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

- ***pixeltype()*** → *determines a pixel point*
  - *isolated point*
  - *starting point of an new segment*
  - *interior pixel of an old segment*
  - *ending point of an old segment*
  - *junction*
  - *corner*
- Instead of past, current, future regions, there are past, current, future **segments**.

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(1,1)

pixeltype() → start point of new segment

ID

(1)→(1,1)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(1,5)

pixeltype() → start point of new segment

ID

(1)→(1,1)

(2)→(1,5)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(2,2)

pixeltype() → interior point of old segment (1)

ID

(1)→(1,1)→(2,2)

(2)→(1,5)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(2,4)

pixeltype() → interior point of old segment (2)

ID

(1)→(1,1)→(2,2)

(2)→(1,5)→(2,4)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(3,3)

pixeltype() → junction (1)(2)

ID

(1)→(1,1)→(2,2)→(3,3) #end

(2)→(1,5)→(2,4)→(3,3) #end

(3)→(3,3)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(4,3)

pixeltype() → interior point of old segment (3)

ID

(1)→(1,1)→(2,2)→(3,3) #end

(2)→(1,5)→(2,4)→(3,3) #end

(3)→(3,3)→(4,3)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(5,3)

pixeltype() → corner (3)

ID

(1)→(1,1)→(2,2)→(3,3) #end

(2)→(1,5)→(2,4)→(3,3) #end

(3)→(3,3)→(4,3)→(5,3) #end

(4)→(5,3)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(5,4)

pixeltype() → interior point of old segment (4)

ID

(1)→(1,1)→(2,2)→(3,3) #end

(2)→(1,5)→(2,4)→(3,3) #end

(3)→(3,3)→(4,3)→(5,3) #end

(4)→(5,3)→(5,4)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(5,5)

pixeltype() → end point of old segment (4)

ID

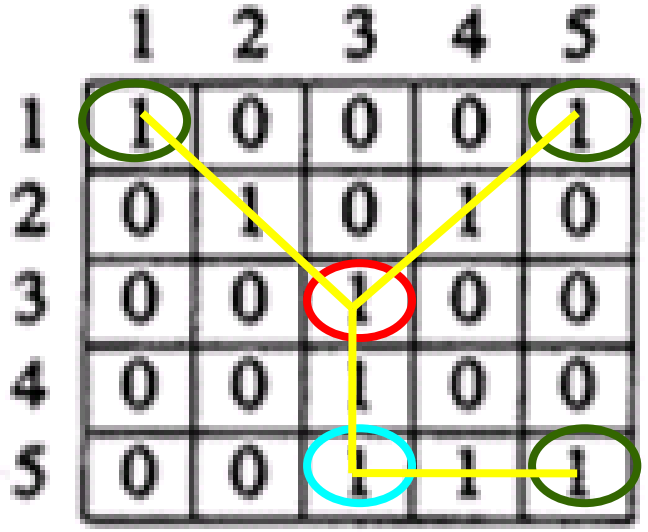
(1)→(1,1)→(2,2)→(3,3) #end

(2)→(1,5)→(2,4)→(3,3) #end

(3)→(3,3)→(4,3)→(5,3) #end

(4)→(5,3)→(5,4)→(5,5) #end

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES



	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

Segment ID	Length	List
1	3	(1,1)(2,2)(3,3)
2	3	(1,5)(2,4)(3,3)
3	3	(3,3)(4,3)(5,3)
4	3	(5,3)(5,4)(5,5)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES

	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(5,3)

pixeltype() → corner (3)

ID

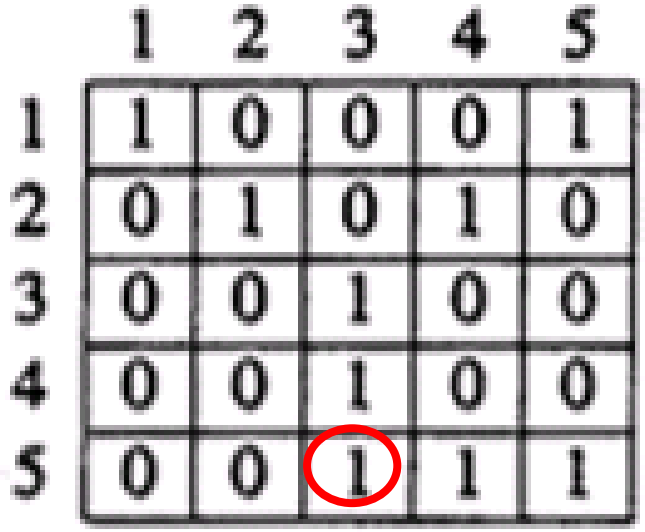
(1)→(1,1)→(2,2)→(3,3) #end

(2)→(1,5)→(2,4)→(3,3) #end

(3)→(3,3)→(4,3)→(5,3) #end

(4)→(5,3)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES



	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(5,3)

pixeltype()→interior point of old segment (3)

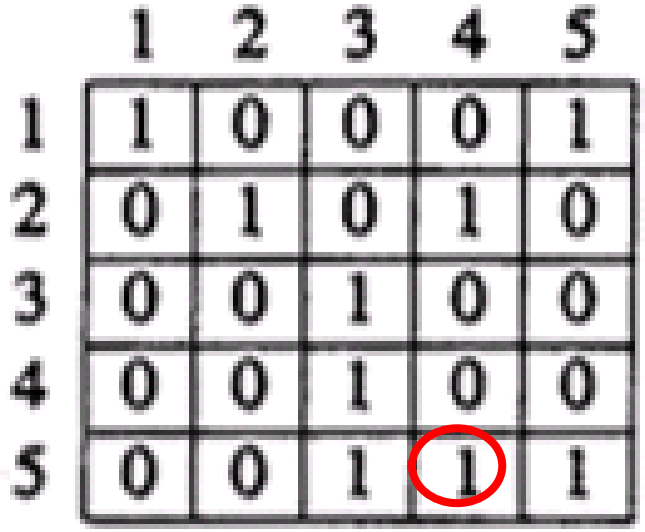
ID

(1)→(1,1)→(2,2)→(3,3) #end

(2)→(1,5)→(2,4)→(3,3) #end

(3)→(3,3)→(4,3)→(5,3)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES



	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(5,4)

pixeltype()→interior point of old segment (3)

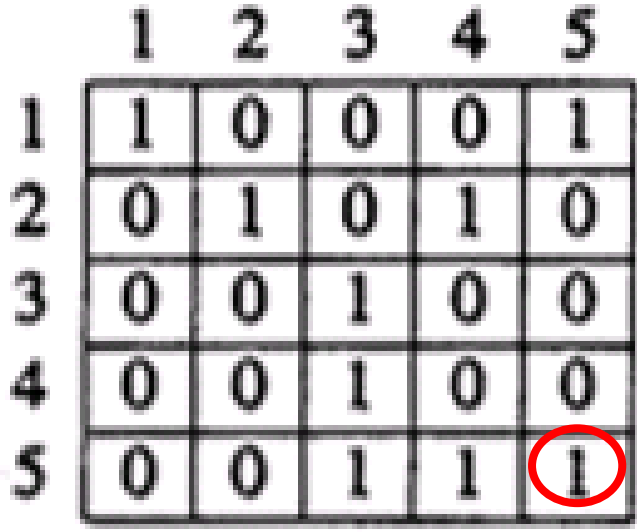
ID

(1)→(1,1)→(2,2)→(3,3) #end

(2)→(1,5)→(2,4)→(3,3) #end

(3)→(3,3)→(4,3)→(5,3)→(5,4)

11.3 LINKING ONE-PIXEL-WIDE EDGES OR LINES



	1	2	3	4	5
1	1	0	0	0	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	1	0	0
5	0	0	1	1	1

(5,5)

pixeltype()→end point of old segment (3)

ID

(1)→(1,1)→(2,2)→(3,3) #end

(2)→(1,5)→(2,4)→(3,3) #end

(3)→(3,3)→(4,3)→(5,3)→(5,4)→(5,5) #end

11.4 EDGE AND LINE LINKING USING DIRECTIONAL INFORMATION

- **Previous edge-track** : no directional information
- In this section, assume each **pixel is marked** to indicate whether it is an edge (line):
  - Yes, there is an associated angular direction
    - ➢pixels that have similar enough directions can form connected chains and be identified as an arc segment.
    - ➢The arc has a good fit to a simple curvelike line

11.4 EDGE AND LINE LINKING USING DIRECTIONAL INFORMATION

- The linking process -> scan the labeled edge image in top-down, left-right and if a labeled pixel has:
  - No previous labeled neighbors
    - It begins a new group
  - One previous labeled neighbors
    - T-test
  - Two or more previous labeled neighbors
    - T-test
    - Merge test

11.4 EDGE AND LINE LINKING USING DIRECTIONAL INFORMATION

- No previous labeled neighbors:
  - initialize the scatter of group -> $N_0\sigma_0^2$,

σ_0^2 : priori variance

N_0 : # of pixels

11.4 EDGE AND LINE LINKING USING DIRECTIONAL INFORMATION

- If an encountered label pixel has previously encountered labeled neighbors: **Measure t-statistic**

If γ is the mean angle for some group and θ is the angular direction for the given pixel, then the angular direction $\theta_{\min}$, which is the closest of $\theta, \theta+360°$, and $\theta-360°$ to γ, is defined in the following way:

$$\theta_{\min} = \begin{cases} \theta & \text{if } |\theta - \gamma| < |\theta^* - \gamma| \\ \theta^* & \text{otherwise} \end{cases} \quad \text{where} \quad \theta^* = \begin{cases} \theta + 360° & \text{if } \theta - \gamma < 0 \\ \theta - 360° & \text{otherwise} \end{cases}$$

$$t = \frac{|\theta_{\min} - \gamma| / \sqrt{(N+1)/N}}{\sqrt{S^2/N - 1}}$$

11.4 EDGE AND LINE LINKING USING DIRECTIONAL INFORMATION

- If $t < T_{\alpha,N-1}$, the **pixel is added to the group**; the mean and scatter of the group are updated:

$$\gamma_{new} \leftarrow (N\gamma_{old} + \theta_{min})/(N+1)$$
$$S^2 \leftarrow S^2 + N(\gamma_{old} - \gamma_{new})^2 + (\theta_{min} - \gamma_{new})^2$$
$$N \leftarrow N + 1$$
$$\gamma_{old} \leftarrow \gamma_{new}$$

11.4 EDGE AND LINE LINKING USING DIRECTIONAL INFORMATION

- If there are two or more previously encountered labeled neighbors,

$$\gamma_{2\min} = \begin{cases} \gamma_2 & \text{if } |\gamma_2 - \gamma_1| < |\gamma_2^* - \gamma_1| \\ \gamma_2^* & \text{otherwise} \end{cases} \quad \text{where} \quad \gamma_2^* = \begin{cases} \gamma_2 + 360^\circ & \text{if } \gamma_2 - \gamma_1 < 0 \\ \gamma_2 - 360^\circ & \text{otherwise} \end{cases}$$

$$t = \frac{|\gamma_{2\min} - \gamma_1| / \sqrt{\frac{1}{N_1} + \frac{1}{N_2}}}{\sqrt{(S_1^2 + S_2^2)/(N_1 + N_2 - 2)}}$$

11.4 EDGE AND LINE LINKING USING DIRECTIONAL INFORMATION

- If $t < T_{\alpha, N_1+N_2-2}$, then merge two groups
- Create a new group having:

$$N = N_1 + N_2$$
$$\gamma = (\gamma_1 N_1 + \gamma_2 N_2)/N$$
$$S^2 = S_1^2 + S_2^2 + N_1(\gamma_1 - \gamma)^2 + N_2(\gamma_2 - \gamma)^2$$

11.5 SEGMENTATION OF ARCS INTO SIMPLE SEGMENTS

- **Arc segmentation**: partition extracted digital arc sequence into digital arc subsequences (each is a maximal sequence that can fit a straight or curve line)
- Simple arc segment: straight-line or curved-arc segment
- The endpoints of the subsequences are called **corner points** or **dominant points**.

11.5 SEGMENTATION OF ARCS INTO SIMPLE SEGMENTS

- Identification of all locations:
  - have sufficiently high curvature
  - are enclosed by different lines and curves

- Techniques: iterative endpoint fitting and splitting, using tangent angle deflection, or high curvature as basis of the segmentation

11.5.1 ITERATIVE ENDPOINT FIT AND SPLIT

- To segment a digital arc sequence into subsequences that are sufficiently straight.
- Require one distance threshold d^*

- $L = \{(r, c) | \alpha r + \beta c + \gamma = 0\}$ $where$ $|(\alpha, \beta)| = 1$
  - *The straight line composed by 2 endpoints of this arc*

11.5.1 ITERATIVE ENDPOINT FIT AND SPLIT

- $d_i = |\alpha r_i + \beta c_i + \gamma| / |(\alpha, \beta)| = |\alpha r_i + \beta c_i + \gamma|$
  - *The length of the perpendicular line composed by any single pixel of this arc and L*

- $d_m = \max(di)$
- If $d_m > d*$, then split at the point (r_m, c_m)

11.5.1 ITERATIVE ENDPOINT FIT AND SPLIT

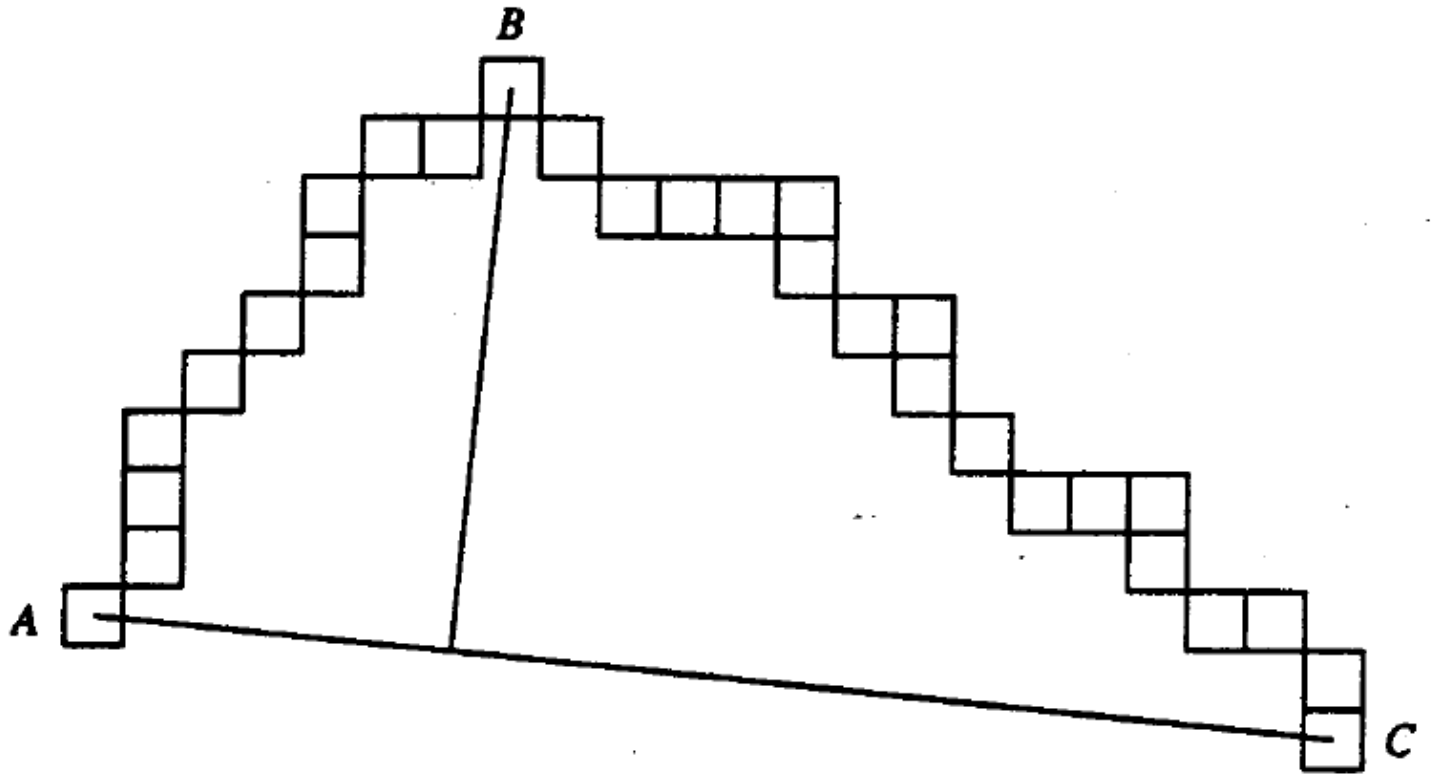


Figure 11.4 Geometry of the iterative endpoint fit and split. The pixel having the farthest distance to the line AC is the pixel B. The iterative endpoint fit and split segments the arc sequence at pixel B, creating two arc subsequences, each of which better fit a straight-line segment.

