# Segmentation – Part 2

Elsayed Hemayed

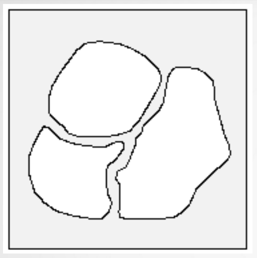
## Outline

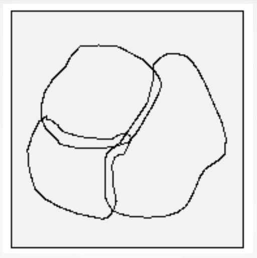
- Border Tracing
- Hough Transforms
- Region Based Segmentation
  - Region Merging
  - Region Splitting
  - Merge and Split

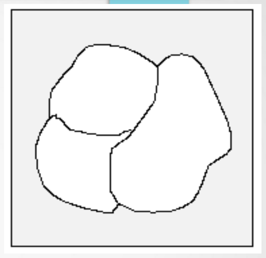
## Border Tracing

- If region border is not known but regions have been defined, borders can be detected.
- During border tracing three types of borders can be defined:
  - An inner region border is a subset of the region.
  - An outer border is not a subset of the region.
  - An extended border

## Border Types


Inner Boundary

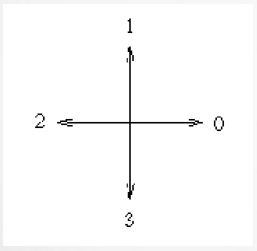

Outer Boundary

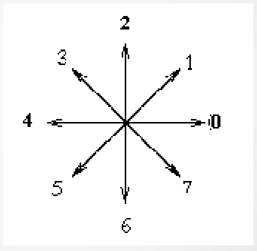

Extended Boundary

The **inner border** is always **part of a region** but the **outer border never is**. Therefore, if two regions are adjacent, they never have a common border, which causes difficulties in higher processing levels with region description, region merging, etc.

The **extended border** defines **single common border between adjacent regions**

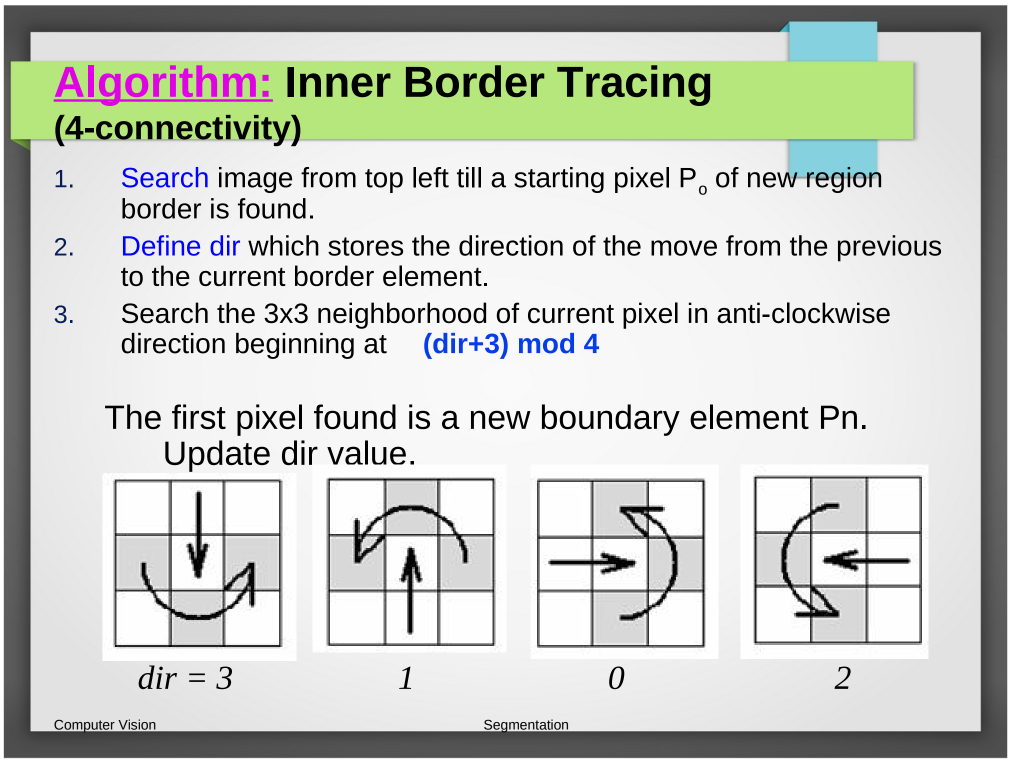
## Connectivity Types





4 - connectivity

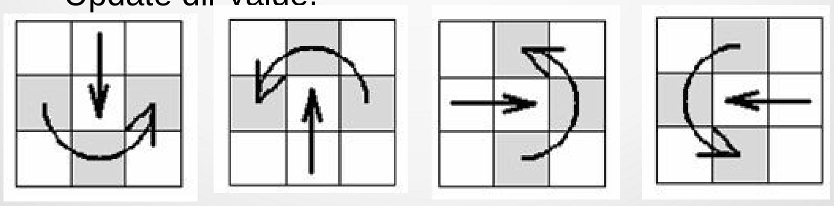
8 - connectivity



## Algorithm: Inner Border Tracing (4-connectivity)

1. Search image from top left till a starting pixel $P_o$ of new region border is found.
2. Define dir which stores the direction of the move from the previous to the current border element.
3. Search the 3x3 neighborhood of current pixel in anti-clockwise direction beginning at (dir+3) mod 4

The first pixel found is a new boundary element Pn. Update dir value.



*dir* = 3 1 0 2

In case of 8 connectivity

Dir+7 mod 8 if direction even

Dir+6 mod 8 if direction odd

## Algorithm: Inner Border Tracing (cont.)

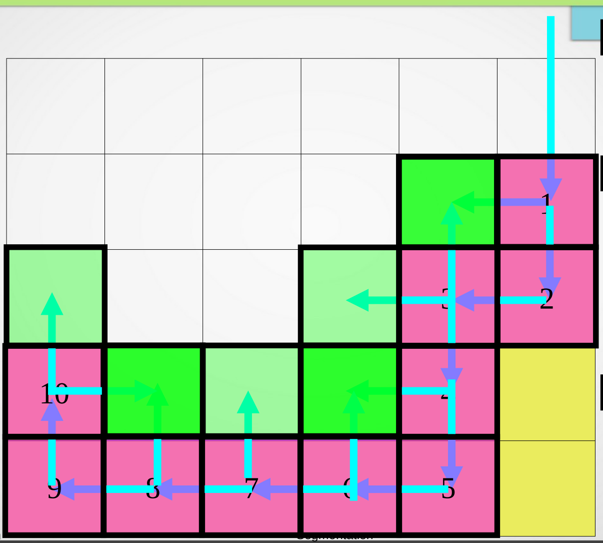
4. If the current boundary element $P_n$ is equal to the second border element $P_1$ and if the previous border element $P_{n-1}$ is equal to $P_0$ stop. Otherwise go to step 3.
5. The detected inner border is represented by $P_0$ ... $P_{n-2}$.

