Subject: Re: weird behavior of Triangulate
Posted by Yngvar Larsen on Tue, 11 Sep 2012 09:19:18 GMT
View Forum Message <> Reply to Message

On Monday, 10 September 2012 17:26:30 UTC+2, Coyote wrote:
> Yngvar Larsen writes:
>
>> From my point of view, GRIDDATA is for gridding _irregular_ data, which is a hard problem. If your data is already on some regular grid, why would you want to triangulate? Regular interpolation is all that is needed if you do it the right way.
>
> It would be wonderful to give people a pointer to the "right way"
> then. I've apparently been doing it the "wrong way" because I
> can't get it to work properly, even though I have been working on
> the problem, off and on, for years.

** Wrong Way **
Transform your perfectly regular input grid points to some other projection. This will then be irregular in the output domain, and you will need to use the Ungodly slooow TRIANGULATE/GRIDDATA approach to get a nice and regular grid in the output domain.

** Right Way **
Start with a regular grid in the _output_ domain. Calculate where these grid points are located in the input domain. Since your input data are on a nice and regular grid, INTERPOLATE or something similar will do the job fast and efficiently!

Basically: it is easier to interpolate an irregular grid from a regular grid than the other way around!

I fiddled around yesterday evening with the example on your web page

http://www.idlcoyote.com/code_tips/usegriddata.html

My solution to the "GridData Conundrum" is simply "Don't use GRIDDATA"...

```
8<---------------------------
;; Read input data
dataFile = 'usegriddata.dat'
nx = 144
ny = 73
indata = fltarr(nx, ny)
openr, unit, dataFile, /get_lun
readu, unit, indata
free_lun, unit

;; Create IDL coordinate mapper for polar stereographic grid.
;; See http://nsidc.org/data/polar_stereo/ps_grids.html
re = 6378273d0
e = 0.081816153d0
```

```
rp = sqrt((1-e^2)*re^2)
map = map_proj_init('polar stereographic', /gctp, $
                    center_latitude=70,          $
                    center_longitude=315,        $
                    semimajor_axis=re,           $
                    semiminor_axis=rp)

;; Define output grid.
dx = 25d3
dy = 25d3
xmin = -3.85d6
ymin = -5.35d6
nxout = 304L
nyout = 448L
mapx = xmin + dx*dindgen(nxout)
mapy = ymin + dy*dindgen(nyout)

mapx = mapx[*,lindgen(nyout)]
mapy = transpose(mapy[*,lindgen(nxout)])

;; Calculate output map grid values in input coordinates
latlon = map_proj_inverse(mapx, mapy, map_structure=map)
lat = reform(latlon[1,*], nxout, nyout)
lon = reform(latlon[0,*], nxout, nyout)

;; Transform lat/lon values to input grid indices.
xind = ((lon + 360) mod 360)/2.5
yind = (90 - lat)/2.5

;; Nearest neighbour
outdata_nn = indata[round(xind), round(yind)]
;; Linear interpolation
outdata_li = interpolate(indata, xind, yind)
;; Cubic interpolation
outdata_ci = interpolate(indata, xind, yind, cubic=-0.5)
8<--------------------------
```

Note that I used a different output projection than you did in your code. I guess you used a spherical earth equirectangular grid instead of the NSIDC polar stereographic grid you mentioned in the text just for simplicity? With a 25 km grid, it does not really matter much.

>> 1) Divide the _output_ grid in manageable blocks. What "manageable" means, will depend on many things, but for satellite data mostly on memory limitations. Overlap between the blocks might be necessary if you are going to include filtering/interpolation.
>

> It would be useful to have a robust algorithm for doing this kind of
> chunking. Or, in the absence of that, at least some general rules of

> thumb that people could use. I've never seen anything written down
> about this.

Hard to make a general rule about this, but here are some advices.

Overlap between patches/blocks: typically as long as your worst case filter length.

Size of patches/blocks: usually limited by memory, but sometimes the algorithm itself requires small pathes, e.g. TRIANGULATE. I usually buffer the regridded output blocks corresponding to a full stripe in the x-direction before writing to file so I can do it with a singe POINT_LUN+WRITEU operation instead of a long slow loop. This may limit the block size in the y-direction to avoid a very large buffer. If the input and the output projections are very different, you also have to think about the size of a rectangular input data block that is "big enough" to cover your output block. The default value of the output block in my system is 256x256, which will usually end up with a fast enough result; Not too big, which will cause memory problems. A nice dimension for FFT-based algorithms to work well, and not too many patches, so the double loop over the loops will not slow down things too much.

Your mileage may vary, so these things should be tuned to your computing system.

> I appreciate the help. I'll see if I can find some time to have another
> go at this. I have been working on a "map_patch" alternative that
> sorta works. Perhaps I can fit these ideas into it in a reasonably
> robust fashion.

A "map_patch" approach is _very_ useful since it scales very well if you suddenly want to use your algorithm for a dataset that is too big to fit in memory, or too slow to work on large blocks. Just make sure the patch you are talking about is situated in our _output_ grid, not in the _input_ grid!

At work, we have implemented this idea in an object oriented, data driven system. Each processing module is a class that basically implements a method GETDATA, which takes as input 4 parameters XSIZE/YSIZE/XPOS/YPOS that describes a patch in the output grid. GETDATA will then retrieve whatever data it needs from the previous processing module, and do its job. Then there is a base class method WRITE that assumes that this method exists, divides the output grid in blocks (or a rectangular subset of the output grid), makes a double loop over calls to GETDATA and dumps the result to file. Blocksize/overlap are settable parameters of these objects, with default values usually set in the INIT method.

Note that we use this framework also for general data processing, not only remapping. I wrote a conference paper about this many years ago. (Regridding as such is not explicitly mentioned however.)

http://earth.esa.int/fringe05/proceedings/papers/427_larsen. pdf

--
Yngvar