11.5.1 ITERATIVE ENDPOINT FIT AND SPLIT

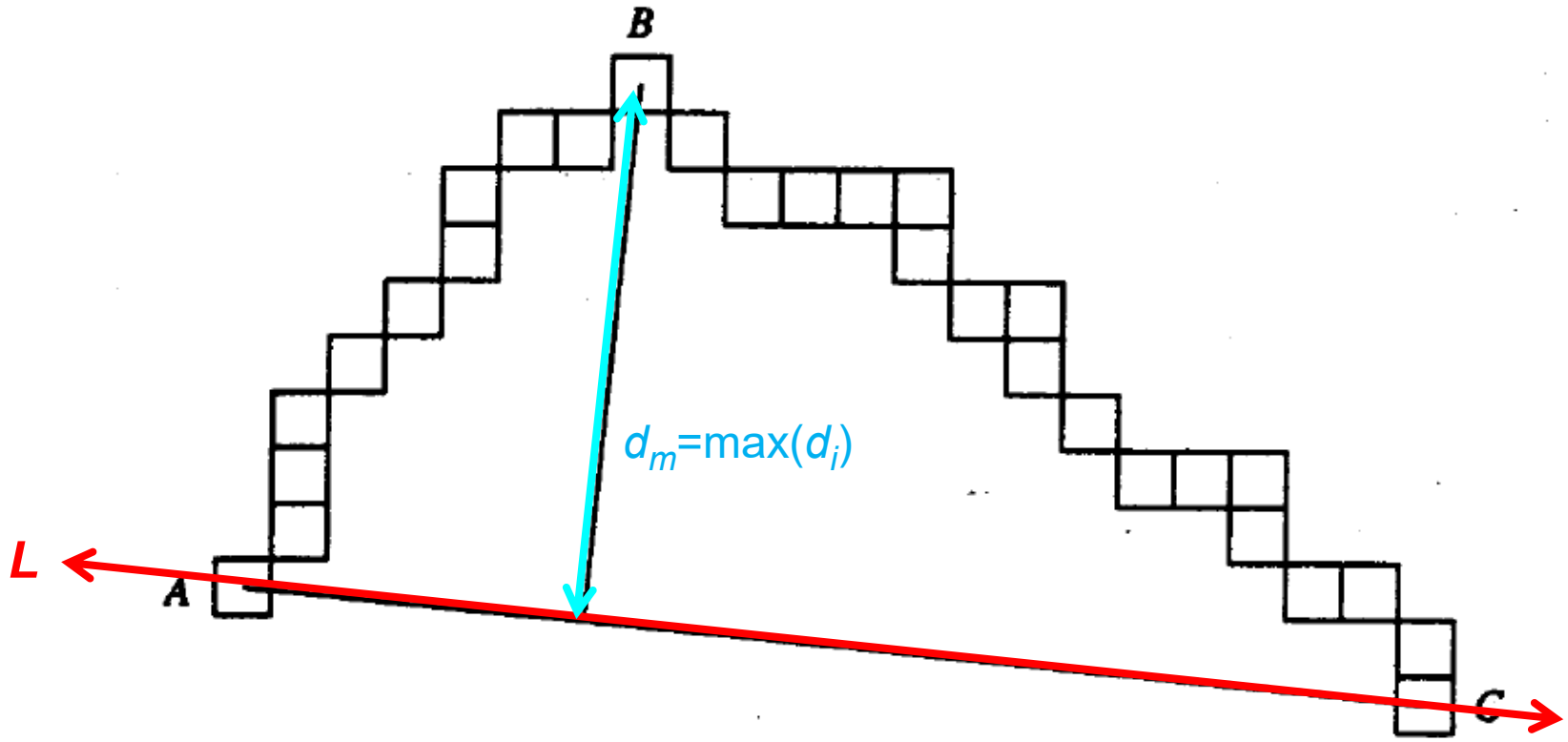


Figure 11.4 Geometry of the iterative endpoint fit and split. The pixel having the farthest distance to the line AC is the pixel B. The iterative endpoint fit and split segments the arc sequence at pixel B, creating two arc subsequences, each of which better fit a straight-line segment.

11.5.1 ITERATIVE ENDPOINT FIT AND SPLIT

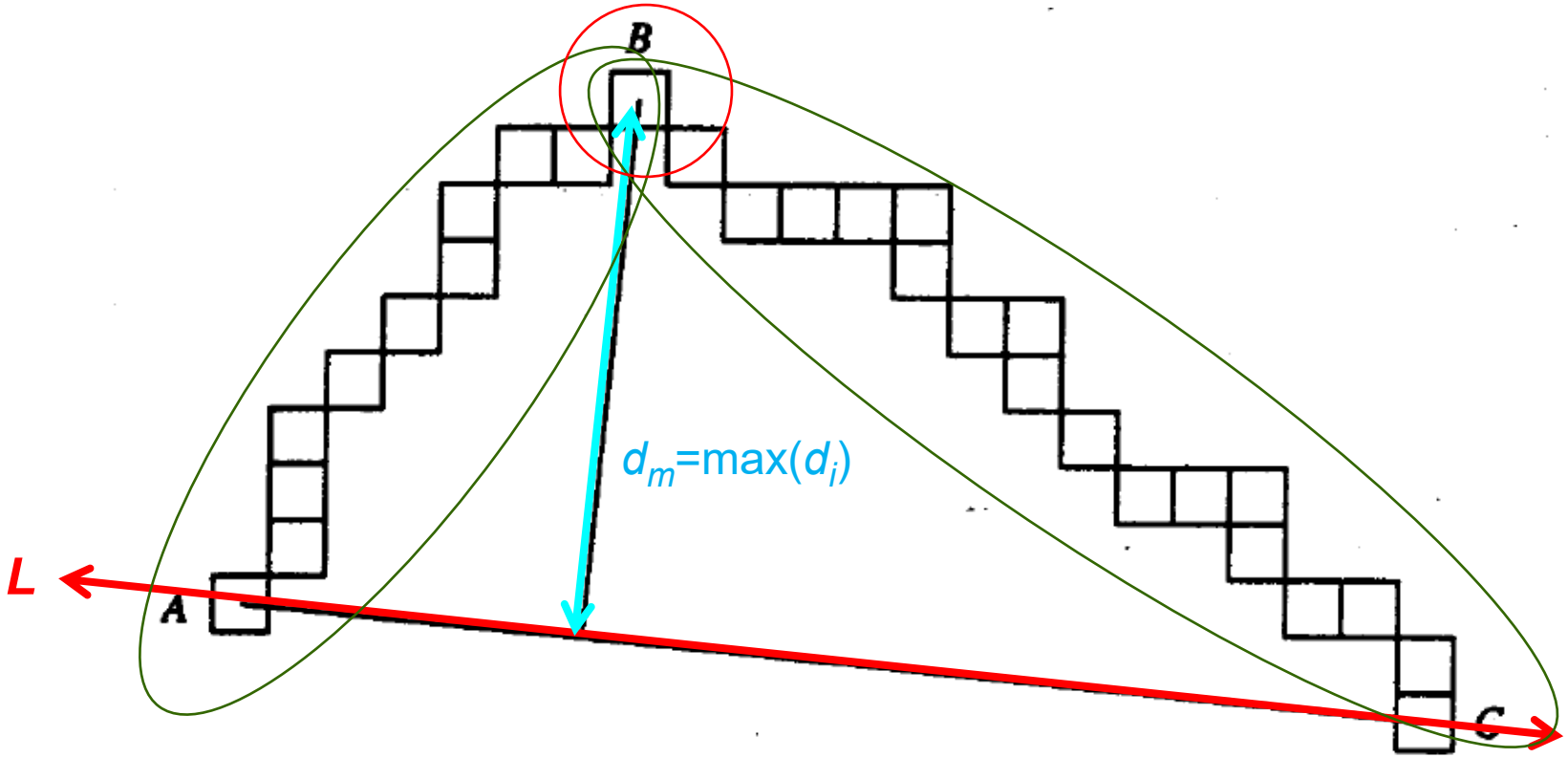


Figure 11.4 Geometry of the iterative endpoint fit and split. The pixel having the farthest distance to the line AC is the pixel B. The iterative endpoint fit and split segments the arc sequence at pixel B, creating two arc subsequences, each of which better fit a straight-line segment.

11.5.1 ITERATIVE ENDPOINT FIT AND SPLIT

- Circular arc sequence
  - ➢Initially split by two points apart in any direction
- Sequence only composed of two line segments
  - ➢Golden section search

11.5.2 TANGENTIAL ANGLE DEFLECTION

- To identify the locations where two line segments meet and form an angle.

➢ $\boldsymbol{a_n(k)} = (r_{n\_k} - r_n, c_{n\_k} - c_n)$

➢ $\boldsymbol{b_n(k)} = (r_n - r_{n+k}, c_n - c_{n+k})$

$$\cos[\theta_n(k)] = \frac{a_n(k)b_n(k)^T}{\| a_n(k) \| \| b_n(k) \|}$$

11.5.2 TANGENTIAL ANGLE DEFLECTION

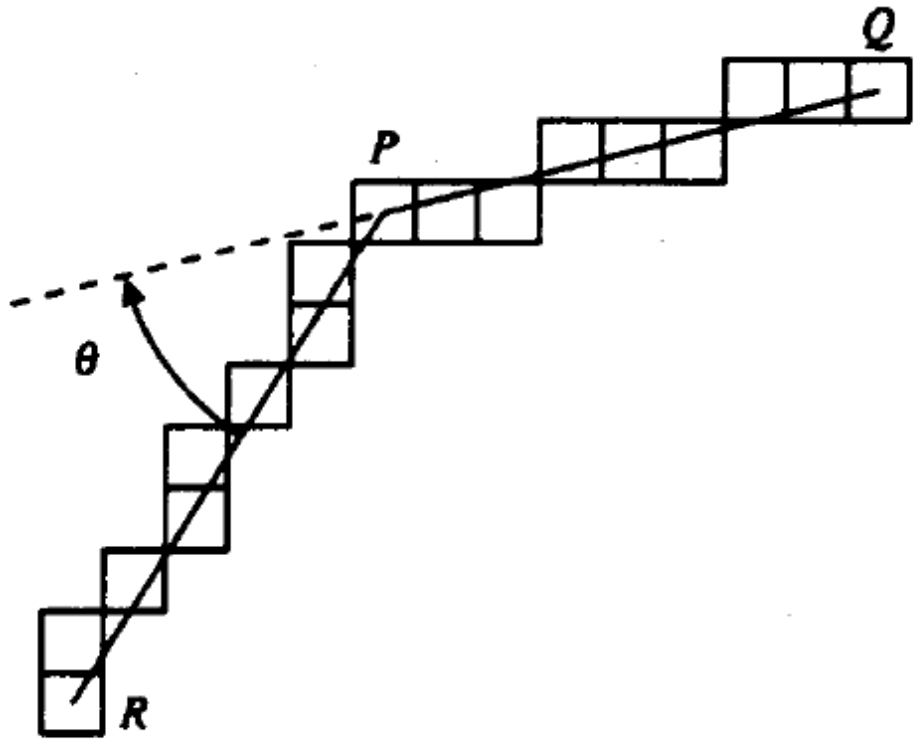


Figure 11.5 Geometry of the exterior angle formed by two line segments.

11.5.2 TANGENTIAL ANGLE DEFLECTION

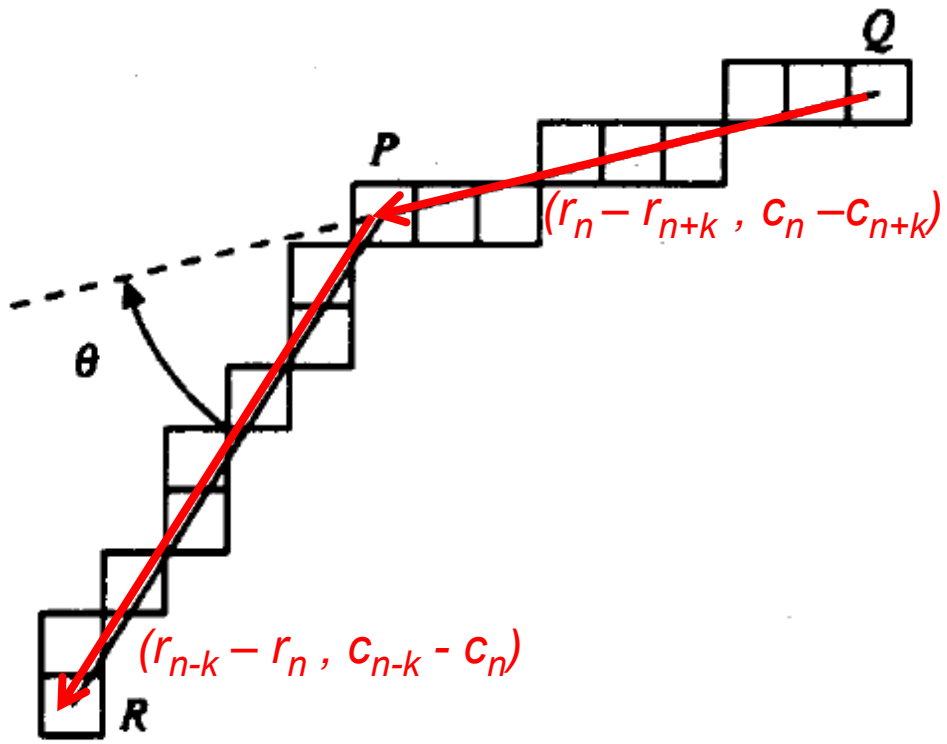
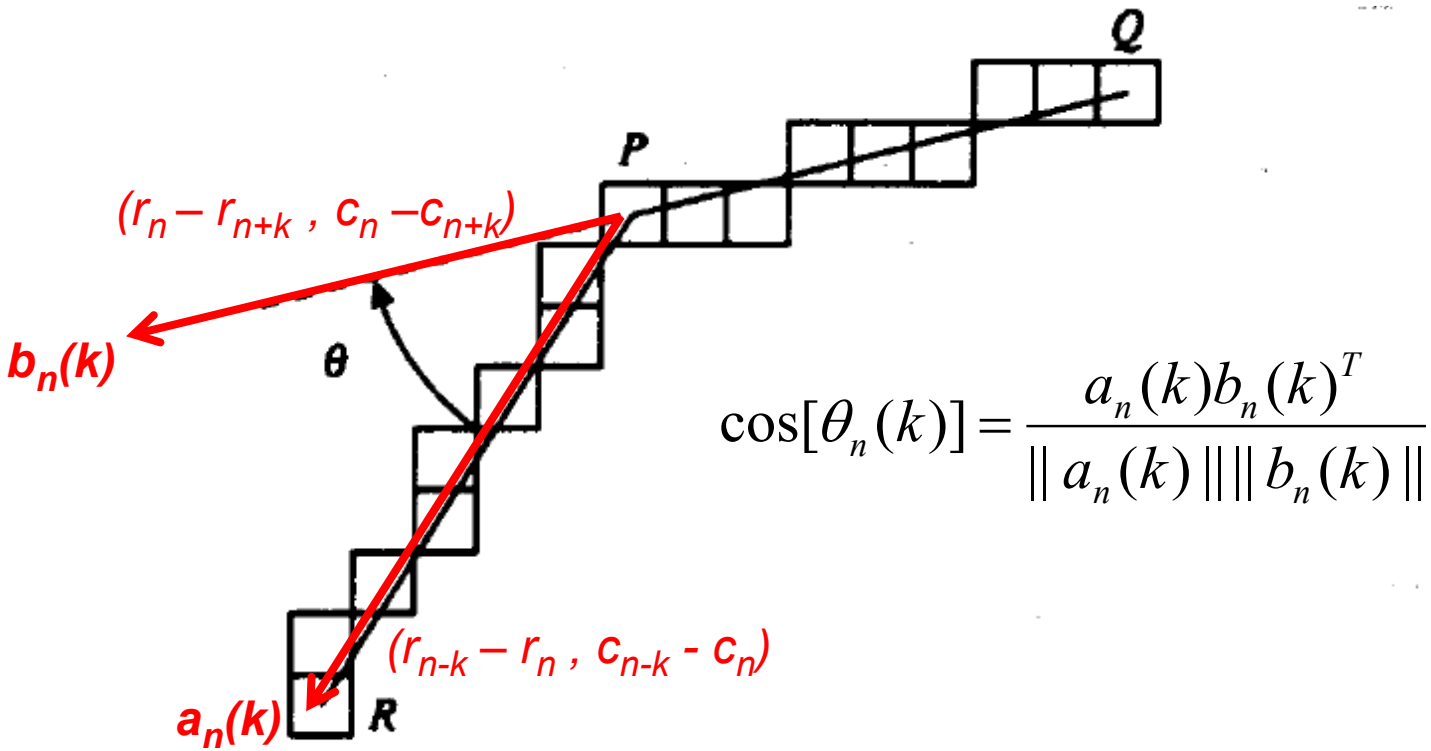


Figure 11.5 Geometry of the exterior angle formed by two line segments.