**Algorithm does not determine edges of region holes**!

## Algorithm: Outer Border Tracing

1. Trace inner boundary region boundary in 4-connectivity until done.
2. The outer boundary consists of **non-region pixels tested during the search process**. Any **pixels tested more than once** they are **listed more than once!**

## Demo 1: Inner and Outer Border Tracing

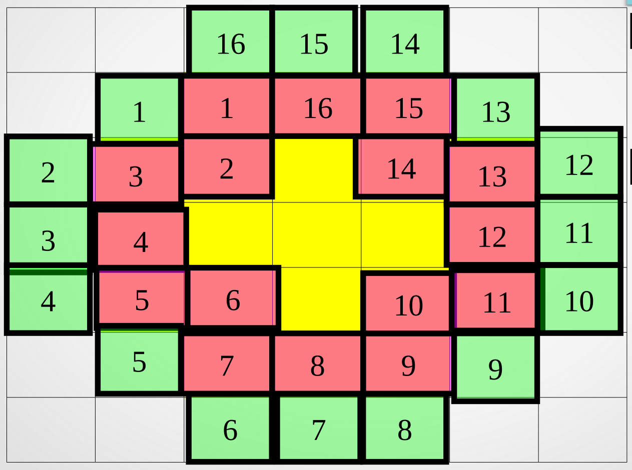


🟥 Inner Border Pixel

🟩 Outer Border Pixel detec. **once**

🟩 Outer Border Pixel detec. **twice**

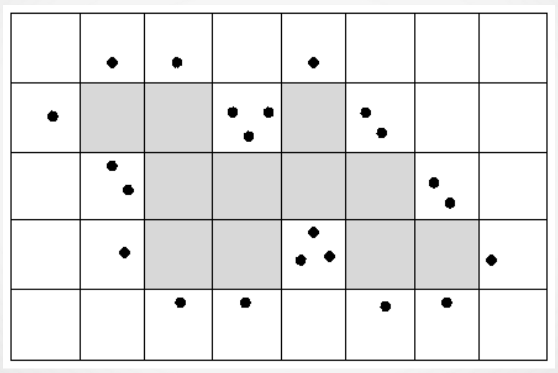
## Demo 2: Inner and Outer Border Tracing



🟥 Inner Border Pixel

🟩 Outer Border Pixel detec.

## Multiple Outer Border Example

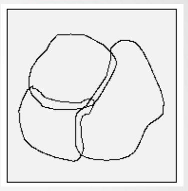


## Algorithm: Extended Border Tracing

It is constructed from the outer boundary by:

1. shifting all upper (**point 2**) outer boundary points one **pixel down and right**
2. all left points (**point 4**) → **one pixel right**
3. all right points (**point 0**) → **one pixel down**
4. all lower points (**point 6**) **remain unchanged.**

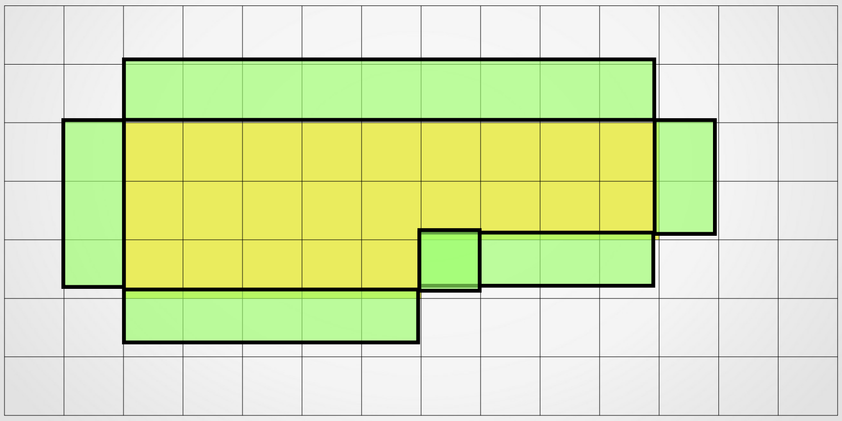
P is a point inside the region



outer boundary

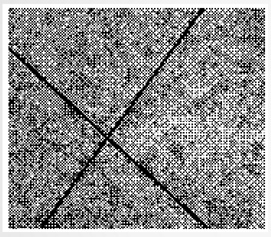
| | | |
|---|---|---|
| 3 | 2 | 1 |
| 4 | P | 0 |
| 5 | 6 | 7 |

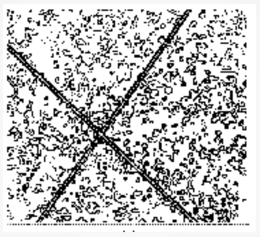
## Demo 1: Extended Border Tracing

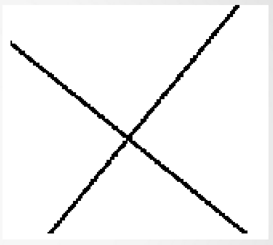


🟩 Outer Boundary

## Finding Curves With Specific Shape


**Original Image**


**Edge Image**


**Target Lines**

## Finding Lines in an image

- **Option 1:**
  - Search for the line at every possible position/orientation
  - What is the cost of this operation?
- **Option 2:**
  - Use a voting scheme:  Hough transform

## HoughTransform

- **<u>Goal:</u>**

  To find a **curve of specific shape** (line, circle, … etc) in an edge image.
  - Edges need not be connected
  - **Key Idea: Edges VOTE for the possible curve**

## Image and Hough Spaces

Equation of Line: $y = mx + c$

Find: $(m, c)$

Consider point: $(x_i, y_i)$

$$y_i = mx_i + c \quad or \quad c = -x_i m + y_i$$

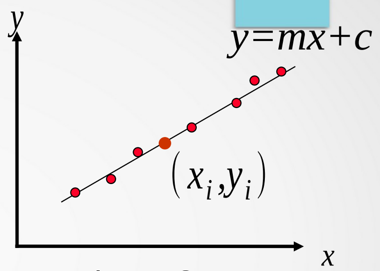
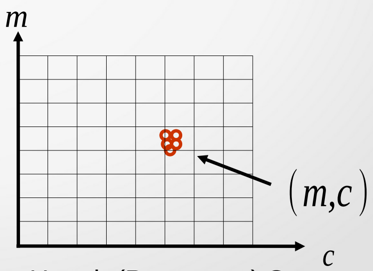


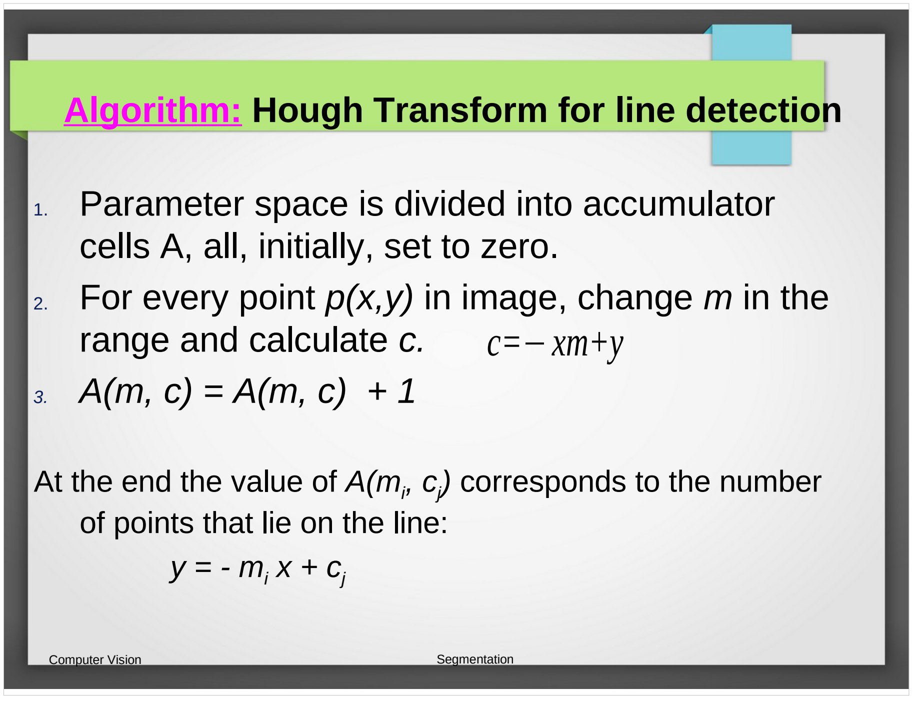
Image Space

- Every pair of points will form a line *(m, c)* and add a vote in the cell indexed by *(m, c)*

- The number of votes at *(m, c)* Corresponds to the number of points that lie on the line *(m, c)*.



