Lecture notes: Edge properties

Edge properties: endpoints, branchpoints, corners

Once a set of pixels has been labeled an edge, we can compute properties of the edge. For example, where are its endpoints? Does it branch? Does it have corners? These properties can be used in further higher-level analysis.

Most edge analysis algorithms require that the edge be only one pixel wide at all points:

X	X						
	X	X				X	X
	X	X	X	X	X	X	X
		X	X	X	X	X	
		X	X				
	X	X					
	X	X					
	X	X					

Which pixel exactly is the branchpoint, or an endpoint?

To help us define these things, we define some terms:

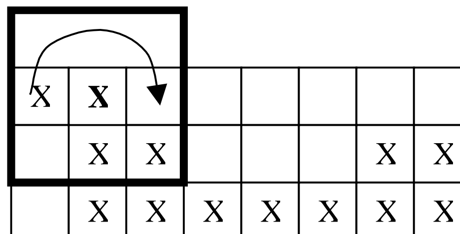
4-neighbors -- N,S,E,W of pixel of interest
8-neighbors -- NW,N,NE,W,E,SW,S,SE of pixel of interest

Thinning -- reducing a set of pixels to its skeleton
Skeleton -- the single pixel wide centerline(s) of a set of pixels

There are several thinning methods. Here is one:

1. Pass through the image looking at each pixel X.

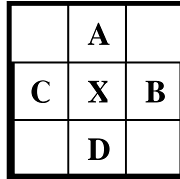
2. Count the number of edge->non-edge transitions in CW (or CCW) order around the pixel X:



(This pixel has 2 edge-to-non-edge transitions.)

3. Count the number of edge neighbor pixels (the example has 3).

4. Check that at least one of the North, East, or (West and South) are not edge pixels:



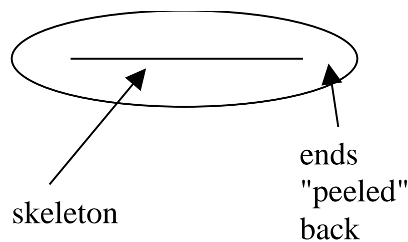
A or B or (C and D) != edge
(The example passes this test.)

5. The edge pixel is marked for erasure if it has

   a) exactly 1 edge -> non-edge transition,
   b) 2 <= edge neighbors <= 6, and
   c) passes item #4,

6. Once all pixels have been scanned, erase those marked, and repeat (back to step 1) until no pixels are marked for erasure.

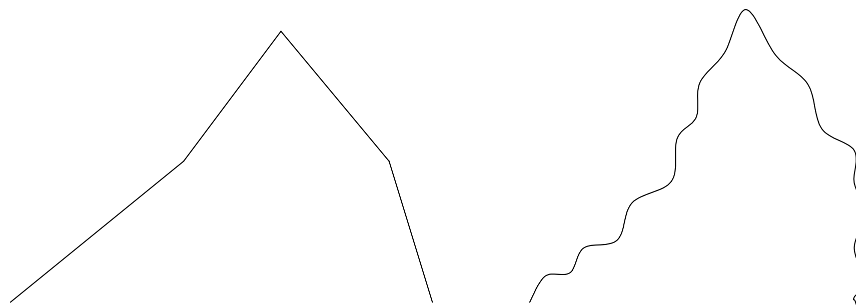
Going through this algorithm, you will see that it "peels back" layers of pixels until the skeleton is reached. Note that the ends are "peeled" just as much as the middle, until the skeleton is reached:



Having thinned the edges, now we can identify edge features:

Endpoint -- has exactly one edge -> non-edge transition
Branchpoint -- has more than two edge -> non-edge transitions

A corner is a more subjective entity, because scale is important:



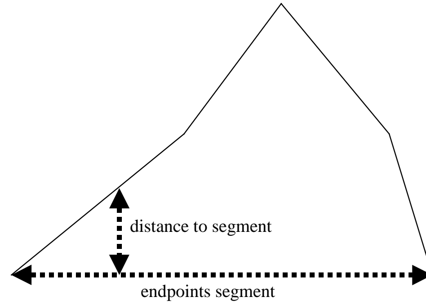
(How many corners do you see?)

In this example you might see one corner, or you might see three. If noise is added (as on the right), you might see more than ten corners. It all depends on what you are trying to do.

One method of finding corners is **iterative endpoint fitting**:

1. Find the line segment between the endpoints.

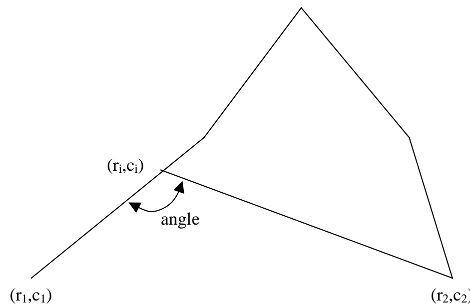
2. At each point on the edge, measure the distance to the line segment:



3. Find the point with the largest distance. If this distance is greater than some threshold, then the point is marked as a corner.

4. New corners spawn new endpoints, repeating from step 1.

Another method measures the angle between each edge point and the endpoints, marking the minimum as a corner if it is less than some threshold (e.g. smaller than 120 degrees):



where the angle is computed using the dot product

$$A \bullet B = |A||B|\cos\theta$$

Substituting we get

$$(r_1 - r_i)(r_2 - r_i) + (c_1 - c_i)(c_2 - c_i) = \sqrt{(r_1 - r_i)^2 + (c_1 - c_i)^2}\sqrt{(r_2 - r_i)^2 + (c_2 - c_i)^2}\cos\theta$$

which yields one equation for the one unknown (theta).

How do we trace along the edge?

Boundary tracing -- identifying the coordinates of the perimeter of a set of pixels in CW or CCW order

Again, there are several methods to trace boundaries. Here is one:

1. Initialize all edge pixels as "not visited".

2. Scan image top->down left->right looking for unvisited edge pixel. Record location as starting point.

3. Initialize Angle=NE (NE pixel could not have been edge, but E could).

4. Rotate Angle 45° CW. If pixel-at-angle is not edge, repeat step 4.

5. Goto pixel-at-angle. If back to original pixel, goto step 2. Otherwise rotate Angle -135° CW and goto step 4.

This procedure traces the boundary of every set of pixels in the image.