11.5.2 TANGENTIAL ANGLE DEFLECTION



$$\cos[\theta_n(k)] = \frac{a_n(k)b_n(k)^T}{\| a_n(k) \| \| b_n(k) \|}$$

Figure 11.5 Geometry of the exterior angle formed by two line segments.

11.5.2 TANGENTIAL ANGLE DEFLECTION

- At a place where two line segments meet

  → the angle will be larger → $\cos\theta_n(k_n)$ smaller

11.5.3 UNIFORM BOUNDED-ERROR APPROXIMATION

- Segment arc sequence into maximal pieces whose points deviate ≤ given amount
- Optimal algorithms: excessive computational complexity

11.5.3 UNIFORM BOUNDED-ERROR APPROXIMATION

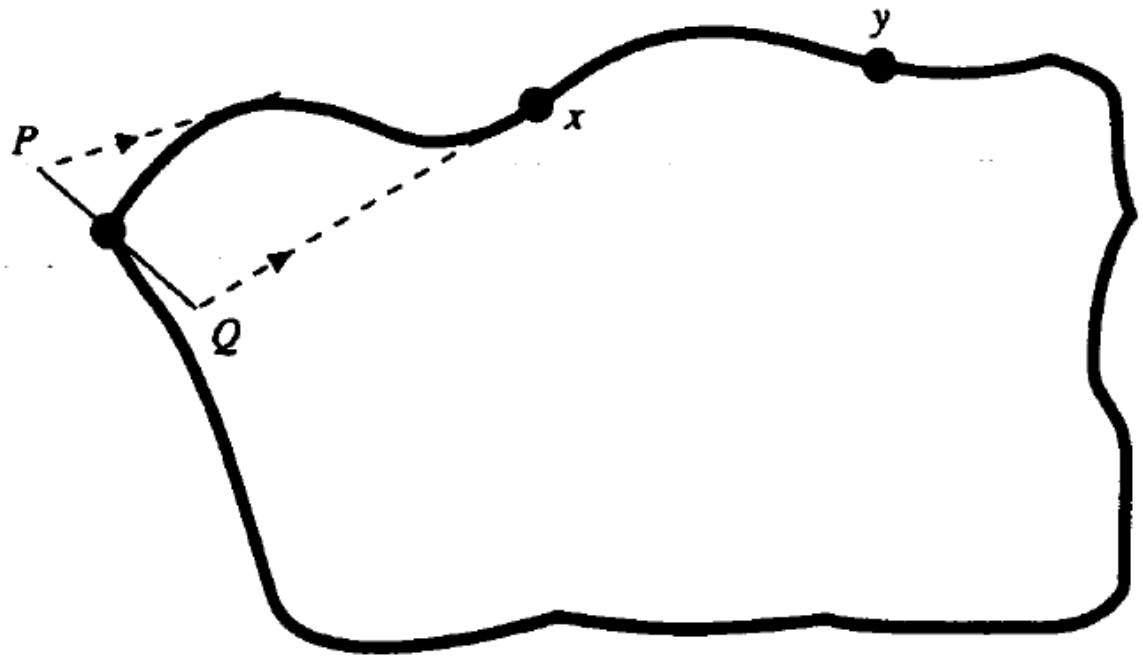


Figure 11.6 Geometry of Tomek's segmentation technique. At the time point x is being processed, the bounding directions emanating from P and Q are still pointing toward each other. However, at the time point y is processed, the bounding directions emanating from P and Q have become parallel and no longer point toward each other. The point y terminates the segment.

11.5.4 BREAKPOINT OPTIMIZATION

- After an initial segmentation: shift breakpoints to produce a better arc segmentation
- First → shift odd final point (i.e. even beginning point) and see whether the max. error is reduced by the shift.
- If reduced, then keep the shifted breakpoints.
- Then→ shift even final point (i.e. odd beginning point) and do the same things.

11.5.4 BREAKPOINT OPTIMIZATION

```
procedure move_break_points(S,segmentlist,snag);
sflag := 0;
flag = 1;
L = length(segmentlist);
while flag = 1 do
   begin
      flag = 0;
      for j := 1 to L-1 by 2 do
       adjust(S,j,segmentlist,flag);
      end for sflag := or(sflag,flag);
      for j := 2 to L by 2 do
         adjust(S,j,segmentlist,flag)
      end for sflag := or(sflag,flag);
   end
end move_break_points;
```

11.5.5 SPLIT AND MERGE

- First: split arc into segments with the error sufficiently small
- Second: merge successive segments if resulting merged segment has sufficiently small error
- Third: try to adjust breakpoints to obtain a better segmentation
- Repeat: until all three steps produce no further change

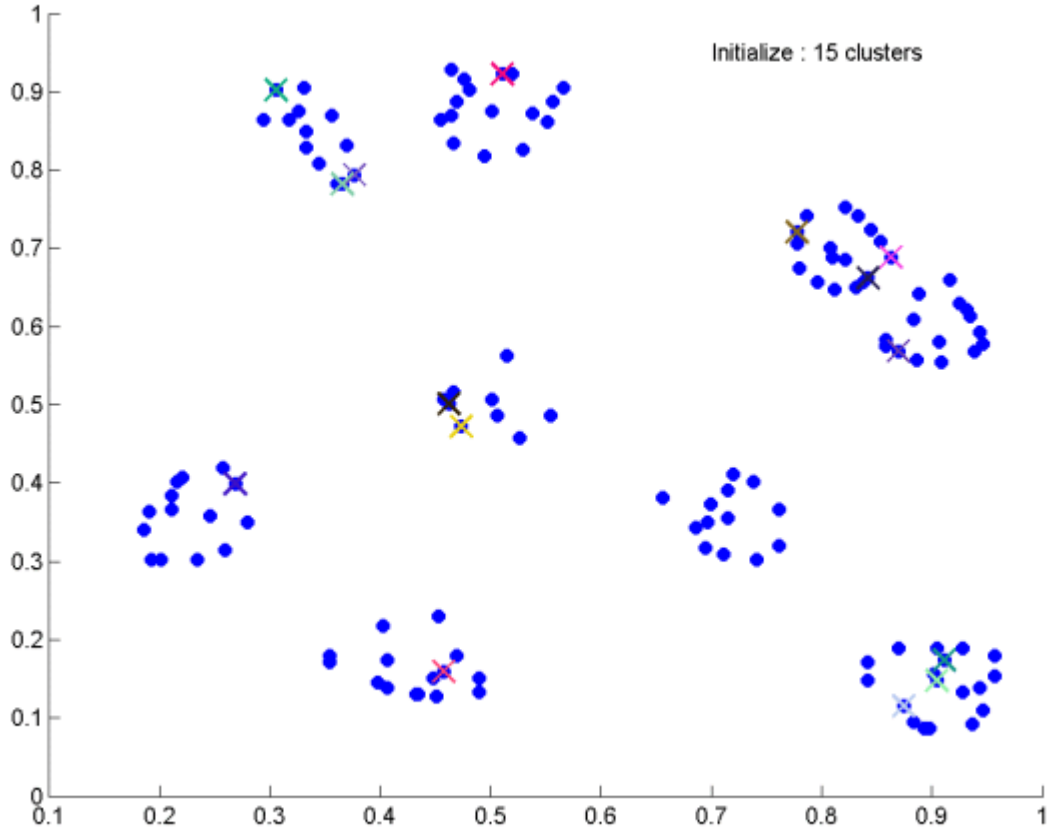
11.5.5 SPLIT AND MERGE

```
procedure arc_split_and_merge(S,ε_max,segmentlist);
segmentlist := nil;
add(segmentlist,(1,N));
repeat
   begin
      endpoint_fit_and_split(S,ε_max,segmentlist,segmentsplitlist,sflag1);
      endpoint_fit_and_merge(S,ε_max,segmentsplitlist,segmentlist,sflag2);
      move_break_points(S,segmentlist,sflag3)
   end;
until sflag1=0 and sflag2=0 and sflag3=0
end arc_split_and_merge;
```

11.5.6 ISODATA SEGMENTATION

- Iterative Self-organizing Data Analysis Techniques Algorithm
- Iterative isodata line-fit clustering procedure: determines line-fit parameter
- Then each point assigned to cluster whose line fit closest to the point

11.5.6 ISODATA SEGMENTATION



11.5.6 ISODATA SEGMENTATION

```
procedure isodata_linefit(S,P^o,P,(α,β,γ));
  q:=0;
  repeat
    begin
      q:=q+1;
      (α^(q-1),β^(q-1),γ^(q-1)) := linefit(P^(q-1));
      P^q:= nil;
      for i=1 to N do
        begin
          k:=indexmindist((r_i,c_i),(α^(q-1),β^(q-1),γ^(q-1)));
                P_k^q := add(P_k^q,(r_i,c_i));
            end
        end for
      end
    until P^(q-1) = P^q;
    P = P^(q-1);
    (α,β,γ) = (α^(q-1),β^(q-1),γ^(q-1))
end isodata_linefit
```

11.5.6 ISODATA SEGMENTATION

```
function indexmindist((r,c),(α,β,γ));
  d:=verylarge_number;
  for k=1 to K do
    begin
      dist:=|α(k)r + β(k)c + γ|;
      if dist < d then
        begin
          d:=dist;
          indexmindist := k
        end
    end
  end for
end indexmindist
```

11.5.7 CURVATURE

- The curvature κ is defined at a point of arc length s along the curve by

  Δs : the change in arc length

  $\Delta\theta$: the change in tangent angle

$$\kappa(s) = \lim_{\Delta s \to 0} \frac{\Delta\theta}{\Delta s}$$

11.5.7 CURVATURE

- Natural curve breaks: curvature maxima and minima
- Curvature passes: through zero local shape changes from convex to concave

11.5.7 CURVATURE

- Surface elliptic: when limb in line drawing is convex
- Surface hyperbolic: when its limb is concave
- Surface parabolic: wherever curvature of limb zero

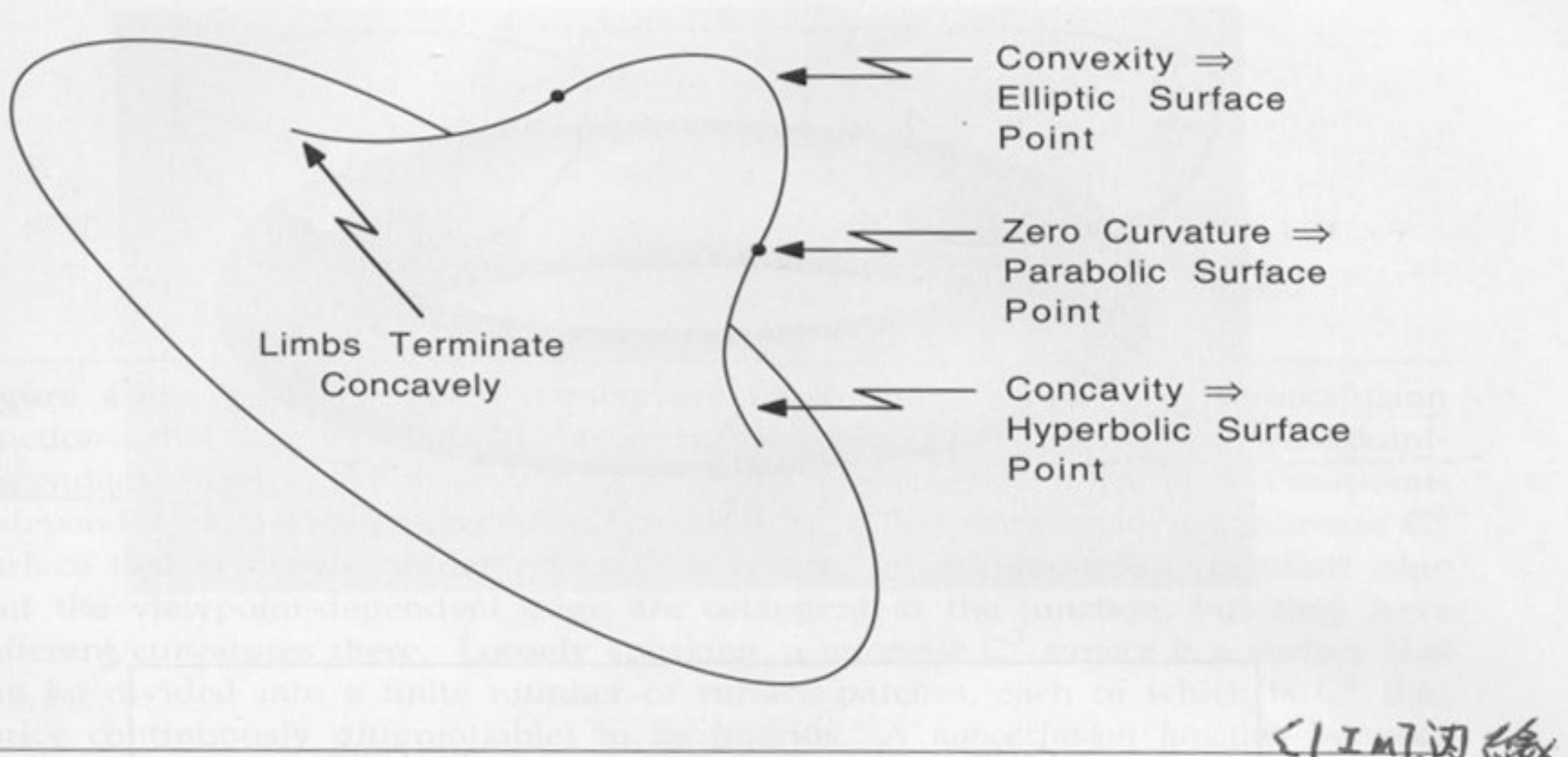


Figure 4.14 The qualitative relationship between the curvature of a limb in a line drawing and the shape of the surface in the vicinity of the limb in space. A C^3 surface is elliptic in space wherever its limb in a line drawing is convex with respect to the object; it is hyperbolic, wherever its limb is concave; it is parabolic, wherever the curvature of its limb is zero. The figure illustrates these assertions with a bell-shaped protrusion from a surface. The figure also illustrates that nonjunction terminations of limbs in line drawings of C^3 surfaces are always concave with respect to the object, which is equivalent to the assertion that cusp singularities of projection can occur only within hyperbolic surface regions. (After [Koenderink 1984b] and [Koenderink and van Doorn 1982].)

