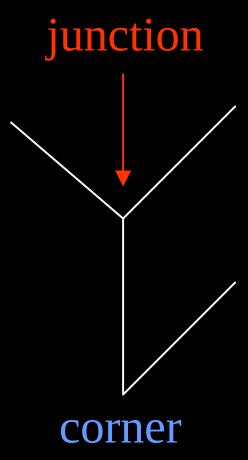
Finding Line and Curve Segments from Edge Images

Given an edge image, how do we find line and arc segments?

Method 1: Tracking



Use masks to identify the following events:

1. start of a new segment
2. interior point continuing a segment
3. end of a segment
4. junction between multiple segments
5. corner that breaks a segment into two

Edge Tracking Procedure

```
for each edge pixel P {
   classify its pixel type using masks
   case
      1. isolated point :          ignore it
      2.  start point :            make a new segment
      3.  interior point :         add to current segment
      4.  end point :              add to current segment and finish it
      5.  junction or corner :     add to incoming segment
                                   finish incoming segment
                                   make new outgoing segment(s)
```

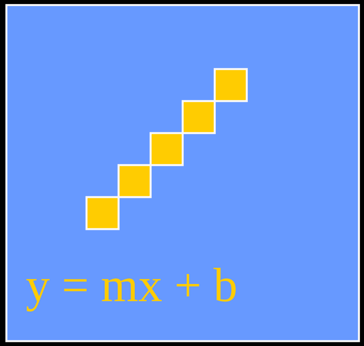
The ORT package uses a fancier corner finding approach.

Hough Transform

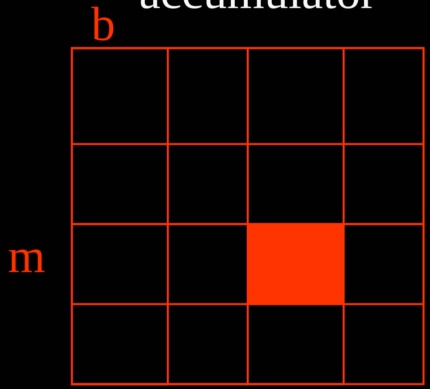
- The Hough transform is a method for detecting lines or curves specified by a parametric function.

- If the parameters are p1, p2, … pn, then the Hough procedure uses an n-dimensional accumulator array in which it accumulates votes for the correct parameters of the lines or curves found on the image.

image

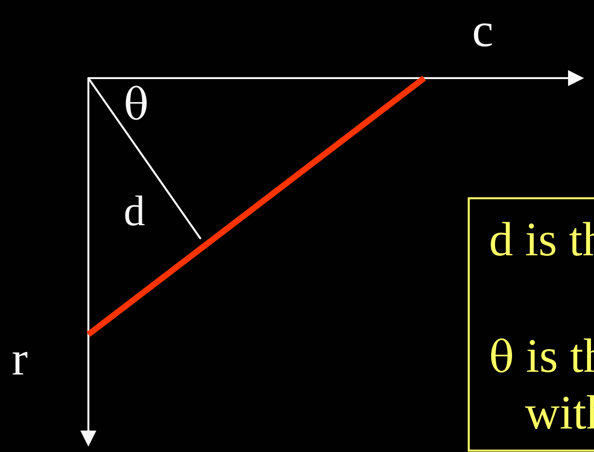


accumulator



Finding Straight Line Segments

- $y = mx + b$ is not suitable (why?)
- The equation generally used is: $d = r \sin \theta + c \cos \theta$



d is the distance from the line to origin

θ is the angle the perpendicular makes with the column axis

Procedure to Accumulate Lines

- Set accumulator array A to all zero.
  Set point list array PTLIST to all NIL.

