Subject: How to create a 2D mask that automatically half’s an irregularly shaped 2D array from top to bottom?
Posted by Dr G. on Fri, 18 Nov 2011 15:04:13 GMT
View Forum Message <> Reply to Message

Hi Folks,

Q: Can the IDL geometry geniuses out there think of a fast way to
create a 2D mask that automatically half’s an irregularly shaped 2D
array along its x axis (i.e., from top to bottom) Eg:

```
[0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0]
[0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0]
[0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0]
[0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0]
[0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0]
[0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0]
[0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0]
[0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0]
[0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0]
[0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0]
[0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0]
[0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0]
```

Merci.

Gf

Subject: Re: How to create a 2D mask that automatically half’s an irregularly shaped 2D array from top to bottom?
Posted by Jeremy Bailin on Wed, 23 Nov 2011 15:45:09 GMT
View Forum Message <> Reply to Message

On 11/22/11 5:47 PM, Dr G. wrote:
> On Nov 19, 7:59 am, Jeremy Bailin<astroco...@gmail.com> wrote:
>> On 11/18/11 2:53 PM, Jeremy Bailin wrote:
>>
>>
>>
>>
>>
>>
>>
>>
>>
>>> On 11/18/11 10:04 AM, Dr G. wrote:
>>>> Hi Folks,

>>
>>>> Q: Can the IDL geometry geniuses out there think of a fast way to
>>>> create a 2D mask that automatically halfï¿½s an irregularly shaped 2D
>>>> array along its x axis (i.e., from top to bottom) Eg:
>>
>>>> [0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0]
>>>> [0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0]
>>>> [0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0]
>>>> [0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0]
>>>> [0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0]
>>>> [0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0]
>>>> [0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0]
>>>> [0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0]
>>>> [0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0]
>>>> [0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0]
>>>> [0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0]
>>>> [0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0]
>>
>>>> Merci.
>>
>>>> Gf
>>
>>> If the input mask is "inmask":
>>
>>> ; how many 1s are there?
>>> rowtot = total(inmask, 1, /int)
>>> ; and it by checking if the cumulative total along the row is less
>>> ; than half of rowtot
>>> outmask = inmask and (total(inmask, 1, /int, /cumul) le $
>>> rebin(transpose(rowtot/2), masksize, /sample))
>>
>>> -Jeremy.
>>
>> Oops, missed the first line when I copied that in:
>>
>> masksize = size(inmask, /dimen)
>>
>> -Jeremy.
>
> Hi Jeremy,
>
> This is pretty amazing. Much better than a nasty for-next for each
> row. Thank you. This is truly geometry genius. My intention is to
> apply your method to masks 60,000 rows deep!!! So far in my tests,
> your solution rapidly works for most situations, but the output is a
> little strange when the input mask is very strangely shaped. One
> alternative to this, is to rotate the mask 90 degrees so that columns
> become rows, and then apply it.

>
> Jeremy, can you please take a moment and explain to me how the rebin&
> transpose combo works? I've read the help file on these but cannot
> figure it out the geometry genius you use.
>
> Additionally, you write "...and it by checking if the cumulative
> total ..." do you mean 'add it...'?
>
> Thank you for your consideration.
>
> Dr G.

The only to worry about when applying it to very large arrays is, of course, memory - it internally needs two arrays of the same size as the input mask. But if you run into problems, you could always try breaking it up into a few chunks and doing one chunk at a time, since each row is independent.

When you say the output is a little strange, what exactly does the input mask do? My solution assumes that each row has only one string of 1s in it - if it has more, then it won't break up each individual string, but will just cut out the last half of them.... a correct solution would require some fancy footwork with label_region.

I mean "and" as in "bitwise and"... it takes the input mask and performs a bitwise and with the test mask. Here's a breakdown of what it does. Let's say the input is:

```
inmask = [[0b,0,1,1,1,1,1,0], [0,1,1,1,1,0,0,0], [1,1,1,1,1,1,0,0]]
```

Then masksize is just the dimensions, in this case [8,3]. The first line figures out how many 1s there are in each row: [5,4,6].