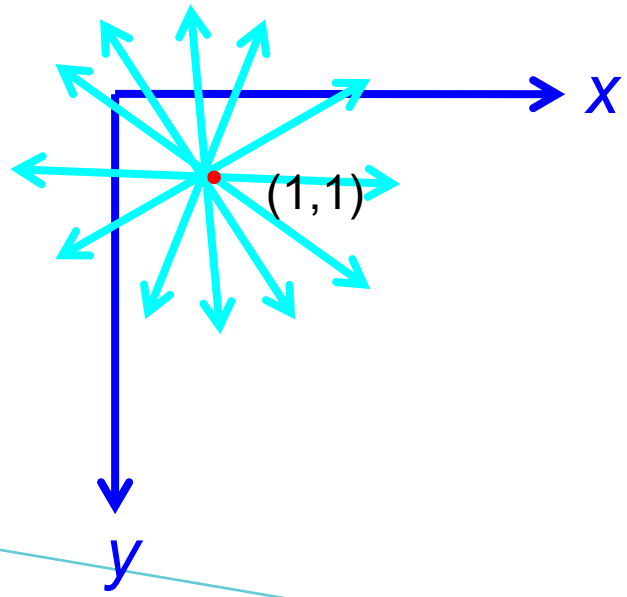
11.6 HOUGH TRANSFORM

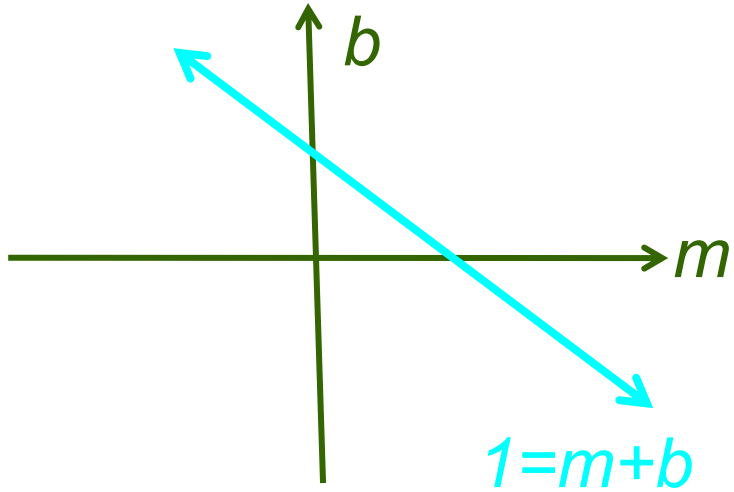
- Hough Transform: method for detecting **straight lines** and **curves** on gray level images.
- Hough Transform: template matching
- The Hough transform algorithm requires an **accumulator array** whose dimension corresponds to the number of unknown parameters in the equation of the family of curves being sought.

FINDING STRAIGHT-LINE SEGMENTS

- Line equation → $y=mx+b$
- Point → (x,y)

  Slope → m , Intercept → b

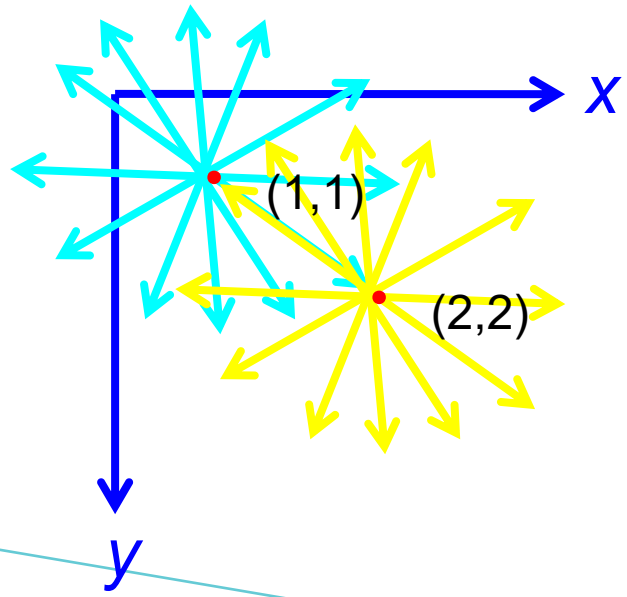


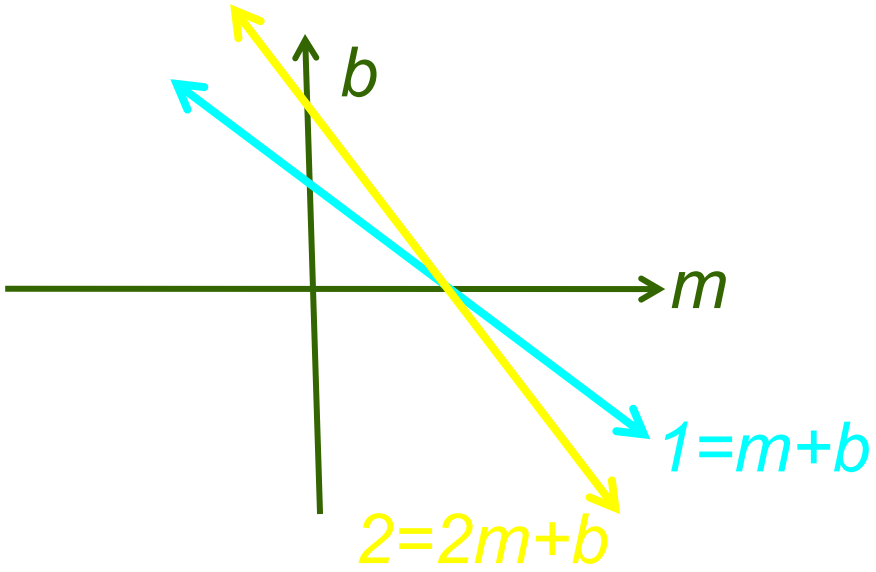


FINDING STRAIGHT-LINE SEGMENTS

- Line equation → $y=mx+b$
- Point → (x,y)

  Slope → m , intercept → b

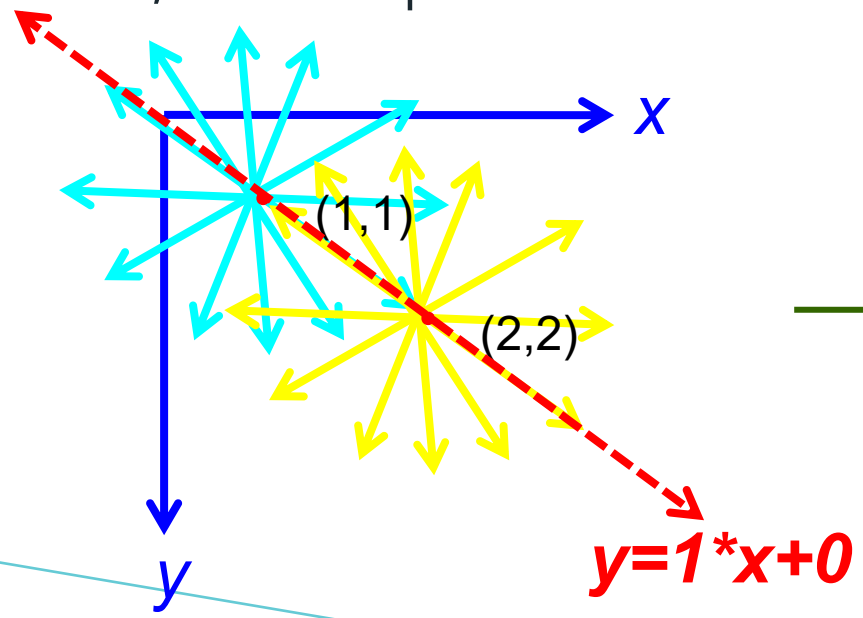


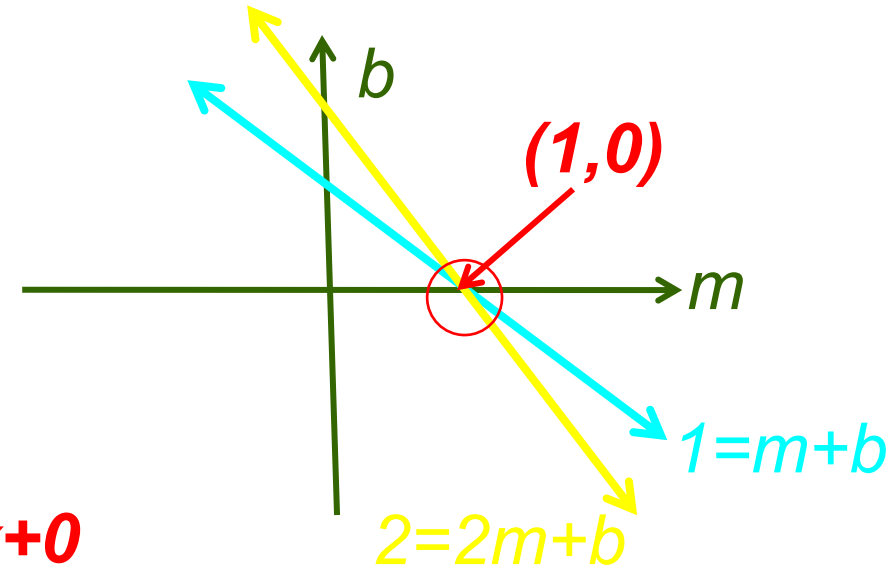


FINDING STRAIGHT-LINE SEGMENTS

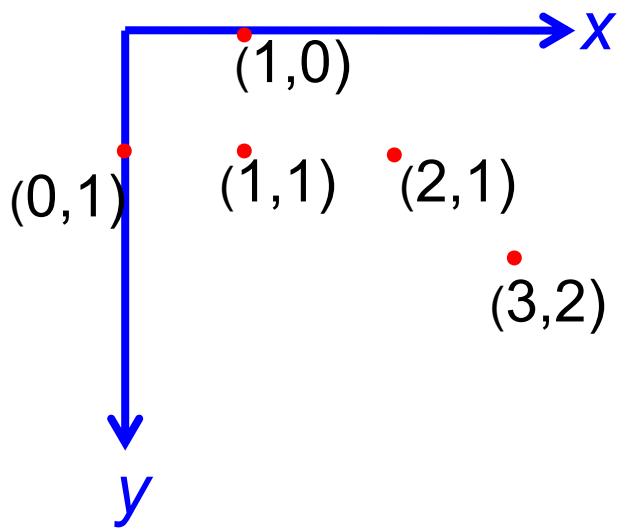
- Line equation → $y=mx+b$
- Point → (x,y)