Hough (Parameter) Space

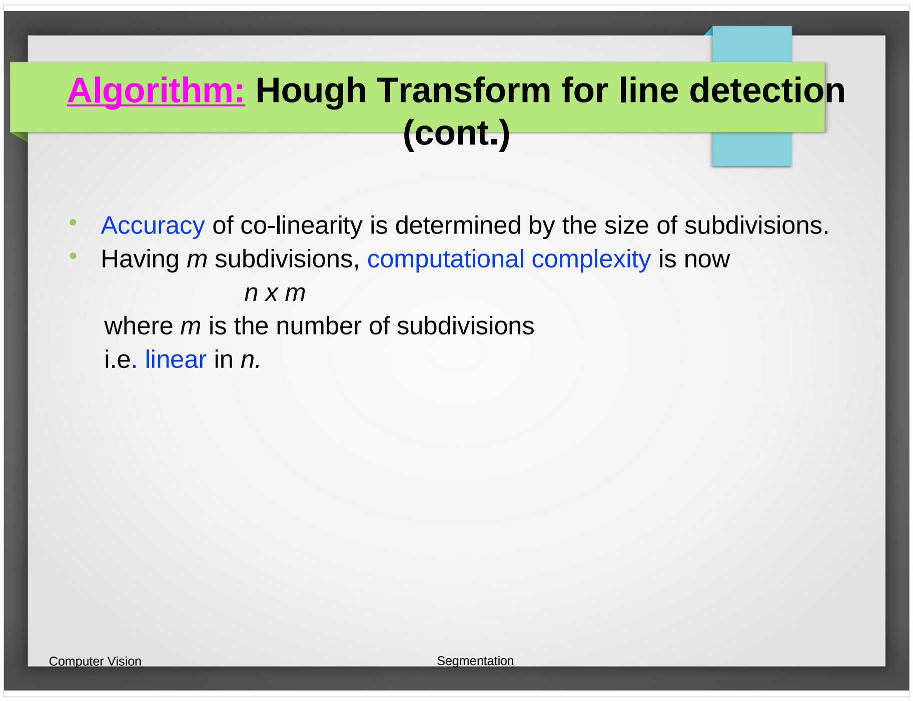


## Algorithm: Hough Transform for line detection

1. Parameter space is divided into accumulator cells A, all, initially, set to zero.
2. For every point *p(x,y)* in image, change *m* in the range and calculate *c*. $c = -xm + y$
3. $A(m, c) = A(m, c) + 1$

At the end the value of $A(m_i, c_j)$ corresponds to the number of points that lie on the line:

$$y = -m_i x + c_j$$



## Algorithm: Hough Transform for line detection (cont.)

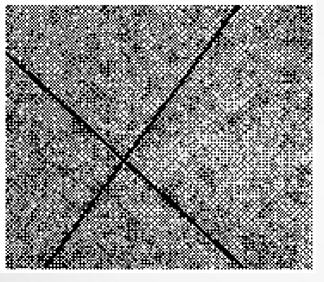
- Accuracy of co-linearity is determined by the size of subdivisions.
- Having *m* subdivisions, computational complexity is now

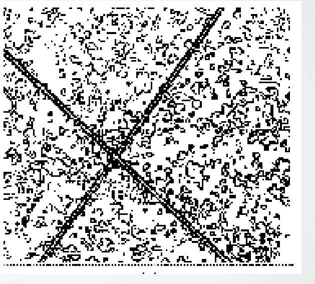
  *n x m*

  where *m* is the number of subdivisions

  i.e. linear in *n.*

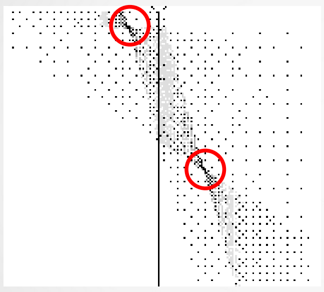
## Hough Transform Line Detection Example


Original Image

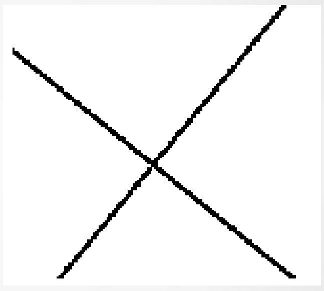

Edge Image

Notice: many non-belonging edges

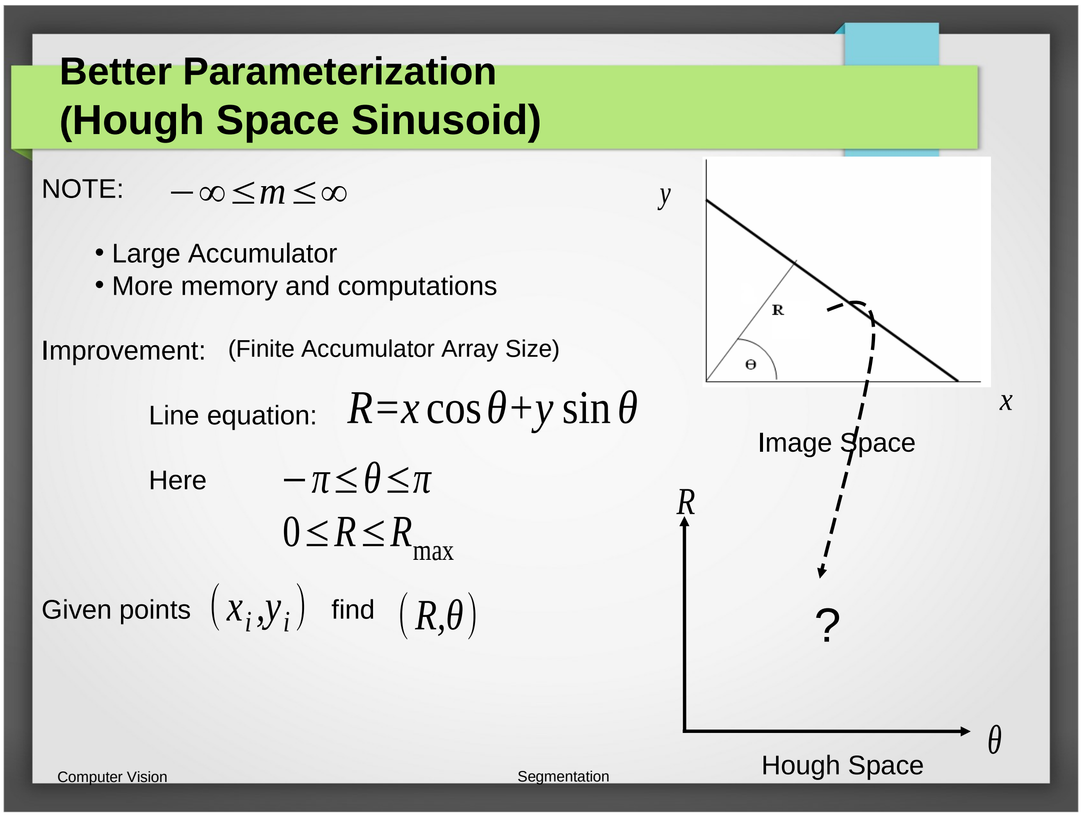
## Hough Transform Line Detection Example



**Parameter Space**



**Detected Lines**



## Better Parameterization (Hough Space Sinusoid)

NOTE: $-\infty \le m \le \infty$

- Large Accumulator
- More memory and computations

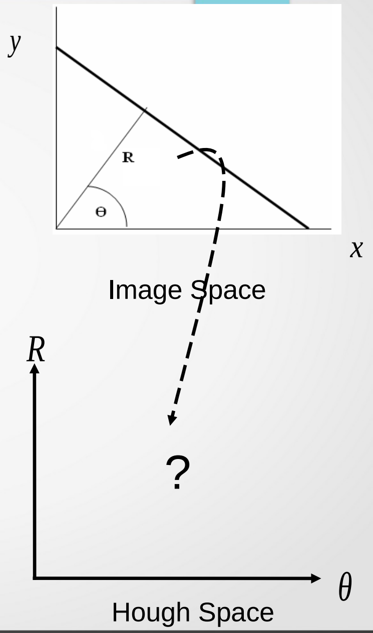
Improvement: (Finite Accumulator Array Size)