- For each pixel (R,C) in the image {

  - compute gradient magnitude GMAG
  - if GMAG > gradient_threshold {

    - compute quantized tangent angle THETAQ
    - compute quantized distance to origin DQ
    - increment A(DQ,THETAQ)
    - update PTLIST(DQ,THETAQ) } }

Example

gray-tone image

0	0	0	100	100
0	0	0	100	100
0	0	0	100	100
100	100	100	100	100
100	100	100	100	100

DQ

-	-	3	3	-
-	-	3	3	-
3	3	3	3	-
3	3	3	3	-
-	-	-	-	-

THETAQ

-	-	0	0	-
-	-	0	0	-
90	90	40	20	-
90	90	90	40	-
-	-	-	-	-

Accumulator A

distance							
360	-	-	-	-	-	-	-
.	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-
3	4	-	1	-	2	-	5
0	-	-	-	-	-	-	-

angle 0 10 20 30 40 …90

PTLIST

360	-	-	-	-	-	-	-
.	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-
3	*	-	*	-	*	-	*
0	-	-	-	-	-	-	-

(1,3)(1,4)(2,3)(2,4)

(3,1)
(3,2)
(4,1)
(4,2)
(4,3)

How do you extract the line segments from the accumulators?

```
pick the bin of A with highest value V
while V > value_threshold {

   order the corresponding pointlist from PTLIST

   merge in high gradient neighbors within 10 degrees

   create line segment from final point list

   zero out that bin of A

   pick the bin of A with highest value V }
```

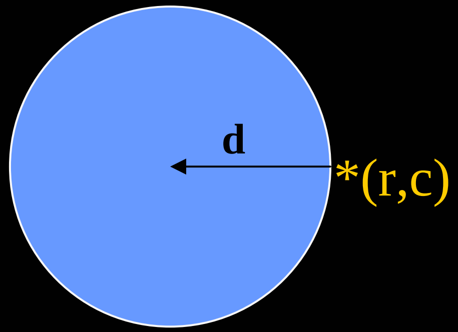
Finding Circles

Equations:

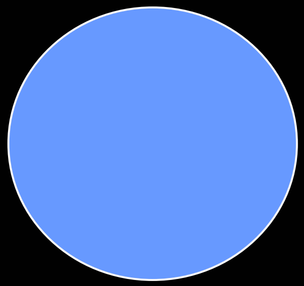
$$r = r0 + d \sin \theta$$
$$c = c0 + d \cos \theta$$

r, c, d are parameters

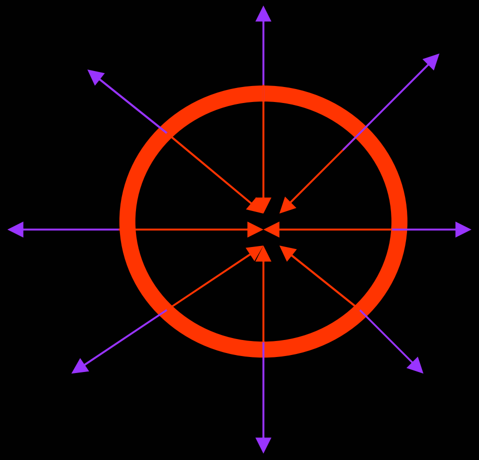
Main idea: The gradient vector at an edge pixel points to the center of the circle.



Why it works



Filled Circle:
Outer points of circle have gradient direction pointing to center.



Circular Ring:
Outer points gradient towards center.
Inner points gradient away from center.

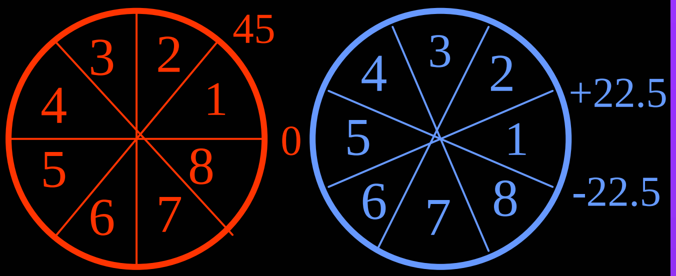
The points in the away direction don't accumulate in one bin!

Procedure to Accumulate Circles

- Set accumulator array A to all zero.
  Set point list array PTLIST to all NIL.

- For each pixel (R,C) in the image {
  For each possible value of D {
  - compute gradient magnitude GMAG
  - if GMAG > gradient_threshold {
    . Compute THETA(R,C,D)
    . R0 := R - D*cos(THETA)
    . C0 := C - D* sin(THETA)
    . increment A(R0,C0,D)
    . update PTLIST(R0,C0,D) }}

The Burns Line Finder



1. Compute gradient magnitude and direction at each pixel.
2. For high gradient magnitude points, assign direction labels to two symbolic images for two different quantizations.
3. Find connected components of each symbolic image.