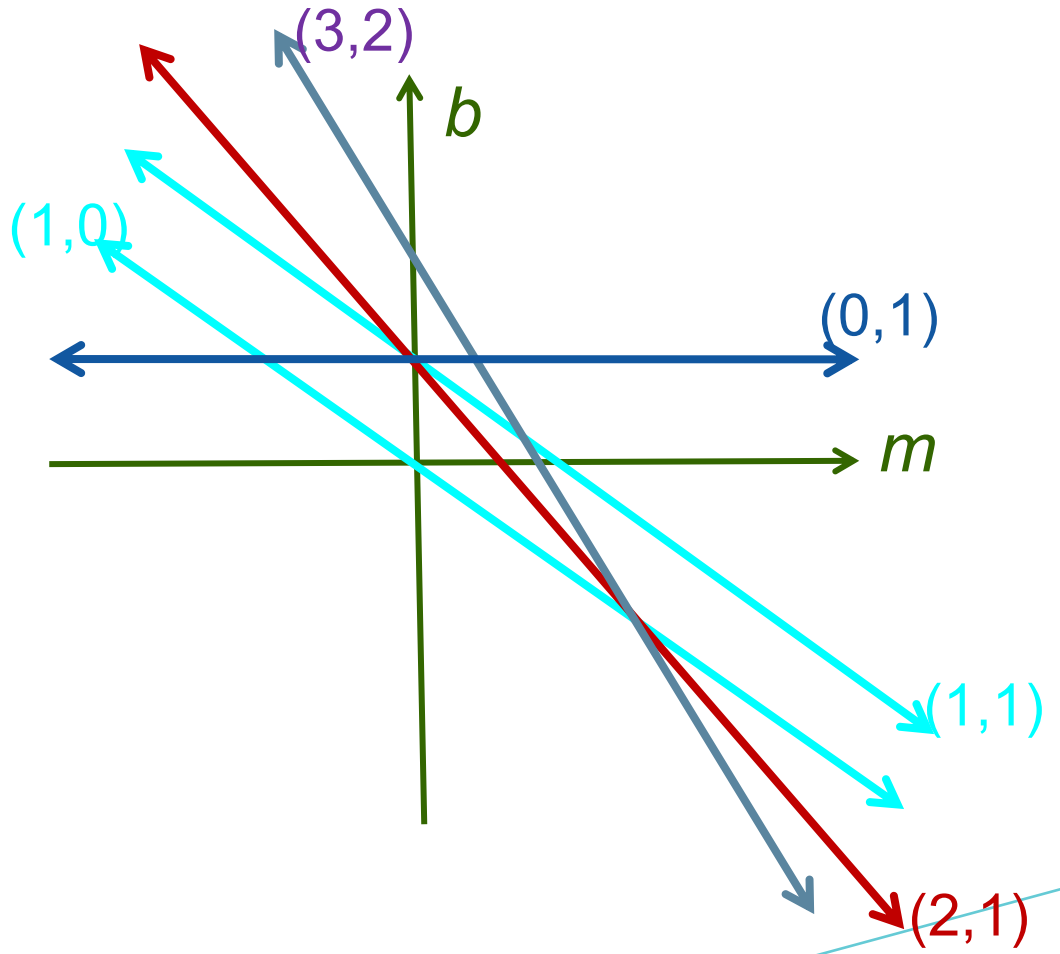
  Slope → m , intercept → b



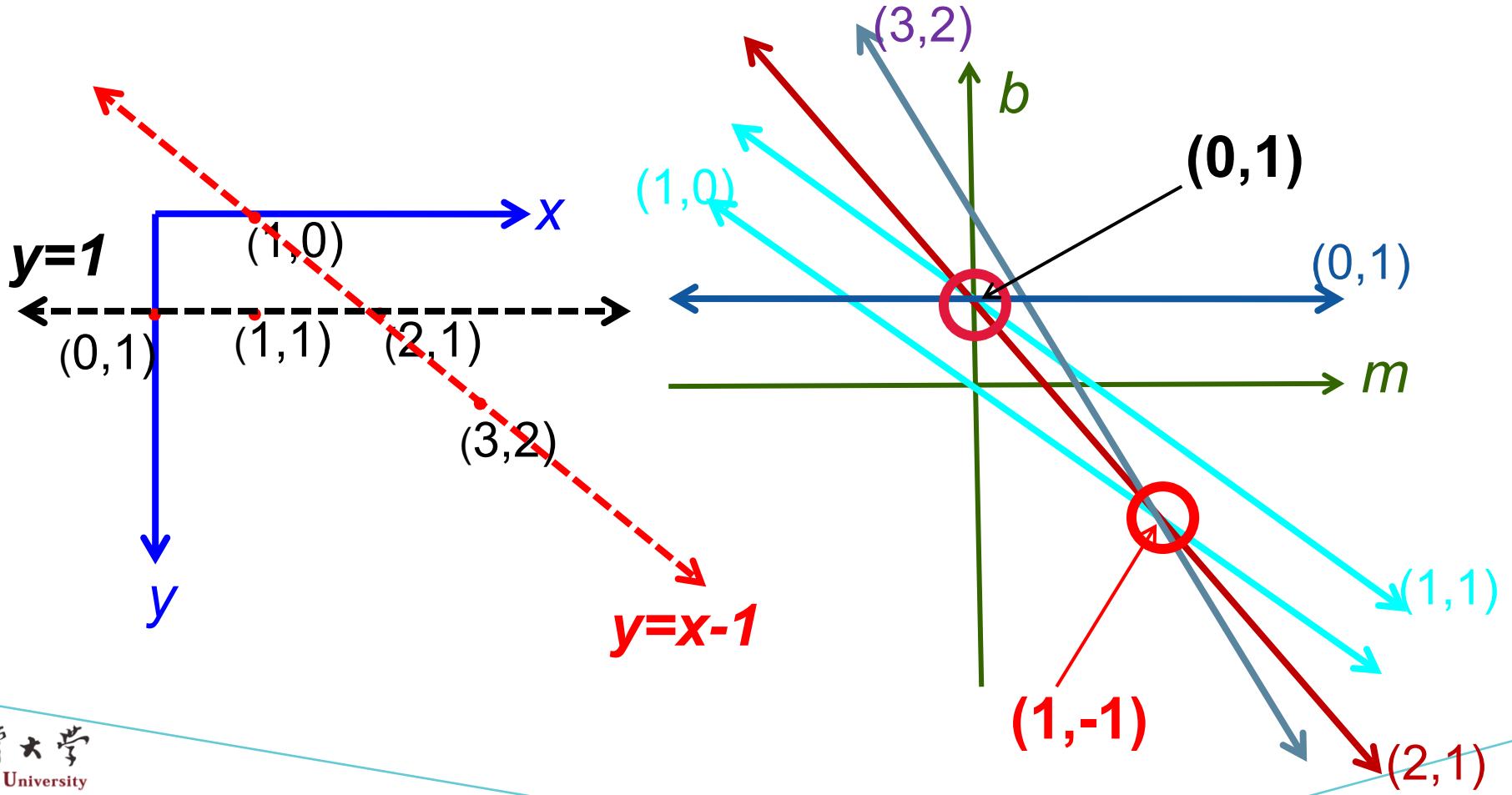


EXAMPLE



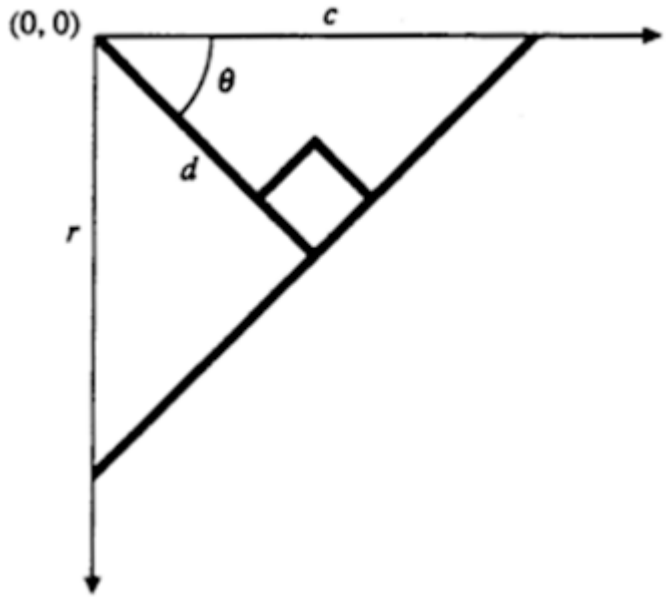


EXAMPLE

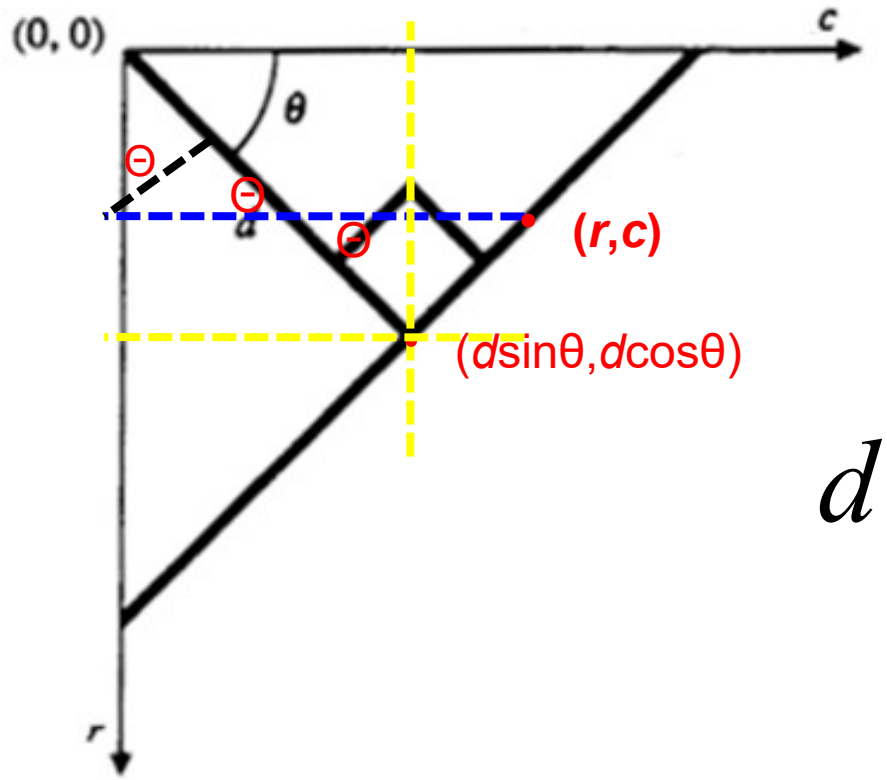


FINDING STRAIGHT-LINE SEGMENTS

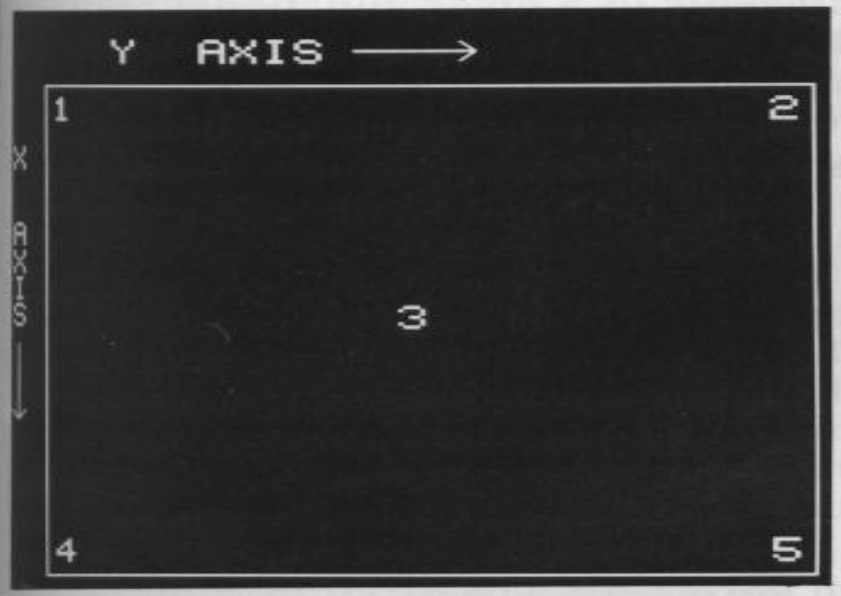
- Vertical lines → $m=\infty$ → doesn't work
- d: perpendicular distance from line to origin
- θ: the angle the perpendicular makes with the x-axis (column axis)



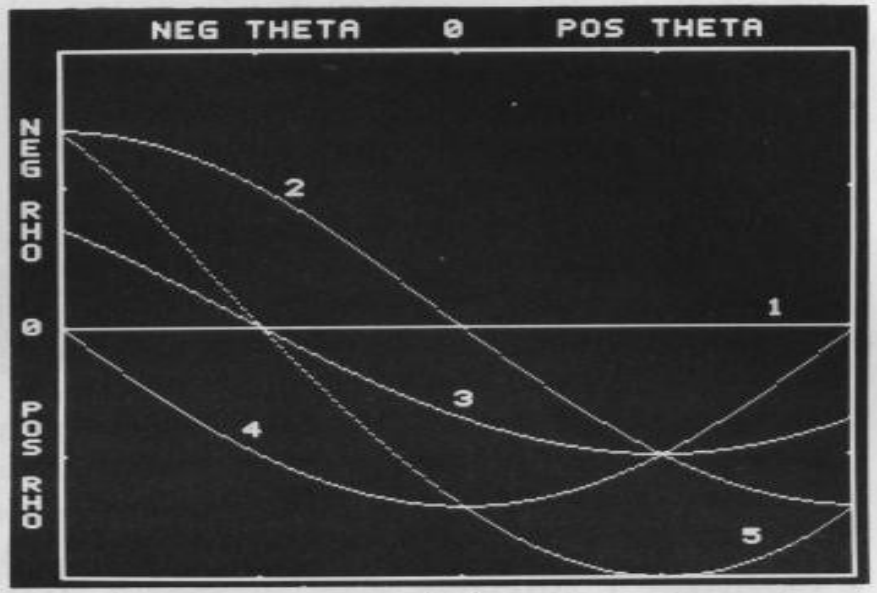
FINDING STRAIGHT-LINE SEGMENTS



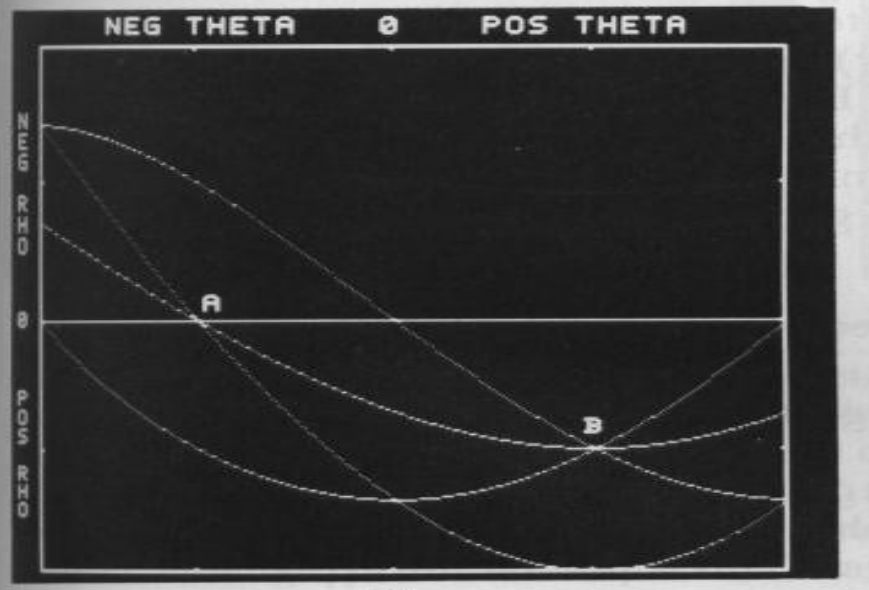
$$d = r\sin\theta + c\cos\theta$$



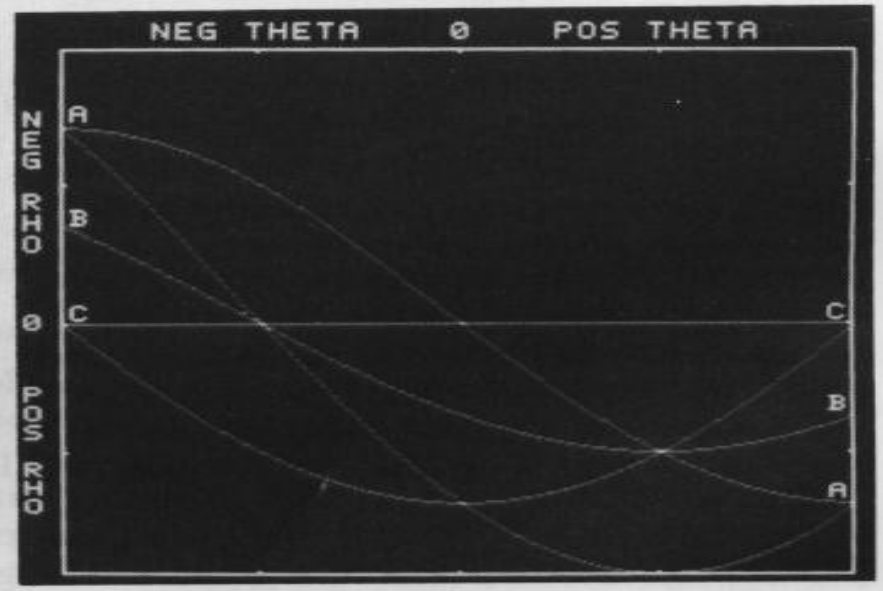
(a)



(b)



(c)



(d)

Figure 7.18 *Illustration of the Hough transform. (Courtesy of D. R. Cate, Texas Instruments, Inc.)*

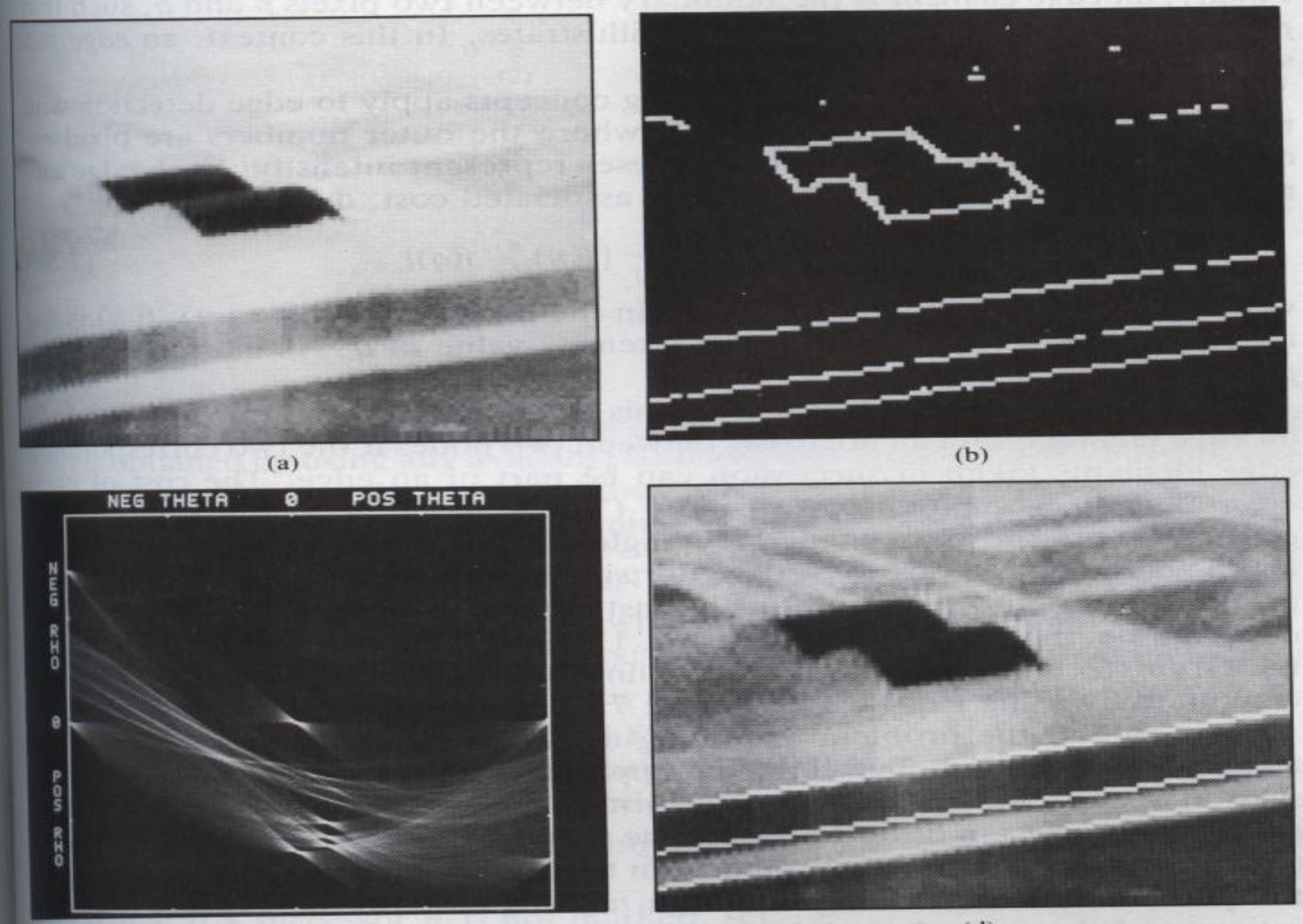
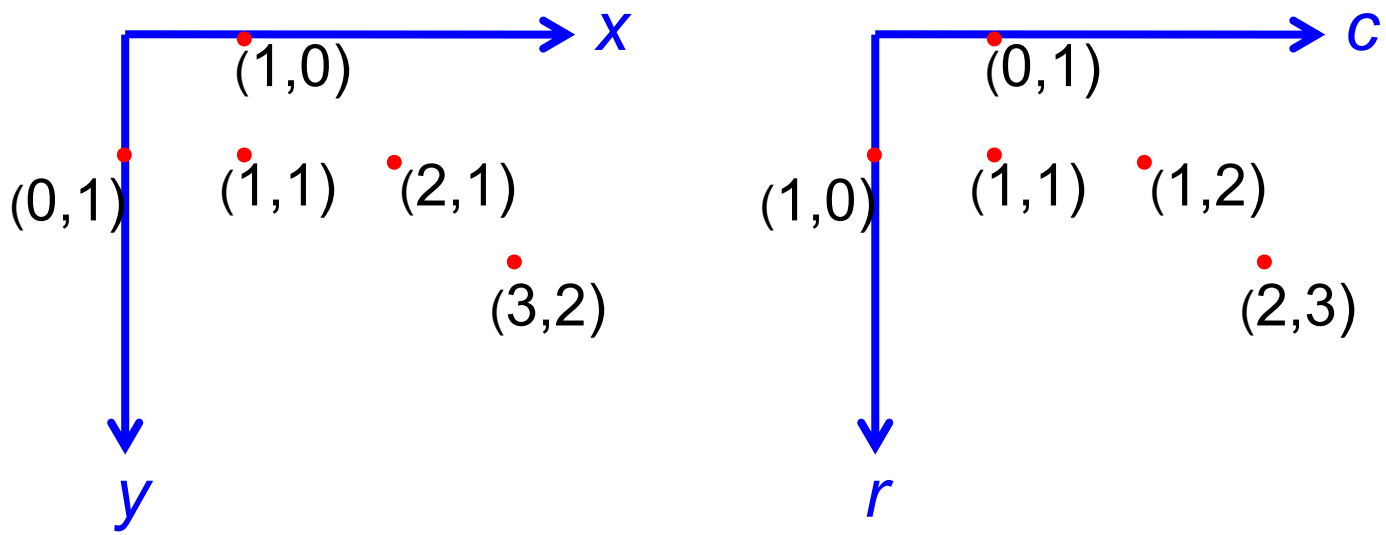


