Subject: Re: Thinning algorithm without for loops
Posted by JD Smith on Wed, 08 Aug 2007 21:33:03 GMT
View Forum Message <> Reply to Message

On Tue, 07 Aug 2007 21:57:05 +0000, nathan12343 wrote:

> On Aug 7, 12:45 pm, nathan12343 <nathan12...@gmail.com> wrote:
>> [quoted text muted]
>
> Thanks for your help, Conor, the shift function appears to have done
> the trick.
>
> This code implements the first iteration of Zhang-Suen thinning
> without a single for loop!
>
> PRO zsthin,img,thinimg
>
> siz=size(img)
>
> ;Array to hold the sums we're looking for
> tot=lonarr(siz[1],siz[2])
>
> tot+=shift(img,1,0)
> tot+=shift(img,1,1)
> tot+=shift(img,1,-1)
> tot+=shift(img,0,1)
> tot+=shift(img,0,-1)
> tot+=shift(img,-1,0)
> tot+=shift(img,-1,1)
> tot+=shift(img,-1,-1)

Here's an alternative set of approaches.

Complete test 1 using CONVOL:

Test 1:

```
k=make_array(3,3,VALUE=1.)
k[1,1]=0.
tot=convol(img,k,/EDGE_WRAP,/CENTER)
del=where(img AND tot ge 2 AND tot le 6,del_cnt)
```

Now you only need to do the rest of the tests on the 'del_cnt' pixels which passed the first test. As you pass each subsequent test, you discard all pixels which didn't pass.

Since you need to accumulate all of p[1]...p[8] into a single array of size 8xn, you might instead just build the indices directly yourself,

rather than shift and concatenate.

```
xs=siz[0]
offs=[-xs,-xs+1,1,xs+1,xs,xs-1,-1,-xs-1] ; p[1]...p[8]
t=[8,del_cnt]
del=rebin(transpose(del),t,/SAMPLE)+rebin(offs,t,/SAMPLE)

p=img[del] ; 8xn list of the neighbors of those pixels which passed test 1.
```

Now you can proceed with your tests.

Test 2:

```
del2=where(total(p eq 0 AND shift(p,-1,0) eq 1,1,/PRESERVE_TYPE) eq 1b,cnt2)
p=p[*,del2]
del=del[del2]
```

Test 3:

```
del3=where(p[2,*]*p[4,*]*p[6,*] eq 0,cnt3)
p=p[*,del3]
del=del[del3]
```

Test 4:

```
del4=where(p[0,*]*p[2,*]*p[4,*] eq 0,cnt4)
del=del[del4]
```

And so del is now a list of indices in img to be deleted. How this resulting trim list is applied during iteration 2 wasn't clear from your description, but the same techniques should work there as well.

You'll want to insert checks after each test to ensure some pixels actually passed. Note that the offset method does not "wrap around" on edge pixels, but just truncates to the last pixel in that direction (i.e. the first or last in the array). If you care about edge pixels, you should probably pad the array first anyway.

JD