- Each pixel belongs to 2 components, one for each symbolic image.
- Each pixel votes for its longer component.
- Each component receives a count of pixels who voted for it.
- The components that receive majority support are selected.

See transparencies for comparisons.

Consistent Line Clusters for Object Recognition (Yi Li's Structure Features)

A **Consistent Line Cluster** is a set of lines that are homogeneous in terms of some line features.

- **Color-CLC**: The lines have the same color feature.
- **Orientation-CLC**: The lines are parallel to each other or converge to a common vanishing point.
- **Spatially-CLC**: The lines are in close proximity to each other.

Color-CLC

- Color feature of lines: color pair (c_1, c_2)
- Color pair space:

  RGB $(256^3 * 256^3)$ Too big!

  Dominant colors (20*20)
- Finding the color pairs:

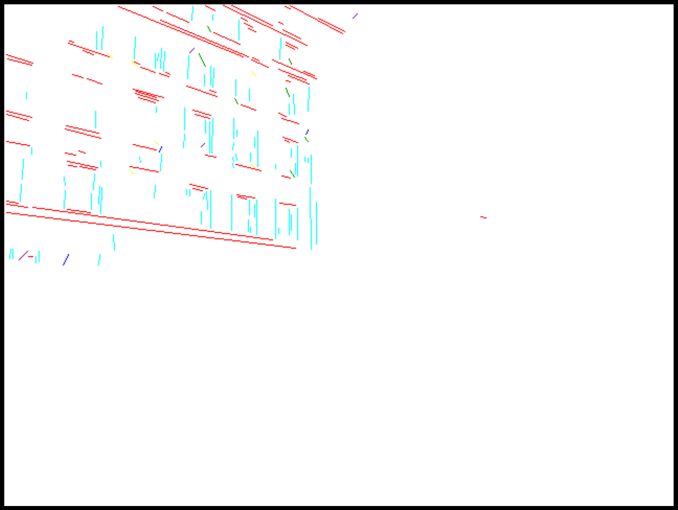
  One line → Several color pairs
- Constructing Color-CLC: use clustering

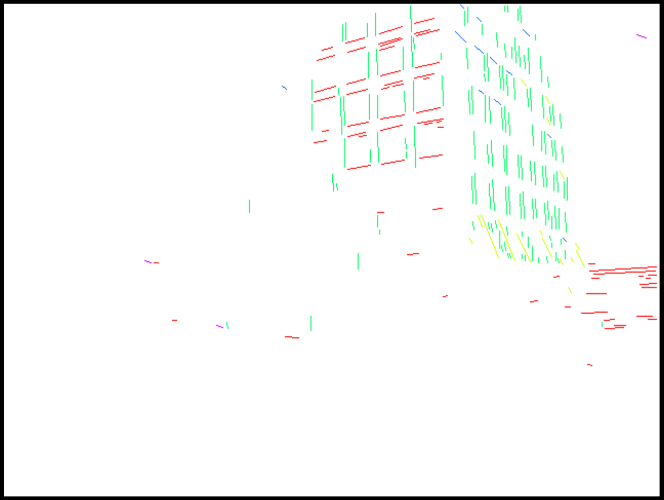
Color-CLC

Orientation-CLC

- The lines in an Orientation-CLC are parallel to each other in the 3D world
- The parallel lines of an object in a 2D image can be:
  - Parallel in 2D
  - Converging to a vanishing point (perspective)