Figure 7.19 *(a) Infrared image; (b) gradient image; (c) Hough transform; (d) linked pixels. (Courtesy of D. R. Cate, Texas Instruments, Inc.)*

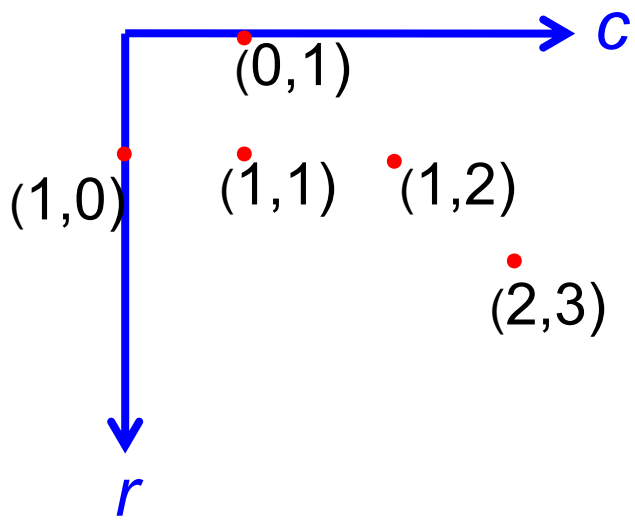
EXAMPLE

$$d = r\sin\theta + c\cos\theta$$



EXAMPLE

$$d = r\sin\theta + c\cos\theta$$



	-45°	0°	45°	90°
(0,1)	0.707	1	0.707	0
(1,0)	-0.707	0	0.707	1
(1,1)	0	1	1.414	1
(1,2)	0.707	2	2.121	1
(2,3)	0.707	3	3.535	2

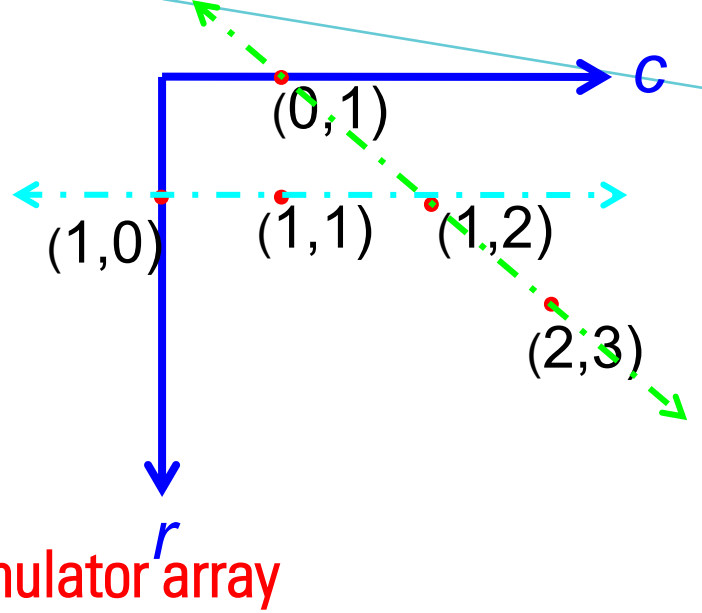
EXAMPLE

	-45°	0°	45°	90°
(0,1)	0.707	1	0.707	0
(1,0)	-0.707	0	0.707	1
(1,1)	0	1	1.414	1
(1,2)	0.707	2	2.121	1
(2,3)	0.707	3	3.535	2

accumulator array

	-0.707	0	0.707	1	1.414	2	2.121	3	3.535
-45°	1	1	3	-	-	-	-	-	-
0°	-	1	-	2	-	1	-	1	-
45°	-	-	2	-	1	-	1	-	1
90°	-	1	-	3	-	1	-	-	-

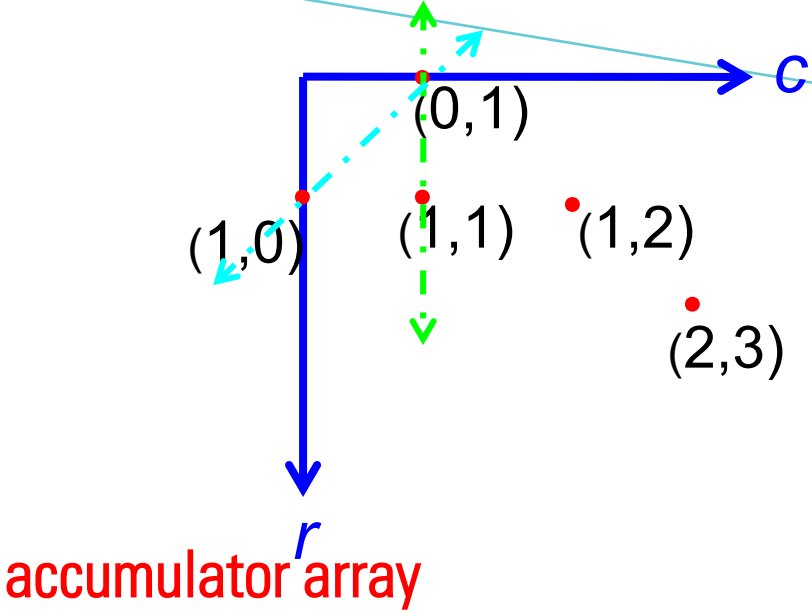
EXAMPLE



accumulator array

	-0.707	0	0.707	1	1.414	2	2.121	3	3.535
-45°	1	1	3	-	-	-	-	-	-
0°	-	1	-	2	-	1	-	1	-
45°	-	-	2	-	1	-	1	-	1
90°	-	1	-	3	-	1	-	-	-

EXAMPLE



	-0.707	0	0.707	1	1.414	2	2.121	3	3.535
-45°	1	1	3	-	-	-	-	-	-
0°	-	1	-	2	-	1	-	1	-
45°	-	-	2	-	1	-	1	-	1
90°	-	1	-	3	-	1	-	-	-

FINDING STRAIGHT-LINE SEGMENTS

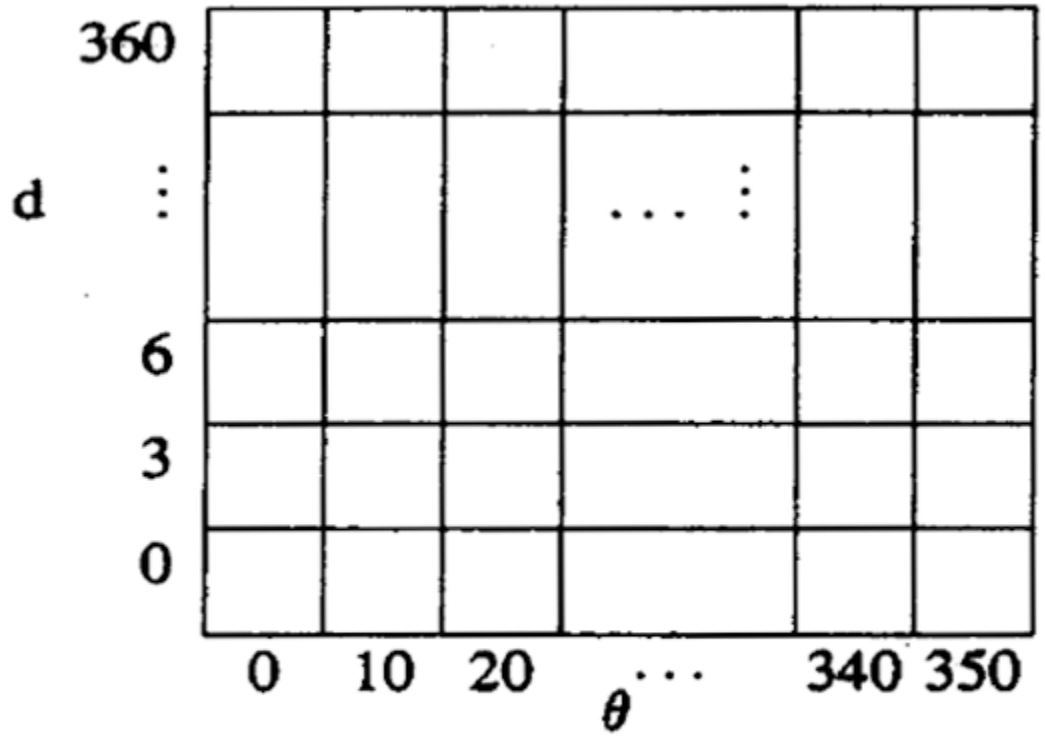
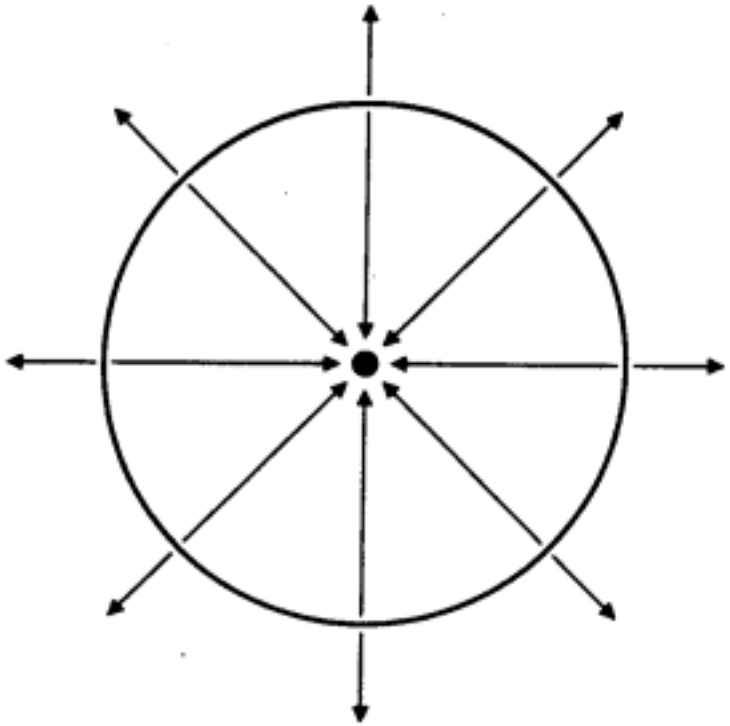


Figure 11.9 Accumulator array for finding straight-line segments in images of size 256 × 256.

FINDING CIRCLES

- r : row
- c : column
- r_0 : row-coordinate of the center
- c_0 : column-coordinate of the center
- d: radius
- $(r - r_0)^2 + (c - c_0)^2 = d^2$: implicit equation for a circle

FINDING CIRCLES



$$(r - r_0)^2 + (c - c_0)^2 = d^2$$

$$r = r_0 + d\sin\theta$$

$$c = c_0 + d\cos\theta$$

EXTENSIONS

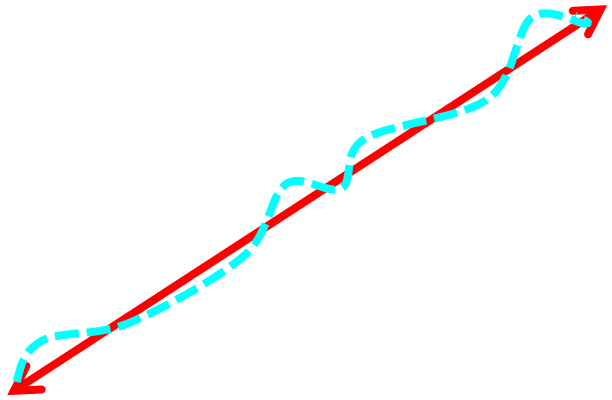
- The Hough transform method can be extended to any curve with analytic equation of the form $f(\vec{x}, \vec{a}) = 0$, where $\vec{x}$ denotes an image point and $\vec{a}$ is a vector of parameters.

