Subject: Re: Label_region and Erosion
Posted by Struan Gray on Wed, 04 Nov 1998 08:00:00 GMT
View Forum Message <> Reply to Message

Lisa Bryan, lbryanNOSPAM@arete-az.com writes:

> Now if I can just get dilate to work on a
> greyscale image.

I'm not sure you really need any help, but here are some random thoughts.

As a general rule, IDL is much happier, and programs run much faster, if you use as much memory and as few loops and sub-arrays as possible. Unfortunately, the duty sadists at RSI have seen to it that you have no decent tools within IDL to keep track of your memory usage. However, your images are not too large, so keeping extra masks and working copies of the image in memory is not going to be a problem on most platforms, even if like me you need to keep users with elderly, RAM-poor, PC-class machines in mind.

So you can always make a binary mask from your greyscale image, and use that to define your regions. The most obvious way is to manually set pixel values using the output of a WHERE call, but often it is possible, and faster, to use a array comparison like:

mask = image > threshold_val

where threshold_val distinguishes between the regions (100 or so in your example) and mask becomes a binary image containing 0s where the condition is false and 1s where it is true. Then you can erode and dilate this mask to get rid of single-pixel spikes and feed the result to LABEL_REGION to find the regions themselves.

At this point I find it useful to make a *second* mask, which is a n-times dilated version of the first. By subtracting the first from the second I get a mask containing a n-pixel wide ribbon of boundary pixels for the regions I want to filter. I can then use the ribbon to fill the relevant pixels of a temporary copy of the original image with values appropriate to whatever filter I want to apply (region average, nearest value, NANs, whatever). Having filtered the copy of the image containing the ad-hoc ribbon values I copy the filtered regions back into the final image using the first mask as a guide.

This sounds complex, but is not hard to implement and is particularly useful for things like differentiation filters where the example you gave would produce nasty effects at the edges of the dilated but unaltered regions where the values go from 300+/-10 to

0+/-10. My way puts all the nasties outside the region you are actually going to use. Downsides are that if you want to filter both the 300+/-10 areas and the 0+/-10 ones you have to do the whole thing twice, and that the choice of pixel values to put into the ribbons, while often perfectly logical and scientifically respectable, is a fudge that might lead unsophisticated users astray, though the degree to which that occurs depends on both the nature of your data, and of your users.

Enough waffle. I hope this helps.

Struan
