Subject: Re: Display and Navigate Image in IDL 8.2
Posted by lecacheux.alain on Sat, 08 Sep 2012 21:07:47 GMT
View Forum Message <> Reply to Message

Le samedi 8 septembre 2012 01:05:06 UTC+2, David Fanning a écrit :
> David Fanning writes:
>
>
>
>> I am actually making progress on this now, using something very much
>
>> like this. I'm writing it up. Should be ready in an hour or so.
>
>> It only took me two full days of effort. A new speed record for
>
>> a function graphics plot!
>
>
>
> Success at last! Praise God Almighty! Success at last!
>
>
>
> Don't know why I'm channeling Martin Luther King, Jr,
>
> except that it has been a LONG day and I'm ready for
>
> a Friday afternoon beer! Have I mentioned how much I
>
> love Function Graphics.
>
>
>
> If you want to skip to the bottom line, you can run the
>
> code you find here:
>
>
>
> http://www.idlcoyote.com/tip_examples/mapnogrid.pro
>
>
>
> If you want to read all the gory details of how I figured
>
> this out, you can read this article:
>
>

>
> http://www.idlcoyote.com/ng_tips/mapnogrid.php
>
>
>
> If you just want to see the final result, here is the
>
> code:
>
>
>
> ; Get the image.
>
> googleStr = "http://maps.googleapis.com/maps/api/staticmap?" + $
>
> "center=40.6,-105.1&zoom=12&size=600x600" + $
>
> "&maptype=terrain&sensor=false&format=png32"
>
> netObject = Obj_New('IDLnetURL')
>
> void = netObject -> Get(URL=googleStr, FILENAME="googleimg.png")
>
> Obj_Destroy, netObject
>
> googleImage = Read_Image('googleimg.png')
>
>
>
>
>
> ; What I am trying to reproduce in Function Graphics, displayed
>
> ; with Coyote Graphics.
>
> centerLat = 40.6D
>
> centerLon = -105.1D
>
> scale = cgGoogle_MetersPerPixel(12)
>
> map = Obj_New('cgMap', 'Mercator', ELLIPSOID='WGS 84', /OnImage)
>
> uv = map -> Forward(centerLon, centerLat)
>
> uv_xcenter = uv[0,0]
>
> uv_ycenter = uv[1,0]

```
>
>    xrange = [uv_xcenter - (300*scale), uv_xcenter + (300*scale)]
>
>    yrange = [uv_ycenter - (300*scale), uv_ycenter + (300*scale)]
>
>    map -> SetProperty, XRANGE=xrange, YRANGE=yrange
>
>    cgDisplay, 700, 700, Title='Google Image with Coyote Graphics'
>
>    cgImage, googleImage, Margin=0.1
>
>    map -> Draw
>
>    cgMap_Grid, MAP=map, /BOX_AXES
>
>    cgPlotS, -105.1, 40.6, PSYM='filledstar', SYMSIZE=3.0, $
>
>       MAP=map, COLOR='red'
>
>
>
>    ; The code I used to do what I wanted to do. (Needs some
>
>    ; of the code from above.)
>
>    ll = map -> Inverse(xrange, yrange)
>
>    limits = [ll[0,0], ll[1,0], ll[0,1], ll[1,1]]
>
>    xdim = Abs(xrange[1] - xrange[0])
>
>    ydim = Abs(yrange[1] - yrange[0])
>
>    xloc = xrange[0]
>
>    yloc = yrange[0]
>
>    obj = Image(googleImage, MAP_PROJECTION='mercator', $
>
>      ELLIPSOID='WGS 84', GRID_UNITS=1, $
>
>      XRANGE=xrange, YRANGE=yrange, LIMIT=limit, $
>
>      DIMENSIONS=[700,700], POSITION=[0.1, 0.1, 0.9, 0.9], /BOX_AXES, $
>
>      IMAGE_DIMENSIONS=[xdim,ydim], IMAGE_LOCATION=[xloc,yloc])
>
>    sym = Symbol(centerLon, centerLat, 'star', /DATA, $
```

>
>   SYM_COLOR='red', SYM_SIZE=3, SYM_FILLED=1)
>
>   obj.mapprojection.mapgrid.BOX_AXES = 1
>
>   obj.mapprojection.mapgrid.BOX_THICK = 10
>
>   obj.mapprojection.mapgrid.LINESTYLE = 1
>
>   obj.mapprojection.mapgrid.GRID_LONGITUDE = 0.04
>
>   obj.mapprojection.mapgrid.GRID_LATITUDE = 0.03
>
>   obj.mapprojection.mapgrid.LABEL_POSITION = 0
>
>   obj.mapprojection.mapgrid.LONGITUDE_MIN=-105.18
>
>   obj.mapprojection.mapgrid.LATITUDE_MIN=40.54
>
>
>
> There is, apparently, no way to get box axes on the plot,
>
> but to get even this far in two days time is a major
>
> achievment! I'm going for a beer.
>
>
>
> I learned after I wrote the article that the LIMIT
>
> keyword is not needed. (I wouldn't have thought so,
>
> but I was throwing the kitchen sink at the problem!)
>
>
>
> Cheers,
>
>
>
> David
>
>
>
>
>
> --