11.7 LINE FITTING

- (r_n, c_n) → points before noise perturbation
  → lie on the line $\alpha r_n + \beta c_n + \gamma = 0$
- $(\hat{r}_n, \hat{c}_n)$ → noisy observed value
  → $\hat{r}_n = r_n + \xi_n$

  $\hat{c}_n = c_n + \eta_n$

ξ_n, η_n → independent and identically distributed with mean 0 and variance σ^2



11.7 LINE FITTING

- Procedure for the **least-squares** fitting of line to observed noisy values
- Principle of **minimizing the squared residuals** under the constraint that $\hat{\alpha}^2 + \hat{\beta}^2 = 1$

$$\min \ \varepsilon^2 = \sum_{n=1}^{N} (\hat{\alpha}\hat{r}_n + \hat{\beta}\hat{c}_n + \hat{\gamma})^2 \quad \text{subject to} \quad \hat{\alpha}^2 + \hat{\beta}^2 = 1$$

- Lagrange multiplier form:

$$\varepsilon^2 = \sum_{n=1}^{N} [(\hat{\alpha}\hat{r}_n + \hat{\beta}\hat{c}_n + \hat{\gamma})^2 - \lambda(\hat{\alpha}^2 + \hat{\beta}^2 - 1)] = \sum_{n=1}^{N} (\hat{\alpha}\hat{r}_n + \hat{\beta}\hat{c}_n + \hat{\gamma})^2 - \lambda(\hat{\alpha}^2 + \hat{\beta}^2 - 1)N$$

11.7 LINE FITTING

$$\epsilon^2 = \sum^{N} (\hat{\alpha}\hat{r}_n + \hat{\beta}\hat{c}_n + \hat{\gamma})^2 - \lambda(\hat{\alpha}^2 + \hat{\beta}^2 - 1)N$$

$$\frac{\partial \epsilon^2}{\partial \hat{\gamma}} = 2\sum_{n=1}^{N} (\hat{\alpha}\hat{r}_n + \hat{\beta}\hat{c}_n + \hat{\gamma}) = 0 \longrightarrow \hat{\gamma} = -(\hat{\alpha}\hat{\mu}_r + \hat{\beta}\hat{\mu}_c)$$

$$\hat{\mu}_r = \frac{1}{N}\sum_{n=1}^{N} \hat{r}_n \quad \text{and} \quad \hat{\mu}_c = \frac{1}{N}\sum_{n=1}^{N} \hat{c}_n$$

$$\hat{\mu}_{rr} = \frac{1}{N-1}\sum_{n=1}^{N} (\hat{r}_n - \hat{\mu}_r)^2$$

$$\hat{\mu}_{cc} = \frac{1}{N-1}\sum_{n=1}^{N} (\hat{c}_n - \hat{\mu}_c)^2$$

$$\hat{\mu}_{rc} = \frac{1}{N-1}\sum_{n=1}^{N} (\hat{x} - \mu_r)(\hat{c}_n - \mu_c)$$

$$\longrightarrow \frac{\partial \epsilon^2}{\partial \hat{\alpha}} = \sum_{n=1}^{N} 2\left[\hat{\alpha}(\hat{r}_n - \hat{\mu}_r) + \hat{\beta}(\hat{c}_n - \hat{\mu}_c)\right](\hat{r}_n - \hat{\mu}_r) - \lambda(2\hat{\alpha})N = 0$$

$$\frac{\partial \epsilon^2}{\partial \hat{\beta}} = \sum_{n=1}^{N} 2\left[\hat{\alpha}(\hat{r}_n - \hat{\mu}_r) + \hat{\beta}(\hat{c}_n - \hat{\mu}_c)\right](\hat{c}_n - \hat{\mu}_c) - \lambda(2\hat{\beta})N = 0$$

$$\longrightarrow \begin{pmatrix} \hat{\mu}_{rr} & \hat{\mu}_{rc} \\ \hat{\mu}_{rc} & \hat{\mu}_{cc} \end{pmatrix} \begin{pmatrix} \hat{\alpha} \\ \hat{\beta} \end{pmatrix} = \lambda \begin{pmatrix} \hat{\alpha} \\ \hat{\beta} \end{pmatrix}$$

$$\left[\begin{pmatrix} \hat{\mu}_{rr} & \hat{\mu}_{rc} \\ \hat{\mu}_{rc} & \hat{\mu}_{cc} \end{pmatrix} - \hat{\lambda}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right]\begin{pmatrix} \hat{\alpha} \\ \hat{\beta} \end{pmatrix} = 0 \longrightarrow \begin{vmatrix} \hat{\mu}_{rr} - \hat{\lambda} & \hat{\mu}_{rc} \\ \hat{\mu}_{rc} & \hat{\mu}_{cc} - \hat{\lambda} \end{vmatrix} = 0$$

$$\longrightarrow \quad \hat{\lambda} = \frac{(\hat{\mu}_{rr} + \hat{\mu}_{cc}) \pm \sqrt{(\hat{\mu}_{rr} + \hat{\mu}_{cc})^2 - 4(\hat{\mu}_{rr}\hat{\mu}_{cc} - \hat{\mu}_{rc}^2)}}{2}$$

$$= \frac{(\hat{\mu}_{rr} + \hat{\mu}_{cc}) \pm \sqrt{(\hat{\mu}_{rr} - \hat{\mu}_{cc})^2 + 4\hat{\mu}_{rc}^2}}{2}$$

$$\longrightarrow \quad \begin{pmatrix} \hat{\alpha} \\ \hat{\beta} \end{pmatrix} = \frac{1}{\sqrt{\hat{\mu}_{rc}^2 + (\lambda - \hat{\mu}_{rr})^2}} \begin{pmatrix} \hat{\mu}_{rc} \\ \hat{\lambda} - \hat{\mu}_{rr} \end{pmatrix} = \frac{1}{\sqrt{\hat{\sigma}_{xy}^2 + (\hat{\mu}_{cc} - \lambda)^2}} \begin{pmatrix} \hat{\mu}_{cc} - \hat{\lambda} \\ -\hat{\mu}_{rc} \end{pmatrix}$$

11.7.2 PRINCIPAL-AXIS CURVE FIT

- The principal-axis curve fit is obviously a generalization of the line-fitting idea.
- The curve $0 = f(r,c) = \sum_{k=1}^{K} \alpha_k f_k(r,c) \quad , \quad \sum_{k=1}^{K} \alpha_k^2 = 1$
- e.g. conics : $ax^2 + bxy + cy^2 + dx + ey + f = 0$

$$f_1(x,y) = x^2, f_2(x,y) = xy, f_3(x,y) = y^2$$
$$f_4(x,y) = x, f_5(x,y) = y, f_6(x,y) = 1$$

11.7.2 PRINCIPAL-AXIS CURVE FIT

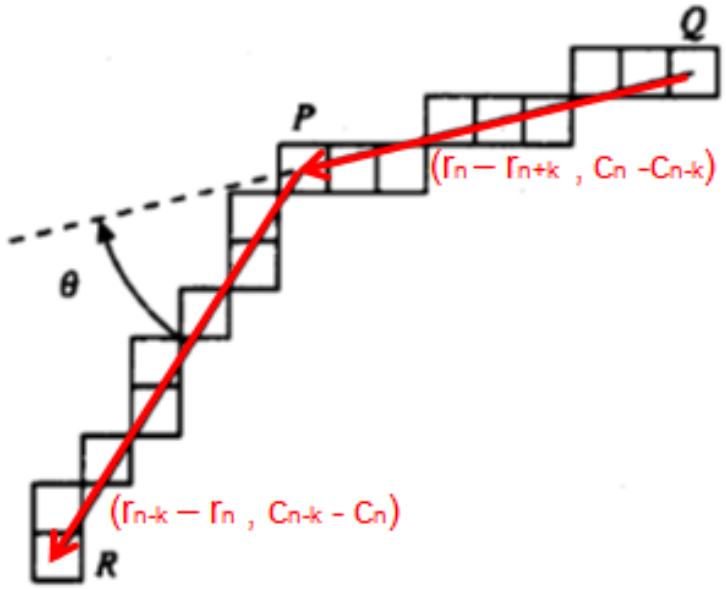
- The curve $0 = f(r,c) = \sum_{k=1}^{K} \alpha_k f_k(r,c) \quad , \quad \sum_{k=1}^{K} \alpha_k^2 = 1$
- minimize $\sum_{n=1}^{N} f(\hat{r}_n, \hat{c}_n)^2 \quad \text{subject to} \quad \sum_{k=1}^{K} \hat{\alpha}_k^2 = 1$

$$\rightarrow \min \ \varepsilon^2 = \sum_{n=1}^{N} f(\hat{r}_n, \hat{c}_n) - \lambda\left(\sum_{k=1}^{K} \alpha_k^2 - 1\right)$$

$$= \sum_{n=1}^{N}\left[\sum_{k=1}^{K} \alpha_k f_k(r_n, c_n)\right] - \lambda\left(\sum_{k=1}^{K} \alpha_k^2 - 1\right)$$

11.8 REGION-OF-SUPPORT DETERMINATION

- Region of support too large: fine features smoothed out
- Region of support too small: many corner points or dominant points produced

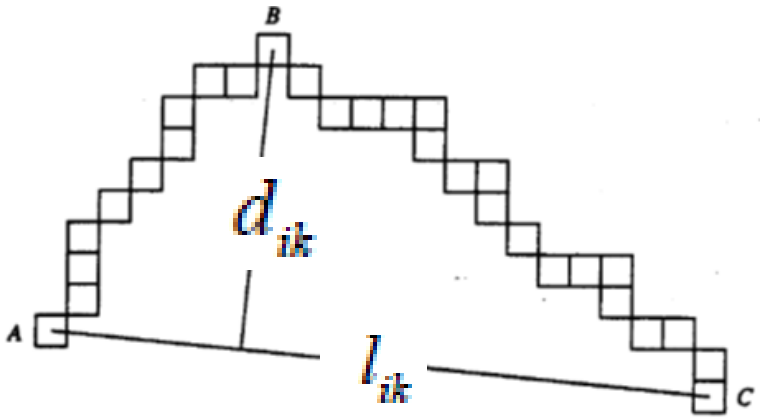


11.8 REGION-OF-SUPPORT DETERMINATION

- Teh and Chin
  - Calculate d_{ik} , l_{ik} until

$$1)\ l_{ik} \geqslant l_{i,k+1}$$

$$2)\begin{cases} \dfrac{d_{ik}}{l_{ik}} \geqslant \dfrac{d_{i,k+1}}{l_{i,k+1}} & d_{ik} > 0 \\ \dfrac{d_{ik}}{l_{ik}} \leqslant \dfrac{d_{i,k+1}}{l_{i,k+1}} & d_{ik} < 0 \end{cases}$$



  - Region of support: $D(p_i) = \{p_{i-k}, \cdots, p_{i-1}, p_i, p_{i+1}, \cdots, p_{i+k}\}$

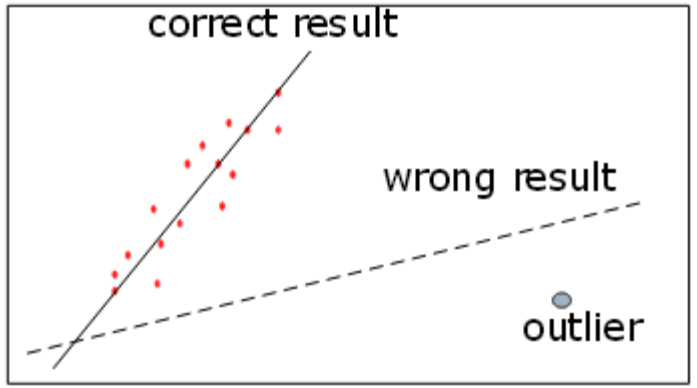
11.9 ROBUST LINE FITTING

- Fit insensitive to a few outlier points
- Give a least-squares formulation first and then modify it to make it robust.

- $$\begin{bmatrix}\hat{\alpha}\\ \hat{\beta}\\ \hat{\gamma}\end{bmatrix} = \vec{p} = \arg\min_{\vec{p}} \|\vec{\varepsilon}\|^2 = \vec{\varepsilon}^t\vec{\varepsilon} \quad \text{subject to } \|\vec{p}\| = 1$$

$$A = \begin{pmatrix} \hat{r}_1 & \hat{c}_1 & 1 \\ \hat{r}_2 & \hat{c}_2 & 1 \\ \dots & \dots & \dots \\ \hat{r}_N & \hat{c}_N & 1 \end{pmatrix}_{Nx3}, \quad \vec{p} = \begin{bmatrix}\hat{\alpha}\\ \hat{\beta}\\ \hat{\gamma}\end{bmatrix}$$

$$\vec{\varepsilon} = A\vec{p}$$



11.9 ROBUST LINE FITTING

- In the weighted least-squares sense:

$$\begin{bmatrix}\hat{\alpha}\\ \hat{\beta}\\ \hat{\gamma}\end{bmatrix} = \vec{p} = \arg\min_{\vec{p}} \ \vec{\varepsilon}^{t} W W \vec{\varepsilon} \qquad \text{subject to } \|\vec{p}\| = 1$$

$$W_{k+1} = \begin{pmatrix} w_1 & & \\ & \cdots & \\ & & w_N \end{pmatrix}$$

$$w_i = \begin{cases} [1-(\frac{e_i}{cZ})^2] & \text{if } (\frac{e_i}{cZ})^2 < 1 \\ 0 & \text{otherwise} \end{cases} \quad , Z: \text{median of} |e_i| \ , c: \text{constant}$$

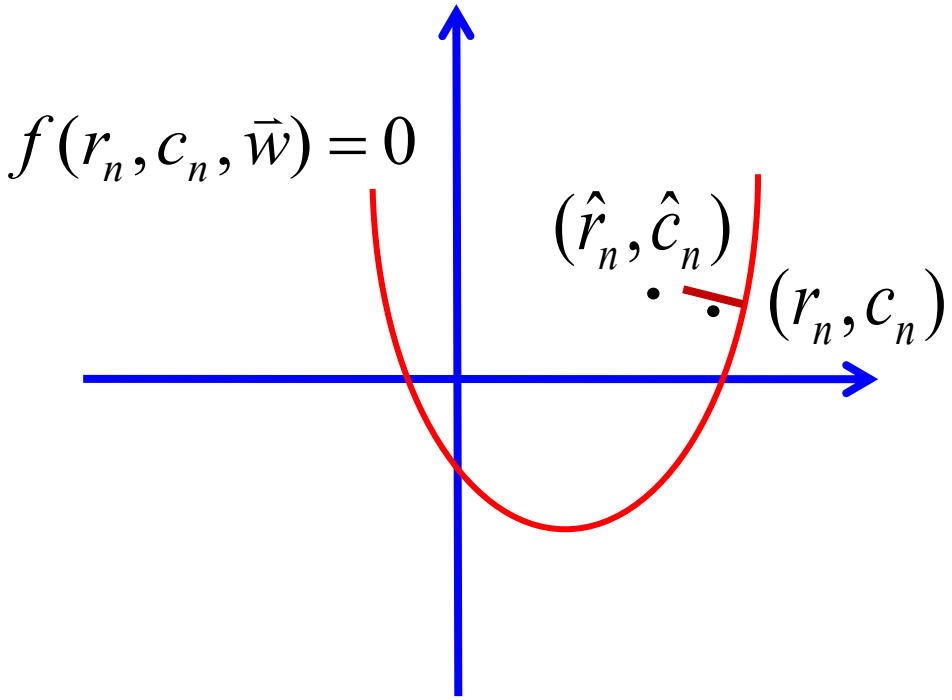
11.10 LEAST-SQUARES CURVE FITTING

- Determine the parameters of the curve that **minimize the sum of the squared distances** between the noisy observed points and the curve.