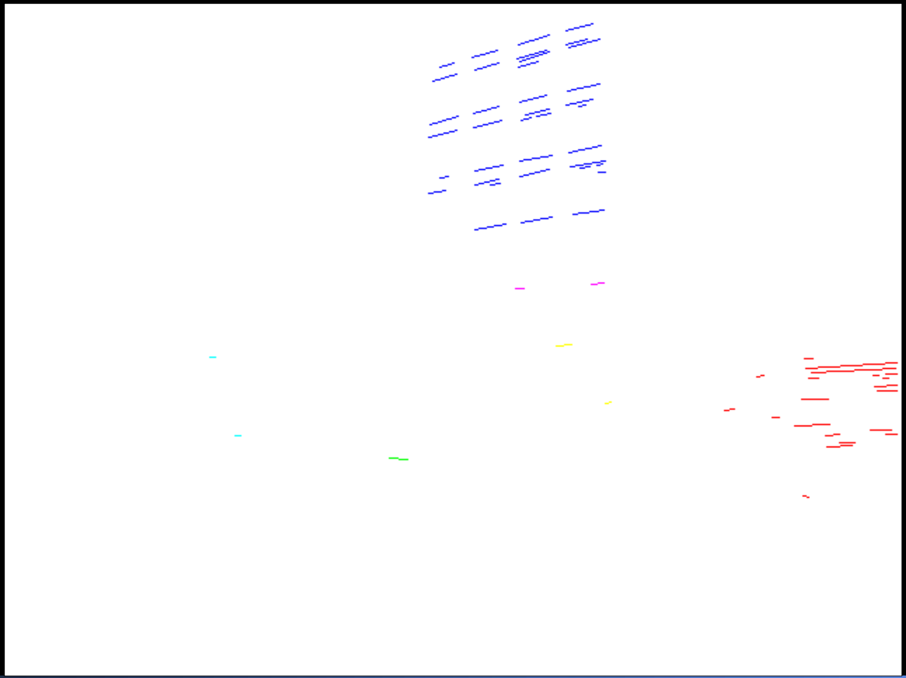
Orientation-CLC



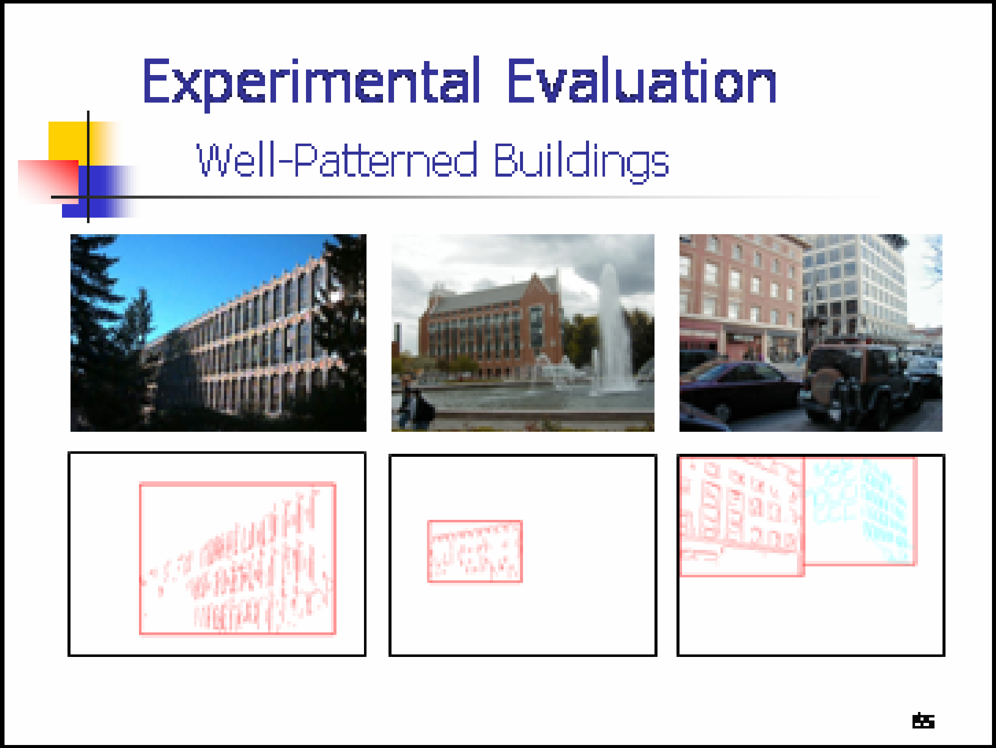


Spatially-CLC

- Vertical position clustering
- Horizontal position clustering



Use in Building Recognition (to be covered in the CBIR lecture)



http://www.cs.washington.edu/research/imagedatabase/demo