Line equation: $R = x\cos\theta + y\sin\theta$

Here $-\pi \le \theta \le \pi$

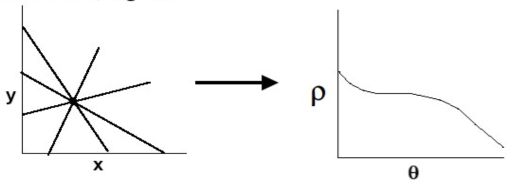
$0 \le R \le R_{\max}$

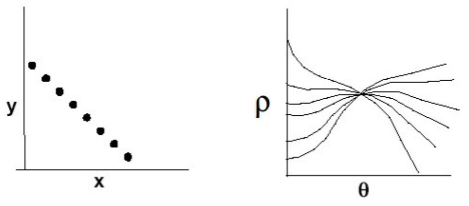
Given points $(x_i, y_i)$ find $(R, \theta)$



The range of The range of R is ±N√2 if the image is of size NxN

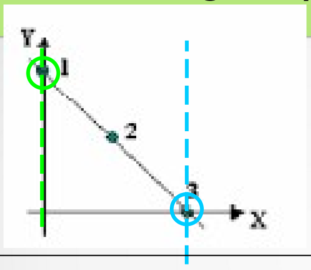
All lines passing through a point map to a sinusoidal curve in the θ-ρ (parameter) space.

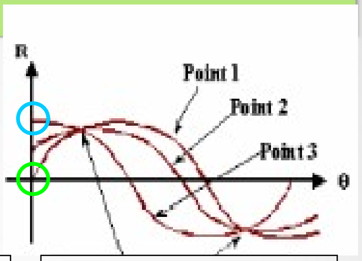




Points on the same line define curves in the parameter space that pass through a single point.

## Hough Space Sinusoid





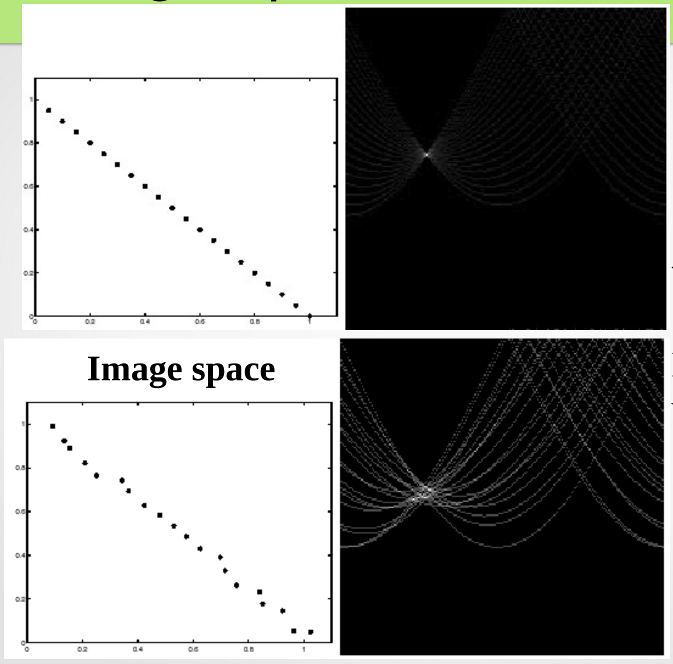
Points transform to sinusoids in ***R*-θ** space

Intersection points estimate the line equation

The Normal representation approach is similar BUT:

- •Loci are sinusoidal! i.e. each point in the ***x-y*** plane yields one sinusoidal curve on the ***R*-θ** plane and
- •***m*** co-linear points in ***x-y*** plane yield ***m*** sinusoidal curves in the ***R*-θ** plane intersecting at a certain ***R*** and **θ** corresponding to the line connecting them.
- •Range of **θ** is 0 to π

## Hough Space Sinusoid Example



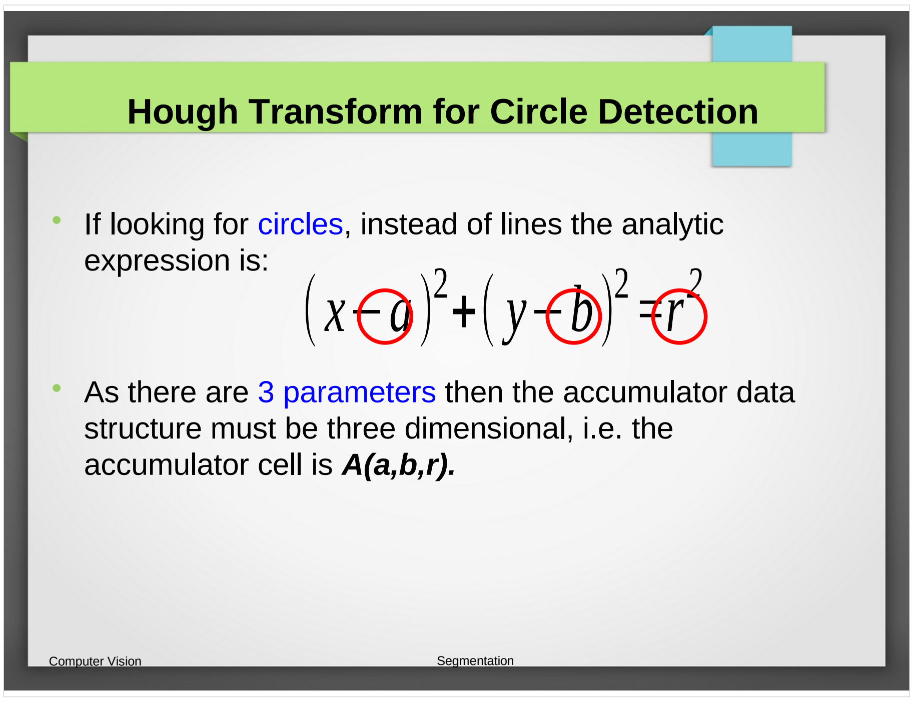
**Votes**

Horizontal axis is θ, vertical is *R*.

Figure 15.1, top half. Note that most points in the vote array are very dark, because they get only one vote.

## Mechanics of the Hough Transform

- Difficulties
  - how big should the cells be? (too big, and we merge quite different lines; too small, and noise causes lines to be missed)
- How many lines?
  - Count the peaks in the Hough array
  - Treat adjacent peaks as a single peak
- Which points belong to each line?
  - Search for points close to the line
  - Solve again for line and iterate

