Subject: Re: clever way to subregion an image?
Posted by R.G.Stockwell on Fri, 09 Apr 2010 18:50:16 GMT
View Forum Message <> Reply to Message

> "Paolo" <pgrigis@gmail.com> wrote in message
> news:929b2d07-c3ce-448e-b36d-0227a031b22e@20g2000vbr.googleg roups.com...
> On Apr 9, 12:05 pm, "R.G. Stockwell" <noem...@please.com> wrote:
>> I need to cut an image into 4 equal-size parts, which
>> obviously is very easy to do in a few lines.
>> image1 = im[0:nx/2-1, 0:ny/2-1]
>> image2 = im[0:nx/2-1, ny/2:*]
>> image3 = im[nx/2:*, 0:ny/2-1]
>> image4 = im[nx/2:*, ny/2:*]
>>
>> i came across a way to do this with reform, but
>> it required 4 steps (several reforms, a couple transposes)
>> to do it properly.
>>
>> I'd be interested (just for fun) in a vectorized general way to do this
>> if any of you 'dimension jugglers' have any clever ideas,
>> for how to take an image and cut it into 4, or 16, or 64,
>> or 256 equal pieces (that would probably be about the maximum)
>
> Well, if it is just for fun, why not use a recursive approach?
> I always like the simplicity of these :) (though they are not
> always the most efficient way, and sometimes they are the worst
> way to do it).
>
> res=segim(dist(512,512),level=4)
> IDL> help,res
> RES FLOAT = Array[8192, 32]
>
> The output is just an array with the images side by side, i.e.
> [im1,im2,...,im256].
>
> There are (2^level)^2 subarrays. That is,
>
> lev =1 -> 4
> lev =2 -> 16
> lev =3 -> 64
> lev =4 -> 256
> etc.
>
> Ciao,
> Paolo
>
> FUNCTION segim,im,level=level
>

```
>  IF n_elements(level) EQ 0 THEN level=4
>
>  n=size(im,/dimension)
>  nx=n[0]
>  ny=n[1]
>
>  IF level EQ 0 THEN return,im
>
>  return,[segim(im[0:nx/2-1, 0:ny/2-1],level=level-1),segim(im[0:nx/2-1,
>  ny/2:*],level=level-1), $
>         segim(im[nx/2:*  , 0:ny/2-1],level=level-1),segim(im[nx/2:*  ,
>  ny/2:*],level=level-1)]
>  END
```

very nice. I've always been a big fan of recursion. One thing i didn't like
about the recursion approach was that the ordering is a bit awkward, especially
for higher levels. I had originally wanted to start at the top right, and go
left to right, with the ordering.

But as i was looking at this, it occured to me that all i need to
do is create a latitude array, and a longitude array and segment those as well,
then I can just loop through and process everything in a straightforward manner.

I'd just point out that one could do a little juggle to break the arrays out with:

```
IDL> result2  =
 transpose(reform(transpose(result),nx/(2^level),ny/(2^level) ,(2^(2*level))),[1,0,2])
```

then the subarrays are

```
IDL>print,result2[*,*,0],result2[*,*,1], etc
```

cheers,
bob

PS this if for google earth network linking of images, at the varying resolutions.
You know how you zoom in from orbit, all the way down to your house and street.