- $$\min \sum_{n=1}^{N} (\hat{r}_n - r_n)^2 + (\hat{c}_n - c_n)^2 \quad \text{subject to} \quad f(r_n, c_n, \vec{w}) = 0$$

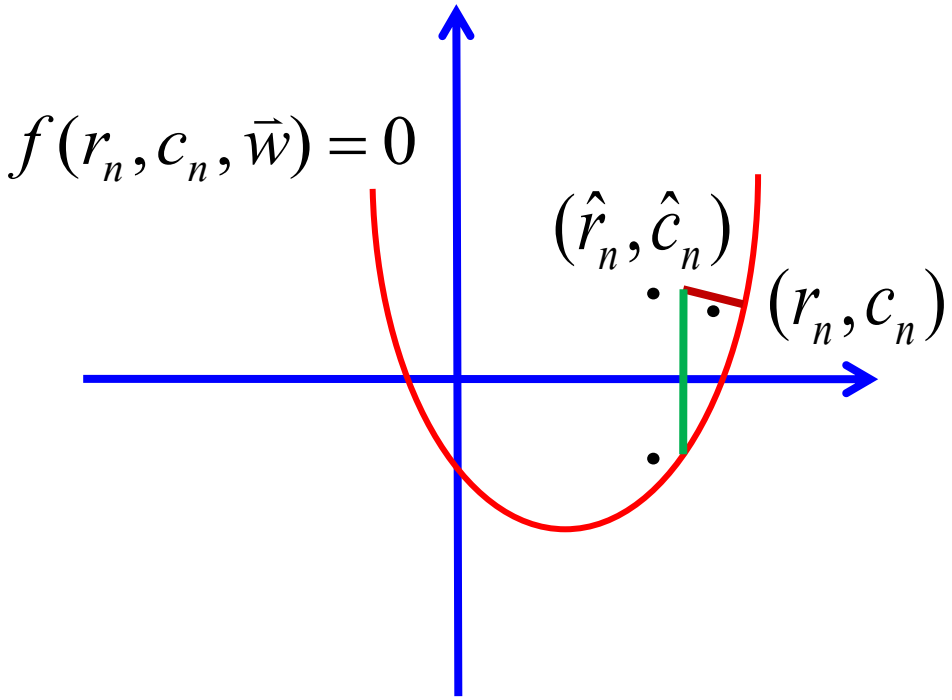
11.10 LEAST-SQUARES CURVE FITTING

- $\min \sum_{n=1}^{N} (\hat{r}_n - r_n)^2 + (\hat{c}_n - c_n)^2$ subject to $f(r_n, c_n, \vec{w}) = 0$



11.10 LEAST-SQUARES CURVE FITTING

- $$\min \sum_{n=1}^{N} f(\hat{r}_n, \hat{c}_n, \vec{w})^2$$



11.10 LEAST-SQUARES CURVE FITTING

$$\text{we define } \epsilon^2 = (r - r_o)^2 + (c - c_o)^2 - 2\lambda f(r,c)$$

$$\frac{\partial \epsilon^2}{\partial r} = 2(r - r_o) - 2\lambda \frac{\partial f}{\partial r}(r,c) = 0$$

$$\frac{\partial \epsilon^2}{\partial c} = 2(c - c_o) - 2\lambda \frac{\partial f}{\partial c}(r,c) = 0 \longrightarrow \begin{pmatrix} r - r_o \\ c - c_o \end{pmatrix} = \lambda \begin{pmatrix} \frac{\partial f}{\partial r}(r,c) \\ \frac{\partial f}{\partial c}(r,c) \end{pmatrix} = \lambda \begin{pmatrix} \frac{\partial f}{\partial r}(r_o,c_o) \\ \frac{\partial f}{\partial c}(r_o,c_o) \end{pmatrix}$$

$$\frac{\partial \epsilon^2}{\partial \lambda} = -2f(r,c) = 0 \longrightarrow 0 = f(r,c) = f(r_o,c_o) + (r - r_o)\frac{\partial f}{\partial r}(r_o,c_o) + (c - c_o)\frac{\partial f}{\partial c}(r_o,c_o)$$

$$\longrightarrow \lambda = \frac{-f(r_o,c_o)}{\frac{\partial f}{\partial r}(r_o,c_o)^2 + \frac{\partial f}{\partial c}(r_o,c_o)^2}$$

11.10 LEAST-SQUARES CURVE FITTING

- Distance d between $(\hat{r}_n, \hat{c}_n)$ and the curve :

$$d = \frac{\left| f(\hat{r}_n, \hat{c}_n, \vec{w}) \right|}{\sqrt{\left[\frac{\partial}{\partial r} f(\hat{r}_n, \hat{c}_n, \vec{w}) \right]^2 + \left[\frac{\partial}{\partial c} f(\hat{r}_n, \hat{c}_n, \vec{w}) \right]^2}}$$

- $$\min \sum_{n=1}^{N} (\hat{r}_n - r_n)^2 + (\hat{c}_n - c_n)^2 \quad \text{subject to} \quad f(r_n, c_n, \vec{w}) = 0$$

$$\Leftrightarrow \quad \min \varepsilon^2(\vec{w}) = \sum_{n=1}^{N} \frac{f(\hat{r}_n, \hat{c}_n, \vec{w})^2}{\left[\frac{\partial}{\partial r} f(\hat{r}_n, \hat{c}_n, \vec{w}) \right]^2 + \left[\frac{\partial}{\partial c} f(\hat{r}_n, \hat{c}_n, \vec{w}) \right]^2}$$

11.10.1 GRADIENT DESCENT

- First-order iterative technique in minimization problem
- Initial value : $\vec{w}_0$

  (t+1)-th iteration → $\vec{w}_{t+1} = \vec{w}_t + \Delta\vec{w}$
- First-order Taylor series expansion around $\vec{w}_t$

$$\varepsilon^2(\vec{w}_{t+1}) = \varepsilon^2(\vec{w}_t + \Delta\vec{w}) = \varepsilon^2(\vec{w}_t) + (\Delta\vec{w})^T \nabla\varepsilon^2(\vec{w}_t)$$

- $\Delta\vec{w}$ should be in the negative gradient direction

$$\Delta\vec{w} = -\beta\varepsilon^2(\vec{w}_t)\frac{\nabla\varepsilon^2(\vec{w}_t)}{\left\|\nabla\varepsilon^2(\vec{w}_t)\right\|^2 + \alpha^2}$$

11.10.2 NEWTON METHOD

- Second-order iterative technique in minimization problem
- Second-order Taylor series expansion around $\vec{w}_t$

$$\varepsilon^2(\vec{w}_t + \Delta\vec{w}) = \varepsilon^2(\vec{w}_t) + (\Delta\vec{w})^T \nabla\varepsilon^2(\vec{w}_t) + \frac{1}{2}(\Delta\vec{w})^T H \Delta\vec{w}$$

$H = H(w_t)$ → second-order partial derivatives, Hessian

- Take partial derivatives to zero with respect to $\Delta\vec{w}$

$$H(\Delta\vec{w}) + \nabla\varepsilon^2(\vec{w}_t) = 0 \quad \Rightarrow \quad \Delta\vec{w} = -H^{-1}\nabla\varepsilon^2(\vec{w}_t)$$

11.10.4 FITTING TO A CIRCLE

- Circle : $f(r,c,\vec{w})=0$

$$f(r,c,\vec{w})=f(r,c,a,b,R)=(r-a)^2-(c-b)^2-R^2$$

- $$\min \varepsilon^2(\vec{w})=\sum_{n=1}^{N}\frac{f(\hat{r}_n,\hat{c}_n,\vec{w})^2}{\left[\frac{\partial}{\partial r}f(\hat{r}_n,\hat{c}_n,\vec{w})\right]^2+\left[\frac{\partial}{\partial c}f(\hat{r}_n,\hat{c}_n,\vec{w})\right]^2}$$

- $$\frac{\partial}{\partial r}f=2(r-a)\ ,\quad \frac{\partial}{\partial c}f=2(c-b)$$

$$\min \varepsilon^2(a,b,R)=\sum_{n=1}^{N}\frac{[(\hat{r}_n-a)^2+(c_n-b)^2-R^2]^2}{4[(\hat{r}_n-a)^2+(c_n-b)^2]}$$

11.10.4 FITTING TO A CIRCLE

- $$\min \varepsilon^2(a,b,R) = \sum_{n=1}^{N} \frac{[(\hat{r}_n - a)^2 + (c_n - b)^2 - R^2]^2}{4[(\hat{r}_n - a)^2 + (c_n - b)^2]}$$

$$\Longrightarrow \min \varepsilon^2(a,b,R) = \sum_{n=1}^{N} [\sqrt{(\hat{r}_n - a)^2 + (c_n - b)^2} - R]^2$$

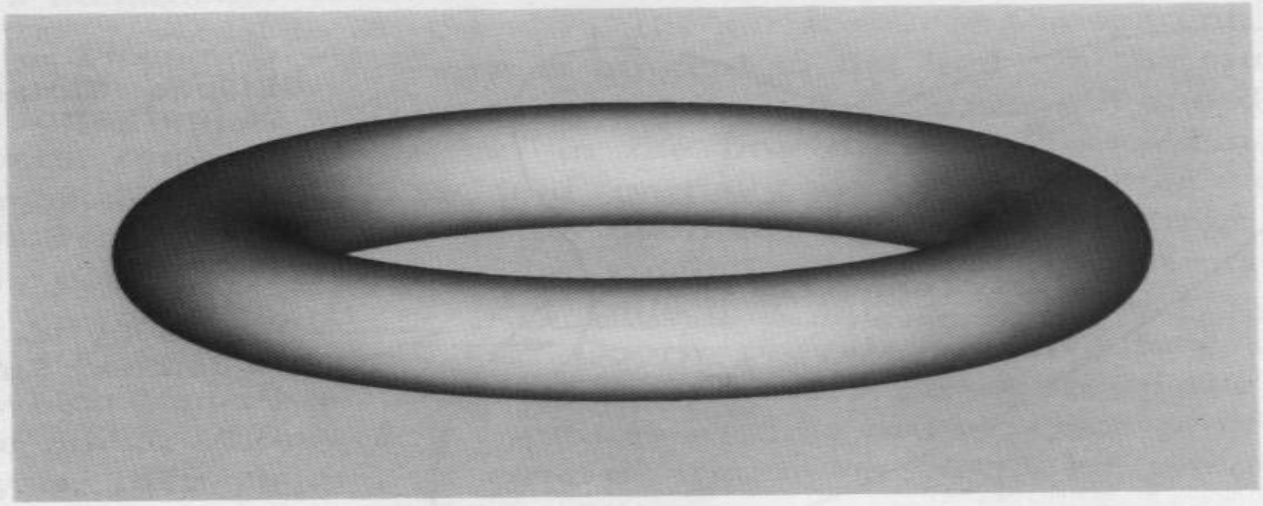
11.10.6 FITTING TO A CONIC

- In conic :

$$f(r,c,\vec{w}) = f(a,b,A,B,C) = A(r-a)^2 + 2B(r-a)(c-b) + C(c-b)^2 - 1$$

- $$\frac{\partial}{\partial r} f = 2A(r-a) + 2B(c-b)$$
  $$\frac{\partial}{\partial c} f = 2B(r-a) + 2C(c-b)$$

$$\varepsilon^2 = \frac{1}{4}\sum_{n=1}^{N} \frac{[A(\hat{r}_n - a)^2 + 2B(\hat{r}_n - a)(\hat{c}_n - b) + C(\hat{c}_n - b)^2 - 1]^2}{[A(\hat{r}_n - a) + B(\hat{c}_n - b)]^2 + [B(\hat{r}_n - a) + C(\hat{c}_n - b)]^2}$$



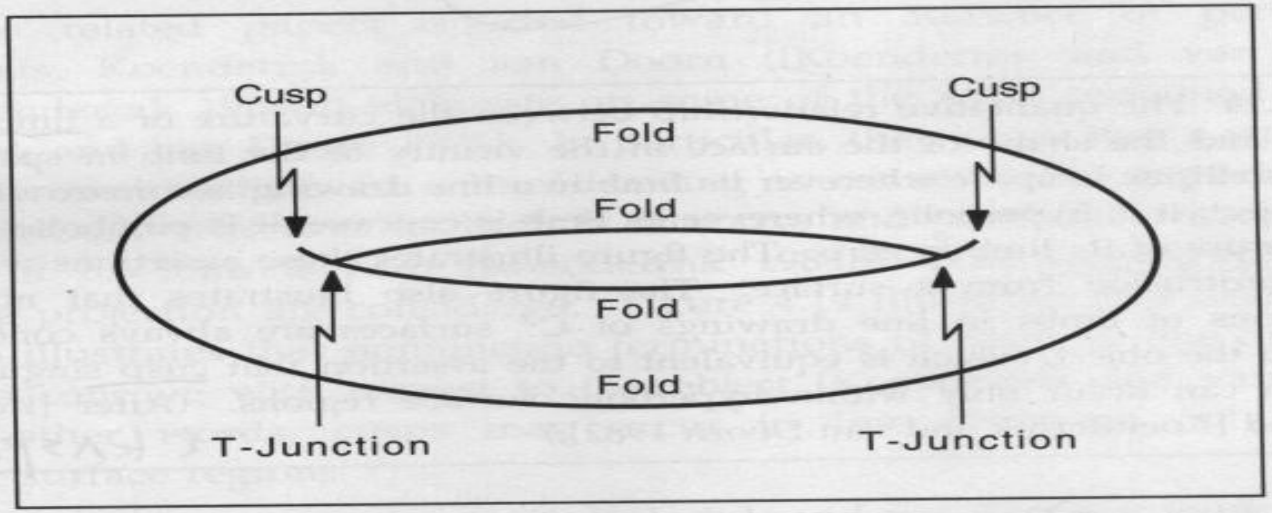
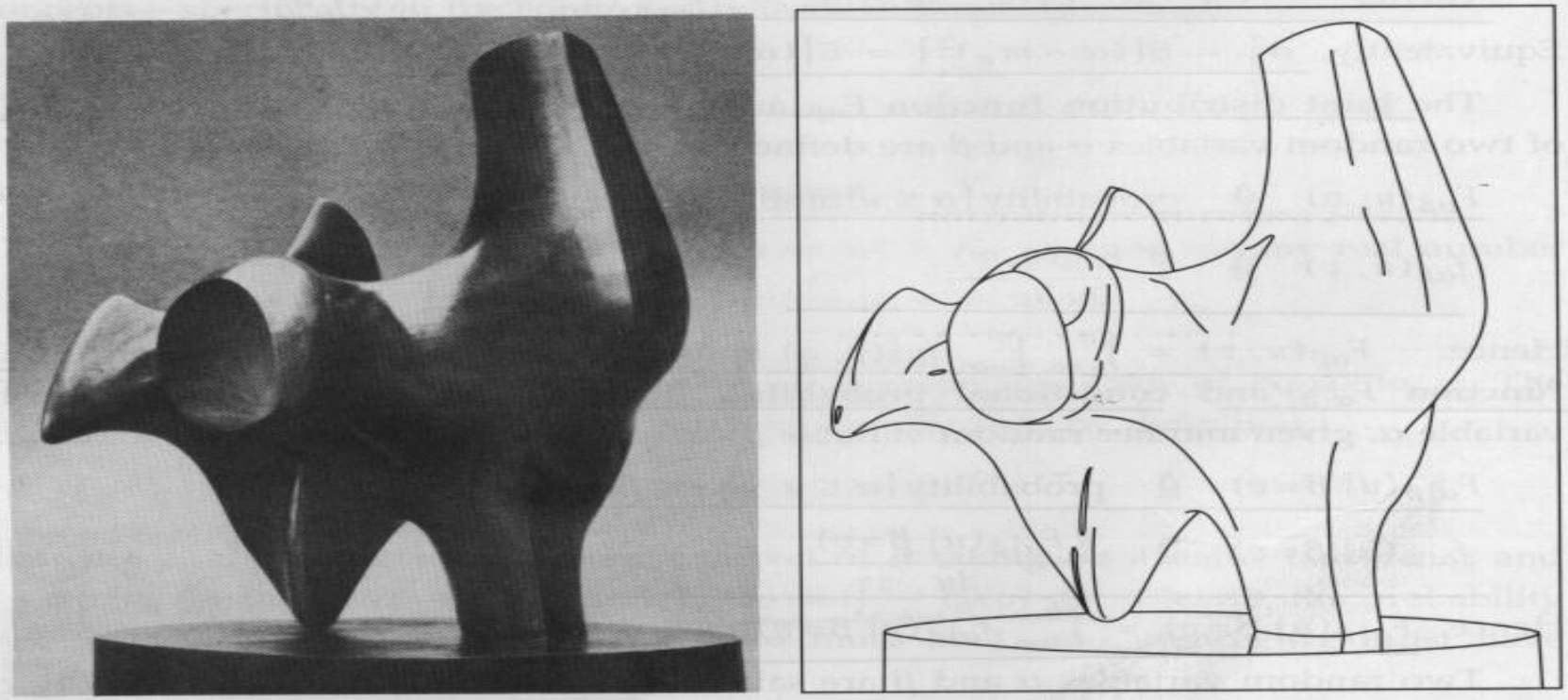


Figure 4.15 Oblique view of a torus exhibiting folds, cusps, and T-junctions. Folds, cusps, and T-junctions are the canonical singularities of the projection of a C^3 surface under general viewpoint (see Figure 4.13). A *torus* is a (doughnut-shaped) surface that we can generate by revolving a circle about a straight line that lies within the plane of the circle but that does not intersect the circle. In the figure, a synthetic image of a torus is shown above its line drawing.

considering here. The loci of the instances of singular projection in a line



['ɑrtʃɚ] 弓箭手

Figure 3.1 *The Archer,* working model, by Henry Moore, 1964. On the left is an image of a working model of a sculpture, and on the right is an artist's rendering of some of the salient brightness edges in the image. An *edge* in an image is an image contour across which the intensity of the image changes abruptly; an image-intensity edge may or may not correspond to a physical edge of an object. It is sometimes said that the objective of science is to describe nature economically. Analogously, one purpose of edge detection in an image is to strip away some of the redundancy of sensing—to encode and describe the information contained in the image in a form more economical than that in which the information impinges on the sensors [Attneave 1954]. (Photograph, courtesy The Henry Moore Foundation, Much Hadham, England.)

THANK YOU