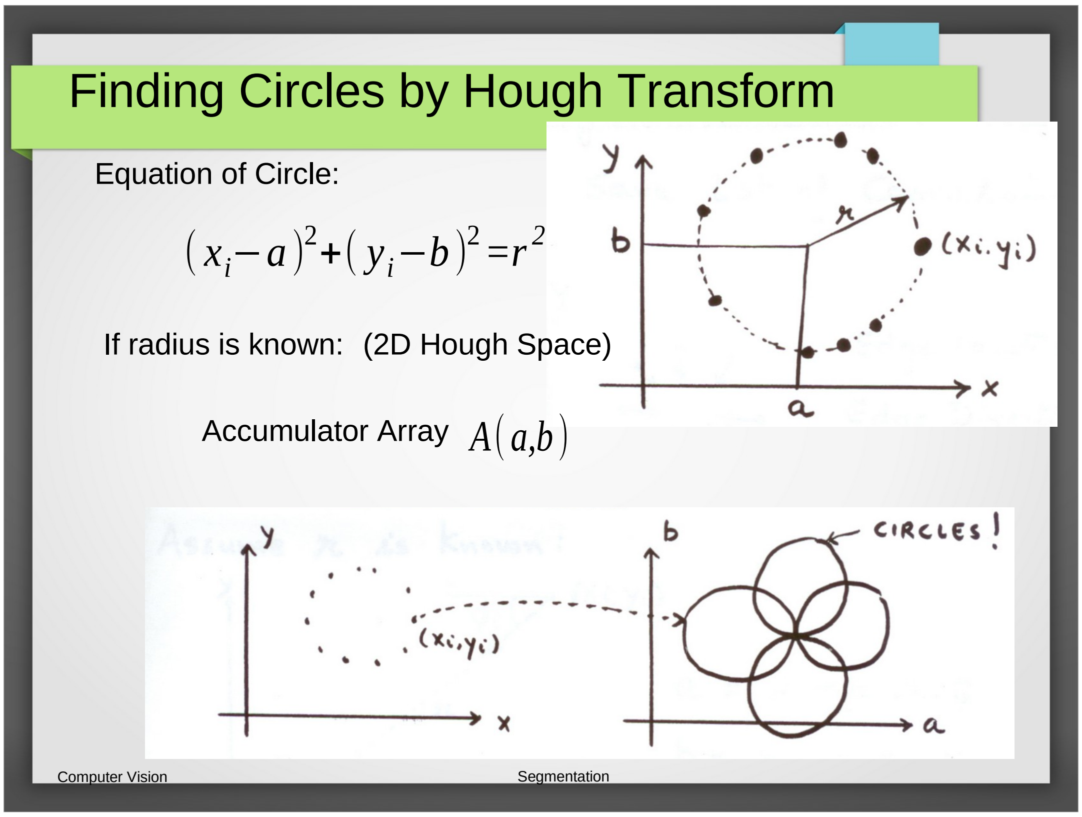


## Hough Transform for Circle Detection

- If looking for circles, instead of lines the analytic expression is:

$$(x-\boxed{a})^2+(y-\boxed{b})^2=\boxed{r}^2$$

- As there are 3 parameters then the accumulator data structure must be three dimensional, i.e. the accumulator cell is ***A(a,b,r).***



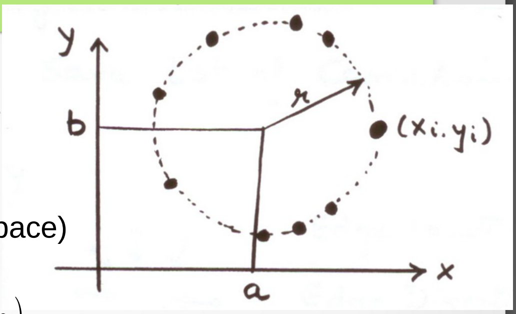
## Finding Circles by Hough Transform

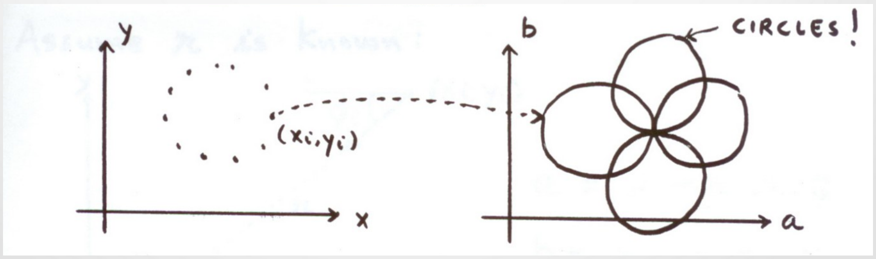
Equation of Circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

If radius is known: (2D Hough Space)

Accumulator Array $A(a,b)$





Computer Vision Segmentation

The locus of (a, b) points in the parameter space fall on a circle of radius R centered at (x, y). The true center point will be common to all parameter circles, and can be found with a Hough accumulation array.

x = a + Rcosθ

y = b + Rsinθ

When the θ varies from 0 to 360, a complete circle of radius R is generated.

So, every point in the xy space will be equivalent to a circle in the ab space (R isn't a parameter, we already know it). This is because on rearranging the equations, we get:

$a = x_1$ - Rcosθ

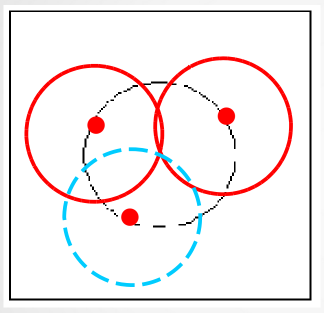
$b = y_1$ - Rsinθ

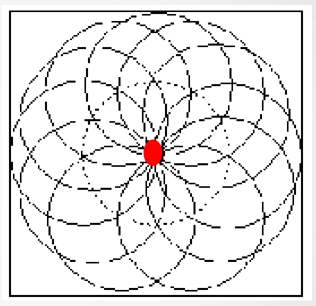
for a particular point ($x_1$, $y_1$). And θ sweeps from 0 to 360 degrees.

## Algorithm: Hough Transform for Circle Detection

- Again the Hough transform is applied to each point whose edge magnitude exceeds a certain threshold.
- The processing results correspond to points of local maxima of accumulator cells in the parameter space ***a,b*** of the center point of the circle and the radius ***r***.
- If the length of the radius of the circle searched is known then we can deal with a 2-dimensional parameter space (for center point coordinates ***a,b***).

