Subject: Re: filtering problem
Posted by John-David T. Smith on Fri, 17 Nov 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Allan wrote:
```
>
>  This should do the trick.
>  Allan Wienecke
>  Visual Numerics, Inc.
>  Celebrating 30 Years as an Independent Software Vendor
>
>  ; a is the array
>  ; n is the kernel size
>  ; t is the threshold
>
>  function dave, a, n, t
>
>    m = a gt t
>    k = replicate( 1.0, n, n )
>
>    return, a * avg(a) * convol(m,k) / (convol(m*a,k)+1e-30)
>
>  end
>
>  Dave Brennan wrote:
>
>>  Hi,
>>
>>  i don't know if anyone can help but it's worth a try!
>>
>>  I am trying to filter an array say (256x256) with a window of size 65x65
>>  which scans across the array pixel by pixel. It should compare the
>>  statistics of the area within the kernal with the global statistics of
>>  the image to produce a correction image. (This is a particular type of
>>  inhomogeneity correction)
>>
>>  In detail: 'the algorithm should correct the pixel value by a
>>  multiplicative factor found by dividing the global mean by the window
>>  mean'
>>  A further problem is I want the ability to set a threshold where data
>>  below the threshold are not included in the statistics and not corrected
>>  by the algorithm.
```

Hmmm... well you have to take care of the edges somehow, and I think
Dave requested that the data below the threshold not be included in any
statistic (including presumably the global mean), and that the
sub-threshold data not be modified, so I changed your dave() to be (now

a procedure):

```
pro dave, a, n, t
   m = a ge t
   wh = where(m,cnt)
   if cnt eq 0 then return
   k = replicate( 1.0, n, n )
   a[wh]=a[wh] * mean(a[wh]) * (convol(float(m),k,/EDGE_TRUNCATE) / $
    (convol(a*m,k,/EDGE_TRUNCATE)+1e-30))[wh]
end
```

I added the "where" to get the super-threshold mean, and to leave the sub-threshold pixels alone. I also used one of the EDGE keywords to keep from having a band of uncorrected values on the edges. Note that I had to change your "convol(m..." to "convol(float(m)...", because convolving a byte array (or any other small range type) is not usually what you want. It will typically just rail to the top of the range (e.g. 255b). Why total insists on converting to float and smooth and convol don't, we'll never know (but see the SCALE argument to convol()).

That said, here's a much better version, written in the same format to ease comparison:

```
pro thresh, a, n, t
   m = a ge t
   wh = where(m,cnt)
   if cnt eq 0 then return
   a[wh] = a[wh] * mean(a[wh]) * (smooth(float(m),n,/EDGE) / $
    (smooth(a*m,n,/EDGE)+1.e-30))[wh]
end
```

The basic difference is one of using smooth() instead of convol(). This is faster. *Much* faster. For a 256x256 array of randomu's with a threshold value of .1, and box size 65x65, it's ~500 times faster -- that's 1/8 of a second vs. a minute on my machine. convol() is great for doing convolutions with interesting kernels (gaussians, pictures of david fanning, etc.). For kernels of 1's and 0's, it is a total waste. Think of all those "1's" being multiplied over and over and over. Oh my.

For other fast and fun array operations, see median(), and a post of mine from around a month ago documenting many various tricks such tricks.

JD

P.S. If the below-threshold pixels are always few in number, you could

speed up a (tiny) bit by doing the whole assignment over the full array,
and specifically restoring those pixels.

--
J.D. Smith | WORK: (607) 255-6263
Cornell Dept. of Astronomy | (607) 255-5842
304 Space Sciences Bldg. | FAX: (607) 255-5875
Ithaca, NY 14853 |