>
> David Fanning, Ph.D.
>
> Fanning Software Consulting, Inc.
>
> Coyote's Guide to IDL Programming: http://www.idlcoyote.com/
>
> Sepore ma de ni thui. ("Perhaps thou speakest truth.")

David,

I continued to study your problem. Here is my present understanding.
I can see two very different steps in your problem, the second one, which deals with (NG) graphics usage, being not the hardest!

The image you get from the GoogleMaps API is already a projection, with definite horizontal and vertical linear scales in projected meter space. But if you know exactly what this projection is, you can indifferently use the meter space or the lat/lon space. Therefore one get a first step which consists in retrieving the right image parameters, then a second (graphical) step in which the image and the relevant map grid (a line plot with labels, etc...) have to be overlaid.

We start from the GoogleMaps request, which contains four parameters: center coordinates, zoom factor and image size in pixels.

```
;  googleStr = "http://maps.googleapis.com/maps/api/staticmap?" + $
;    "center=40.6000,-105.1000&zoom=12&size=600x600" + $
;    "&maptype=terrain&sensor=false&format=png32"
```

From GoogleMAPS API documentation, the projection is a simple conformal Mercator projection from the WGS84 ellipsoid with true scaling along the equator:

```
  clon = -105.1d0
  clat = 40.6d
  m = Map_Proj_Init('mercator', /GCTP, ELLIPSOID='WGS 84', $
    CENTER_LONGITUDE=clon, TRUE_SCALE_LATITUDE=0d)
```

The scale of the image, in meter/pixel, is given by the "google" zoom factor deduced from the lowest resolution image (zoom=0) which maps the entire Earth equator over 256 pixels. Other zoom factors form a decreasing series in successive powers of 2. Then:

```
  zoom = 12
  res = (2*!dpi*m.A)/256/2^zoom  ;m.A is the equatorial radius of the involved ellipsoid.
  print,res
;       38.218514   ;a slight difference with your value ?
```

Taking into account the requested image size (in pixels), the boundaries (four corners) of the image, both in meter space and lat/lon space, can be further computed:

```
  xy_ = [[-1,-1],[1,-1],[1,1],[-1,1]]*res*299.5
```

```
  cm = map_proj_forward(MAP=m, clon, clat)
  xy_ += cm # [1,1,1,1]
  xy = map_proj_inverse(MAP=m, xy_)
  print, xy
;     -105.20283      40.521578
;     -104.99717      40.521578
;     -104.99717      40.678329
;     -105.20283      40.678329
```

As this point, the right grid map can be built. (However, one thing is still missing: the altitude information. Indeed, GoogleMaps image display true coordinates (i.e. measured on the geoid), while above calculations are done on the ellipsoid. Since geoid and ellipsoid altitude differences remain within +/- 100m over the world, only the highest resolution images will be affected).

Now the second step: the graphics display.

First, the image, unlabelled. (The included margin is only needed because of overlaid grid annotations).

```
  img = IMAGE(googleimage, MARGIN=[0.05,0.15,0.05,0.05])
```

Second, the grid: just an overlay in the current window:

```
  mp = Map('Mercator', ELLIPSOID='WGS 84', $
    CENTER_LONGITUDE=clon, TRUE_SCALE_LATITUDE=0d, $
    LIMIT=[xy[1,0],xy[0,0],xy[1,-1],xy[0,1]], $
    GRID_LONGITUDE=1./20, GRID_LATITUDE=1./30, LINESTYLE=1, $
    LABEL_POSITION=0, $
    BOX_AXES=1, BOX_ANTIALIAS=1, $
    POSITION=[0.05,0.15,0.95,0.95], /CURRENT)
```

The MAP object rendering could be easily enhanced by using child objects like mp.MAPGRID, mp.MAPGRID.LONGITUDES, etc...
Note that the mp.MAPPROJECTION property defines exactly the same projection than the above Map_Proj_Init does; in fact it uses it; there is certainly a way to avoid such a duplicate calculation. I guess that the IMAGE function, throughout the MAP_PROJECTION keyword, is doing similar overlay, but I did not succeed in this way.

Quite simple, indeed ?

Cheers,

Alain