## Demo:Hough Transform

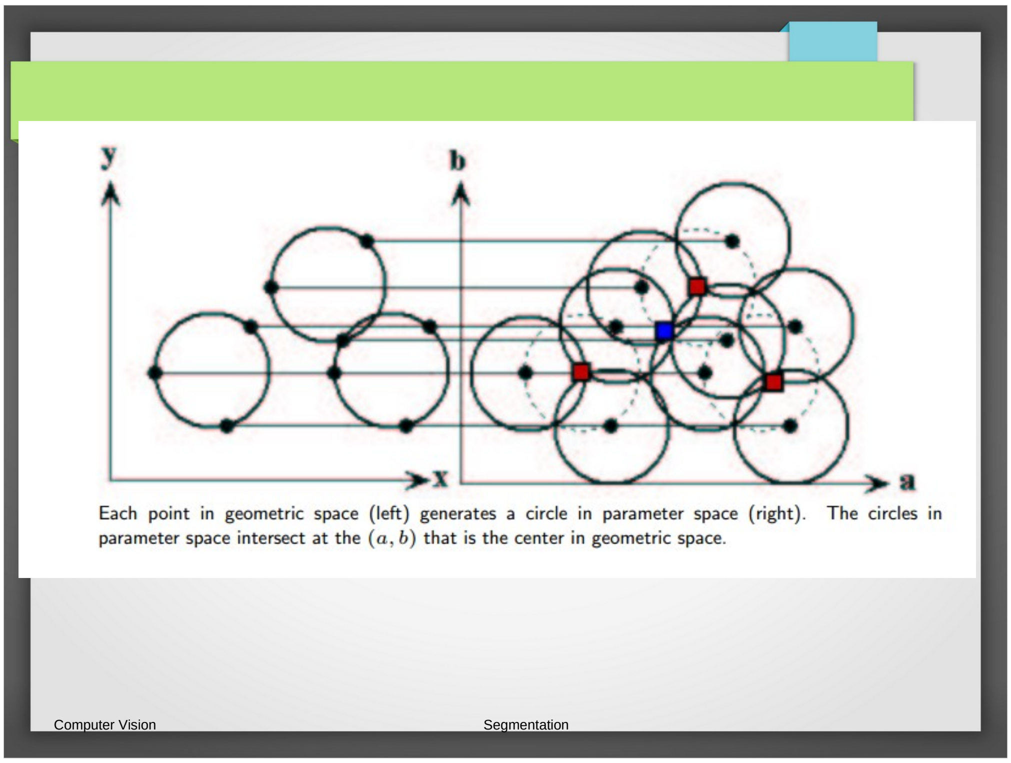


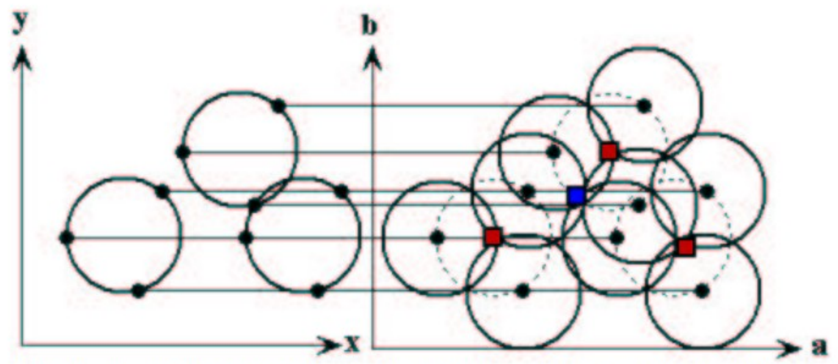


— Original Circle

● point on circle

O Intersection of circles specifies center of original circle





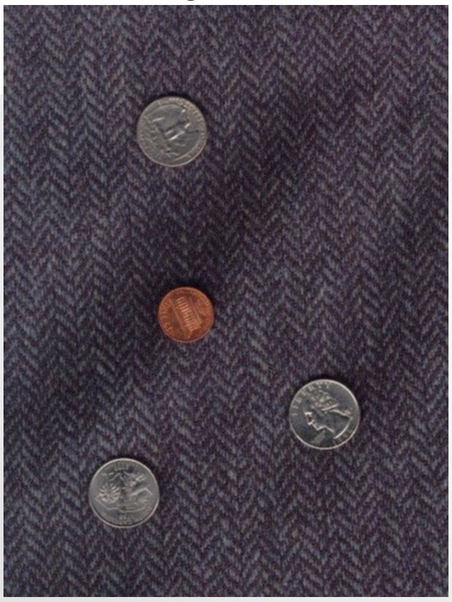
Each point in geometric space (left) generates a circle in parameter space (right). The circles in parameter space intersect at the $(a, b)$ that is the center in geometric space.

Overlap of circles can cause spurious centers to also be found, such as at the blue cell. Spurious circles can be removed by matching to circles in the original image.

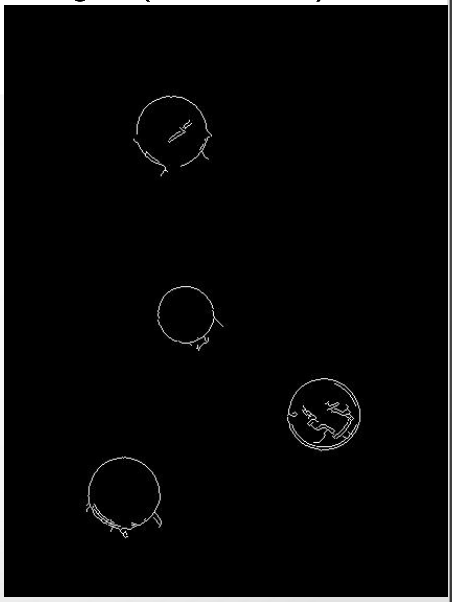
Each point in geometric space (left) generates a circle in parameter space (right). The circles in parameter space intersect at the (a, b) that is the center in geometric space.

## Finding Coins

Original

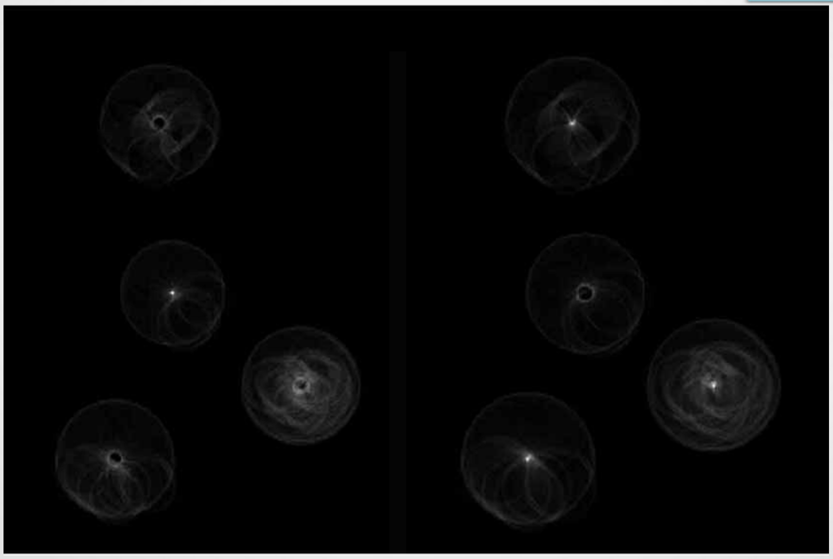


Edges (note noise)

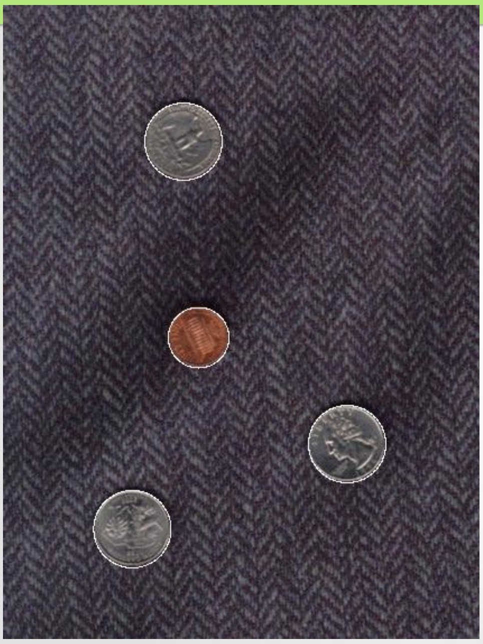


## Hough Space For Coins

Penny Quarters



## Detected Coins



Note that because the quarters and penny are different sizes, a different Hough transform (with separate accumulators) was used for each circle size.

Coin finding sample images and Matlab code from: Vivek Kwatra at http://www.math.tau.ac.il/~turkel/notes/hough4.html

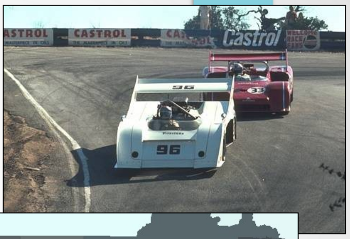
HoughCoin Matlab Files

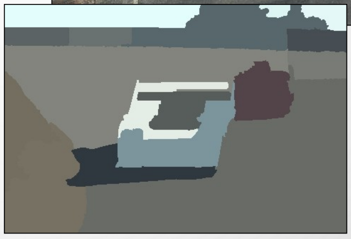
## Hough Transform: Comments

- Works on Disconnected Edges
- Effective for simple shapes (lines, circles, etc)
- Handling inaccurate edge locations:
    - Increment Patch in Accumulator rather than a single point

## Region Based Segmentation

- **Segmentation**
  - Edge detection and Thresholding not always effective.
- **Homogenous regions**
  - *Region-based segmentation.*