The line that does the work takes two arrays of the same dimension as inmask and compares them. To cut the string of 1s in half, we want rowtot/2 of them to remain. The rebin(transpose(... part generates an 8x3 array where each row contains that value. So rowtot/2 is a 3-element array containing [2,2,3], transpose(rowtot/2) is a 1x3 array containing [2,2,3] in the column, and rebin(transpose(...) turns it into an 8x3 array where the [2,2,3] column is expanded out along the first dimension:

```
IDL> print, fix(rebin(transpose(rowtot/2), masksize, /sample))
      2      2      2      2      2      2      2      2
      2      2      2      2      2      2      2      2
      3      3      3      3      3      3      3      3
```

(I'm just using fix to turn the longs into short ints so they display nicely). I then compare that to the cumulative number of 1s along the row:

```
IDL> print, fix(total(inmask, 1, /int, /cumul))
       0       0       1       2       3       4       5       5
       0       1       2       3       4       4       4       4
       1       2       3       4       5       6       6       6
```

When the cumulative count (that second array) reaches half of the total
(the first array), that's all of the ones we want to preserve. So this
is the mask where that's true:

```
IDL> print, total(inmask, 1, /int, /cumul) le $
IDL>   rebin(transpose(rowtot/2), masksize, /sample)
   1   1   1   1   0   0   0   0
   1   1   1   0   0   0   0   0
   1   1   1   0   0   0   0   0
```

Compare that to the input mask:

```
IDL> print, inmask
   0   0   1   1   1   1   1   0
   0   1   1   1   1   0   0   0
   1   1   1   1   1   1   0   0
```

You can see that the mask we've generated picks out the first half of
each string of 1s in inmask. So then we bitwise and them and we're done:

```
IDL> print, inmask and (total(inmask, 1, /int, /cumul) le $
IDL>   rebin(transpose(rowtot/2), masksize, /sample))
   0   0   1   1   0   0   0   0
   0   1   1   0   0   0   0   0
   1   1   1   0   0   0   0   0
```

-Jeremy.

Subject: Re: How to create a 2D mask that automatically half's an irregularly shaped 2D array from top to bottom?
Posted by Jeremy Bailin on Sun, 04 Dec 2011 06:15:47 GMT
View Forum Message <> Reply to Message

> When you say the output is a little strange, what exactly does the input
> mask do? My solution assumes that each row has only one string of 1s in
> it - if it has more, then it won't break up each individual string, but
> will just cut out the last half of them.... a correct solution would
> require some fancy footwork with label_region.

Here's a solution that should work for any arbitrary mask. I'm sure it's
not as fast as the original, but it should be pretty efficient... there

is a for loop, but it is at most a loop through the columns rather than
the rows (and depending on the mask, could be much shorter).

For reference, it's inspired by part of JD's double-histogram solution
to the chunk indexing problem, although it doesn't actually use a double
histogram.

```
masksize = size(inmask, /dimen)

; expand the mask with a column of 0s on each side
; so that shift will not cycle 1s around
mask_expand = bytarr(masksize[0]+2, masksize[1])
mask_expand[1:masksize[0],*] = inmask

; find the left and right ends of each string of 1s
; by checking to see if it changes when shifted left
; or right by one column
leftends = where(mask_expand and not shift(mask_expand,1), nstring)
rightends = where(mask_expand and not shift(mask_expand,-1))
; find the midpoints by averaging
midpts = rebin([[leftends],[rightends]], nstring)
; and how long is each string?
lengths = midpts - leftends + 1

; clear mask_expand so we can fill it up again
mask_expand[*] = 0

; histogram the lengths so we can loop through them and use
; reverse indices to know which string has which length
hlen = histogram(lengths, min=1, reverse_indices=lenri)

; single indices are easy
if hlen[0] gt 0 then mask_expand[leftends[lenri[lenri[0]:lenri[1]-1]]]=1

; loop through length counts
for j=1, n_elements(hlen)-1 do if hlen[j] gt 0 then begin
   ; which ones have this length?
   vec_inds = lenri[lenri[j]:lenri[j+1]-1]
   ; make a 2D array of numbers starting at each left end of this length
   ; and incrementing along the second dimension. This array contains
   ; all of the indices into mask_expand that we want to set
   vec_inds = rebin(leftends[vec_inds], hlen[j], j+1, /sample) + $
     rebin(lindgen(1,j+1), hlen[j], j+1, /sample)
   ; and set them
   mask_expand[vec_inds] = 1
endif
```

```
; finally get rid of those extra columns of 0s we added at the beginning
outmask = temporary(mask_expand[1:masksize[0],*])
```

-Jeremy.