Subject: Thinning algorithm without for loops
Posted by nathan12343 on Mon, 06 Aug 2007 19:31:35 GMT
View Forum Message <> Reply to Message

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

Subject: Re: Thinning algorithm without for loops
Posted by nathan12343 on Wed, 08 Aug 2007 17:41:24 GMT
View Forum Message <> Reply to Message

On Aug 8, 1:33 am, Paolo_Grigis <pgri...@astro.phys.ethz.ch> wrote:
> [...]
>
>
>
>> This code implements the first iteration of Zhang-Suen thinning
>> without a single for loop!
>
> You are shifting "img" too many times... you only need to compute
> your neighbors values p1... p8 first, and then you can have the
> next statements use that values instead, for instance
>
> tot=p1+p2+...+p8
>
> and so on for the other conditions.
>
> Ciao,
> Paolo
>
>> 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)
>
>>  cond3=intarr(siz[1],siz[2])
>
>>  cond3[*,*]=1
>
>>  cond3*=shift(img,1,0)
>>  cond3*=shift(img,-1,0)
>>  cond3*=shift(img,0,-1)
>
>>  ;4. If P[1]*P[3]*P[5]=0
>
>>  cond4=intarr(siz[1],siz[2])
>>  cond4[*,*]=1
>
>>  cond4*=shift(img,0,1)
>>  cond4*=shift(img,1,0)
>>  cond4*=shift(img,0,-1)
>
>>  ;2. The number of 0-1 transitions in the ordered sequence
>>  ;P[1],P[2],...,P[8],P[1] Is exactly 1
>
>>  p1=shift(img,0,1)
>>  p2=shift(img,1,1)
>>  p3=shift(img,1,0)
>>  p4=shift(img,1,-1)
>>  p5=shift(img,0,-1)
>>  p6=shift(img,-1,-1)
>>  p7=shift(img,-1,0)
>>  p8=shift(img,-1,1)
>
>>  cond2=intarr(siz[1],siz[2])
>
>>   p=[[[p1]],[[p2]],[[p3]],[[p4]],[[p5]],[[p6]],[[p7]],[[p8]],[ [p1]]]
>
```

>> FOR i=0,7 DO BEGIN
>> wh=where(p[*,*,i] eq 0 AND p[*,*,i+1] eq 1)
>> cond2[wh]+=1
>> ENDFOR
>
>> tvscl,cond2
>
>> wh=where(cond2 eq 1)
>
>> cond2[*,*]=0
>
>> cond2[wh]=1
>
>> wh=where(tot GE 2 AND tot LE 6 AND cond3 eq 0 AND cond4 eq 0 AND cond2
>> eq 1)
>
>> wh11=intarr(siz[1],siz[2])
>
>> wh11[wh]=1
>
>> newimg=img-wh11
>
>> newimg>=0
>
>> END
>
>> I'll do the second subiteration in a bit, should be too hard.

You are correct, thanks for pointing that out!

-Nathan

Subject: Re: Thinning algorithm without for loops
Posted by nathan12343 on Thu, 09 Aug 2007 01:46:54 GMT
View Forum Message <> Reply to Message

On Aug 8, 3:33 pm, JD Smith <jdsm...@as.arizona.edu> wrote:
> 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

JD, really elegant way of doing that. I'm still trying to figure out what you did with those two rebin commands to resize the pixels that passed step 1, I'm sure I'll figure it out tomorrow, though. I love how big complicated problems can be solved with just a few rebins and reforms in IDL.

Thanks again for everyone's help!

-Nathan Goldbaum

Subject: Re: Thinning algorithm without for loops
Posted by JD Smith on Thu, 09 Aug 2007 16:52:09 GMT
View Forum Message <> Reply to Message

On Thu, 09 Aug 2007 01:46:54 +0000, nathan12343 wrote:

> On Aug 8, 3:33 pm, JD Smith <jdsm...@as.arizona.edu> wrote:
>> [quoted text muted]
>
> JD, really elegant way of doing that. I'm still trying to figure out
> what you did with those two rebin commands to resize the pixels that

> passed step 1,

Just some simple rearrangement. I take the list of kept indices, del=[d1,d2,d3,...], and turn it into an 8xn copy of itself, on its side:

```
 d1 d1 d1 d1 d1 d1 d1 d1
 d2 d2 d2 d2 d2 d2 d2 d2
 d3 d3 d3 d3 d3 d3 d3 d3
 . . . . . . . . .
 . . . . . . . . .
```

To each row of this, I add a custom offset to index the upper right, right, lower right, lower, lower left, left, upper left, upper neighbors. The offset in terms of the single "running" index into the array (of the type returned by WHERE) is -xs for the row above, +xs for the row below, +1 to the right, -1 to the left, etc. I take those eight offsets, call them o1...o8, and simply add them to each row:

```
 o1 o2 o3 o4 o5 o6 o7 o8
 o1 o2 o3 o4 o5 o6 o7 o8
 o1 o2 o3 o4 o5 o6 o7 o8
 . . . . . . . . .
 . . . . . . . . .
```

Add these two together, and you have all 8 neighbors of each of the d's together in a row.

JD