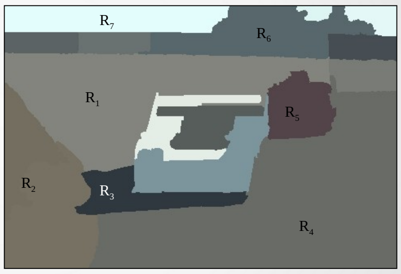


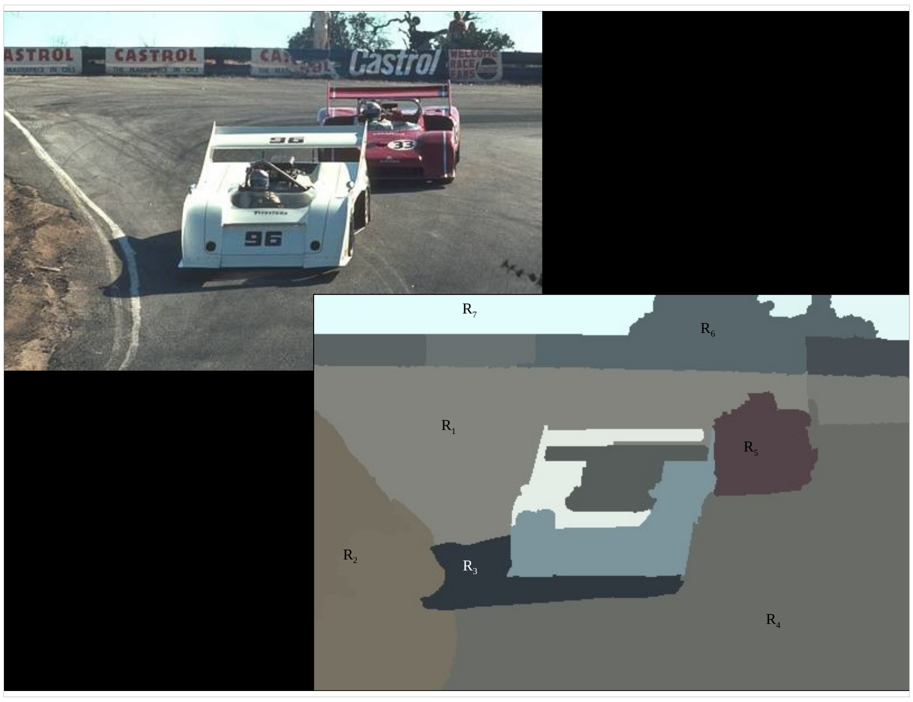


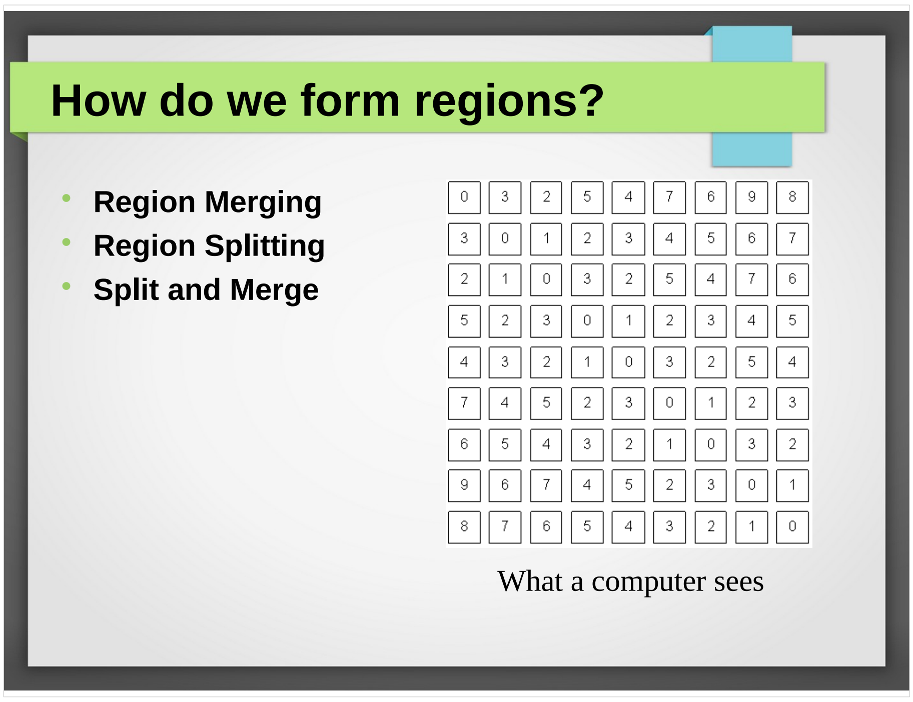
## Definitions

- Based on *sets*.
- Each image R is a set of regions $R_i$.
  - Every pixel belongs to one region.
  - One pixel can only belong to a single region.

