Subject: Re: Thinning algorithm without for loops
Posted by Conor on Tue, 07 Aug 2007 13:28:18 GMT
View Forum Message <> Reply to Message

On Aug 6, 3:55 pm, "Jeff N." <jnett...@utk.edu> wrote:
> On Aug 6, 3:31 pm, nathan12343 <nathan12...@gmail.com> wrote:
>
>
>
>> Hi all-
>
>> I'm trying to impliment the Zhang-Suen thinning algorithm in IDL.
>> This particular algorithm decides whether a pixel needs to be deleted
>> or not based on properties of the pixels immediately surrounding the
>> pixel we are concerned with (i.e. a pixel's 8-neighbors). This
>> naturally lends its self to for loops. Let's say I have an image, a
>> 512X512 array of bytes. The code iteratively scans over each pixel
>> and determines whether it needs to be set to 0 based on the Zhang-Suen
>> thinning rules. What I can't figure out is how to scan the images
>> without for loops. If I use for loops I can easily index the pixels
>> immediately surrounding image[i,j] by saying image[i-1,j] or image[i
>> +1,j-1], etc.
>
>> Does anyone know of a way to do this kind of indexing in an image
>> without the use of for loops?
>
>> -Nathan Goldbaum
>
> I've never done this myself, so someone else will probably have to
> give you details if you need more help, but I think the function you
> need is the SHIFT() function. Have a look at the help files for that
> and see what you think.
>
> Jeff

It really depends on just what the thinning algorithm needs to check.
Can you be specific about that? In general though, the shift function
is probably the way to go, although it will be rather ugly at the same
time (since you'll have to call it 8 times). Let's pretend for a
moment that your thinning algorithm is very simple: namely, let's
imagine that you want to ignore all pixels where the average
surrounding pixels have a value greater than some threshold.

```
; make a fake image
img = byte( randomu(seed,512,512)*100 )
; array to hold the total values
tot = intarr( 514,514 )
```

```
; pad img with zeroes on all sides, because shift() wraps around an
array and you don't want values from the other side
img =  [[fltarr(514)],[fltarr(1,512),img,fltarr(1,512)],[fltarr(514 )]]

; now find the total value of all neigboring pixels
tot += shift(img,1,-1)
tot += shift(img,1,0)
tot += shift(img,1,1)
tot += shift(img,0,-1)
tot += shift(img,0,1)
tot += shift(img,-1,-1)
tot += shift(img,-1,0)
tot += shift(img,-1,1)

; now find the average
avg /= 8

; now pull the original image and the totals out of the padded arrays
img = img[1:512,1:512]
avg = avg[1:512,1:512]

; finally, select everything below a certain threshold:
w = where( avg lt threshold )
```

It's ugly, and it probably isn't the best solution, but it will get
the job done and it will probably be much faster than a loop
solution. Of course, that depends on just what the thinning algorithm
does, and whether or not it can be generalized in such a fashion.