Lecture 16: Segmentation (Edge Based, cont'd)

©Bryan S. Morse, Brigham Young University, 1998–2000
Last modified on February 26, 2000 at 6:00 PM

Contents

Reading

SH&B, 5.2.1–5.2.3

16.1 Introduction

In earlier lectures, we discussed how to identify possible edge pixels. However, these tests are made one-by-one and don't necessarily produce closed object boundaries because of noise, intensity variation, etc. As we discussed in the last lecture, there are many techniques for extracting closed contours given isolated potential edge pixels.

Fitting approaches, such as the Hough transform, don't need to connect the edge pixels because they try to find the best fit of a *known* shape to the edge data. Such approaches are useful for many applications where the general shape (though not its exact position and other parameters) are known ahead of time. In many applications, you don't know the general shape ahead of time, and you thus need to find connected sets of edge pixels explicitly.

In this lecture, we'll start talking about how to link sets of possible edge pixels to find connected boundaries. (In other words, we'll start playing connect-the-dots.)

16.2 Gradient-Magnitude Thresholding

As we have mentioned previously, a common step in edge detection is to follow the gradient operator by a magnitude thresholding operation.

$$E(x,y) = \begin{cases} 1 & \text{if } \|\nabla f(x,y)\| > T \text{ for some threshold } T \\ 0 & \text{otherwise} \end{cases} \tag{16.1}$$

We'll call $\{(x,y) : E(x,y) = 1\}$ the set of *edge pixels*.

Thresholding the gradient magnitude leaves only "strong" edges, but it makes no guarantees of continuity. Your text discusses two general techniques for getting around this problem.

- thresholding with hysteresis
- edge relaxation

Both of these really fall in to the broader category of "relaxation algorithms": ones in which you make an initial tentative labeling or decision then "relax" the categorization *based on what you've already tentatively decided*. The goal is to come up with a globally consistent decision or set of labels.

16.3 Thresholding with Hysteresis

Hysteresis generally refers to the way certain systems tend to stay consistent under gradually changing conditions. They may change from one state to another, but only after a large amount of change–they tend not to fluctuate mildly under small changes.

Thresholding with hysteresis uses two thresholds:

$$\begin{array}{rl} \|\nabla f(x,y)\| \geq t_1 & \text{definitely an edge} \\ t_0 \geq \|\nabla f(x,y)\| < t_1 & \text{maybe an edge, depends on context} \\ \|\nabla f(x,y)\| < t_0 & \text{definitely not an edge} \end{array}$$

The idea is to first identify all definite edge pixels. Then, add all "maybe" pixels (those greater than t_0) *only if they are next to an already-labeled edge pixel*. This process is repeated until it converges.

16.4 Local Processing: Edge Relaxation

A process similar to thresholding with hysteresis is *edge relaxation*.

The idea of edge relaxation is not not simply add pixels if they are next to other edge pixels but to consider the context as well.

Let's consider this question of whether or not a pixel between two sets of edge pixels is itself an edge pixel. One way of determining this is to look at the magnitude of the intervening pixel: if it is relatively high, but less than the threshold used to determine that its neighbors are edge pixels, it's probably an edge. Of course, we can also check the similarity of the gradient magnitude and gradient orientation, just like we did with edge linking.

We can also use this not just to interpolate between edge pixels, but to extrapolate from them as well. Suppose that we have two adjacent edge pixels followed my a slightly sub-threshold one (with similar gradient magnitude and orientation). Again, it's likely that this is really an edge pixel.

We can add these possible pixels to the set of edge pixels and repeat the process. Supposing that these are now really edge pixels, there may be other near-misses that we might want to allow as edge pixels.

In a sense, we are successively relaxing the criteria used to determine edge pixels, taking into account not just the properties of the pixel in question but of its neighbors as well. This process is called *edge relaxation*.

In general, the term *relaxation* applies to any technique such as this that iteratively re-evaluates pixel classification.

16.5 Local Processing: Edge Linking

One way to find connected sets of edge pixels, without having to explicitly first identify which are or are not edges is to trace from pixel to pixel through possible edge points, considering as you go the context along the path.

We can link adjacent edge pixels by examining each pixel-neighbor pair and seeing if they have similar properties:

1. Similar gradient magnitude: $|\ \|\nabla f(x,y)\| - \|\nabla f(x',y')\|\ | \leq T$ for some magnitude difference threshold T.
2. Similar gradient orientation $|\ \phi(\nabla f(x,y)) - \phi(\nabla f(x',y'))\ | \leq A$ for some angular threshold A.

Once the links are established, we take sets of linked pixels and use them as borders.

Notice that unless you constrain the linked pixels in some sense (for example, by scanning along horizontal or vertical lines), these can create clusters of linked pixels rather than long single-pixel thick chains.

Edge linking is usually followed by postprocessing to find sets of linked pixels that are separated by small gaps–these can the be filled in.

As we'll see later, this edge-linking idea can be extended further by considering the set of possible edges as a graph and turning it into a minimum-cost graph searching problem.

16.6 Boundaries of Already-Segmented Regions

In some cases, we may already have an image segmented into regions for which we want to calculate boundaries. In this case, we can simply generate the boundaries by tracing around the region contours. We can do this two ways:

- Trace once around each contour in the image. When we finish tracing one contour, scan the image until we run into another.
- Make one pass through the image, using data structures to keep track of each contour and adding pixels to the appropriate contour as encountered.

The first method is far more common, because you can simply trace each border one by one.

Your text gives a number of algorithms for tracing borders, and they're fairly easy to understand. Think of them as walking around the object with your hand on one wall. The algorithms for tracing these borders are pretty much the same as you'd use if you were in a dark room tracing along a wall and around corners.

These algorithms include tracing either the inner border or the outer border. However, tracing the inner borders means that the borders of two adjacent regions do not share the same pixels (they are adjacent). Likewise, tracing the outer borders means that the border of one object is *inside* the border of an adjacent object. To get around these problems, a hybrid method known as *extended borders* has been developed.

Extended borders basically use inner borders for the upper and left sides of the object and outer borders for the lower and right sides. By doing this, one object's border is the same as the shared border with the adjacent object. (It also has the nice property of being closer in perimeter to the actual shape, thus avoiding the inner-perimeter/outer-perimeter ambiguity we talked about earlier.) The algorithm for extended border extraction is similar to those for inner or outer borders, but you basically keep track at each stage of where the object is relative to you (above or below, left or right).

Vocabulary

- Edge thresholding
- Thresholding with hysteresis
- Edge relaxation
- Edge linking
- Relaxation algorithms