$$R_i \, intersect R_j = \emptyset$$




CASTROL
Castrol
96
33
$R_7$
$R_6$
$R_1$
$R_5$
$R_2$
$R_3$
$R_4$

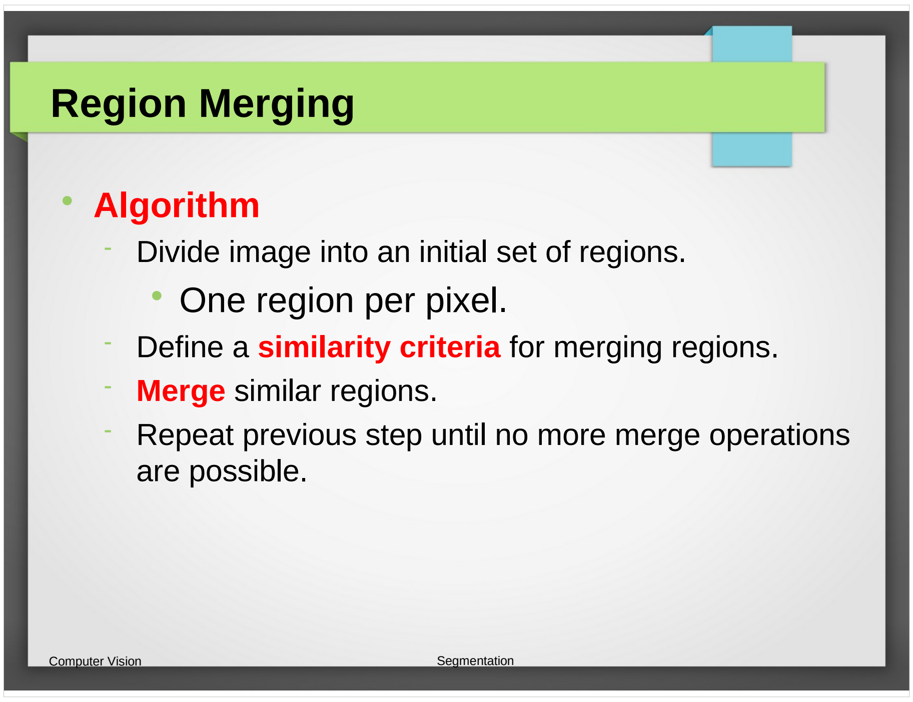


## How do we form regions?

- **Region Merging**
- **Region Splitting**
- **Split and Merge**

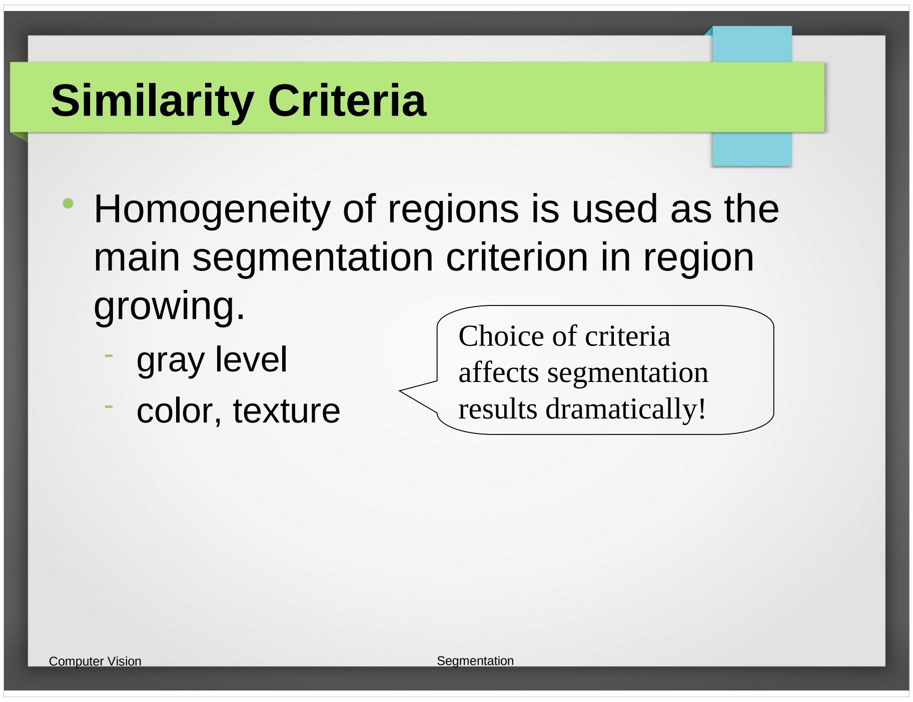
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 7 | 6 | 9 | 8 |
| 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 1 | 0 | 3 | 2 | 5 | 4 | 7 | 6 |
| 5 | 2 | 3 | 0 | 1 | 2 | 3 | 4 | 5 |
| 4 | 3 | 2 | 1 | 0 | 3 | 2 | 5 | 4 |
| 7 | 4 | 5 | 2 | 3 | 0 | 1 | 2 | 3 |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 | 3 | 2 |
| 9 | 6 | 7 | 4 | 5 | 2 | 3 | 0 | 1 |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

What a computer sees



## Region Merging

- **Algorithm**
  - Divide image into an initial set of regions.
    - One region per pixel.
  - Define a **similarity criteria** for merging regions.
  - **Merge** similar regions.
  - Repeat previous step until no more merge operations are possible.



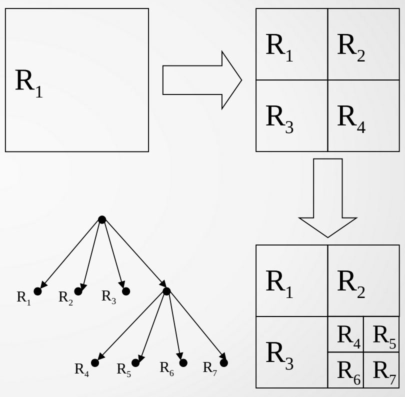
## Similarity Criteria

- Homogeneity of regions is used as the main segmentation criterion in region growing.
  - gray level
  - color, texture

Choice of criteria affects segmentation results dramatically!

## Region Splitting

- **Algorithm**
  - One initial set that includes the **whole image**.
  - **Similarity criteria**.
  - Iteratively **split** regions into sub-regions.
  - Stop when no more splittings are possible.



$R_1$
$R_1$ $R_2$ $R_3$ $R_4$
$R_1$ $R_2$ $R_3$ $R_4$ $R_5$ $R_6$ $R_7$
$R_1$ $R_2$ $R_3$ $R_4$ $R_5$ $R_6$ $R_7$

*The segmentation problem*

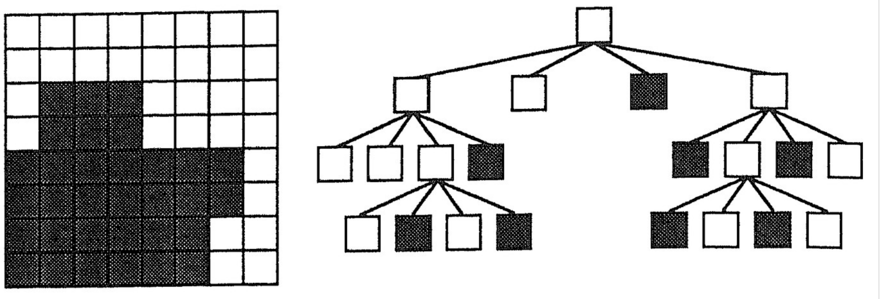
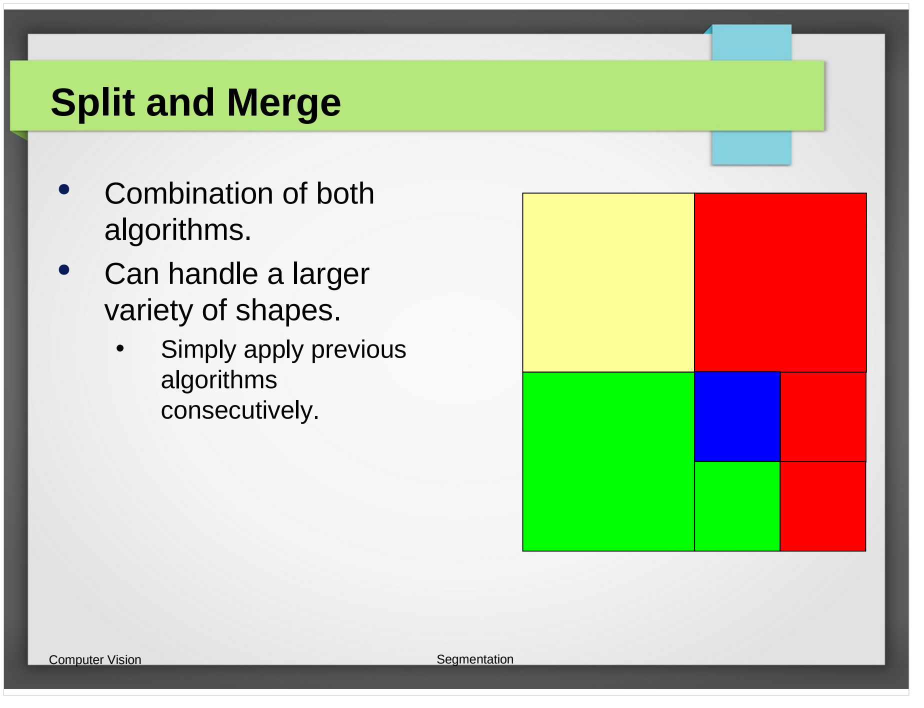
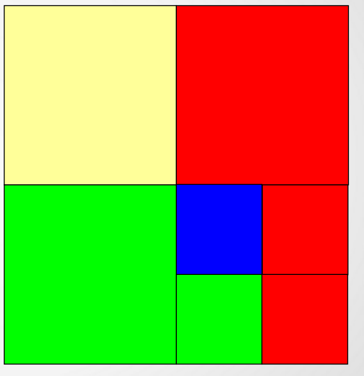


**Figure 5.23** A quad-tree representation of an 8 × 8 binary image.

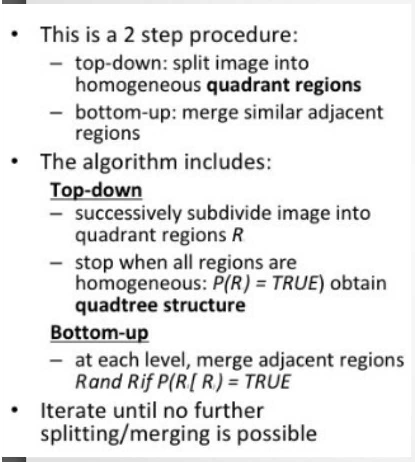


## Split and Merge

- Combination of both algorithms.
- Can handle a larger variety of shapes.
    - Simply apply previous algorithms consecutively.



## Split and Merge



- This is a 2 step procedure:
  - top-down: split image into homogeneous **quadrant regions**
  - bottom-up: merge similar adjacent regions
- The algorithm includes:

  **Top-down**
  - successively subdivide image into quadrant regions *R*
  - stop when all regions are homogeneous: *P(R) = TRUE*) obtain **quadtree structure**

  **Bottom-up**
  - at each level, merge adjacent regions *Rand Rif P(R[ R.) = TRUE*
- Iterate until no further splitting/merging is possible