Subject: Re: FG question: retrieve points within polygon
Posted by Jim Pendleton on Thu, 04 Dec 2014 23:58:45 GMT
View Forum Message <> Reply to Message

On Thursday, December 4, 2014 8:41:26 AM UTC-7, Helder wrote:
> On Thursday, December 4, 2014 4:24:11 PM UTC+1, alx wrote:
>> On Thursday, December 4, 2014 3:32:26 PM UTC+1, Helder wrote:
>>> On Thursday, December 4, 2014 2:51:28 PM UTC+1, alx wrote:
>>>> On Thursday, December 4, 2014 10:50:42 AM UTC+1, Helder wrote:
>>>> > Hi,
>>>> > I'm looking for an easier way to get the indices inside a polygon or ellipse created in function graphics.
>>>> > So here is a basic example that states what I want to do:
>>>> >
>>>> > ;first generate the graphics
>>>> > img = dist(600)
>>>> > w = window(dimensions=[500,500])
>>>> > im = image(img, current=w)
>>>> > pl = polygon([0.25,0.75,0.75,0.25],[0.25,0.25,0.75,0.75],/norm,ta rget=im)
>>>> > ;make some changes to the polygon
>>>> > pl.rotate, 12
>>>> >
>>>> > ;now extract the mean value of the points of the image that are inside the polygon
>>>> >
>>>> > pl->getData, xx, yy
>>>> > o = obj_new('idlanroi', xx*600d, yy*600d, /double, type=2)
>>>> > mask = o->ComputeMask(dimensions=[600,600])
>>>> > obj_destroy, o
>>>> > pts = where(mask, cnt)
>>>> > meanVal = mean(img[pts])
>>>> > print, 'the mean value inside the polygon is ', meanVal
>>>> >
>>>> >
>>>> >
>>>> > So this method works fine. It's maybe not the most obvious, but works. Now the question is... How do I get the same result for an ellipse?
>>>> > Of course I could calculate the perimeter points of the ellipse and use the same method as above, but that would not really be... well ... cool.
>>>> >
>>>> > Any better way to do this? I couldn't find any FG method to get such info.
>>>> >
>>>> > Thanks,
>>>> > Helder
>>>>
>>>> If you could plot an ellipse with FG, you know its equation from the parameters (center, axes, orientation) you have given in the call. Let it be F(x,y)=0.
>>>> Then the indices of the (x,y) points inside the ellipse are those for which F(x,y) is strictly negative.

>>>> alx.
>>>
>>> Hi Alx,
>>> I wanted to avoid doing myself the calculation, but even trying I found that it is not that possible. It seems like the ellipse() function simply generates a polygon() function. Once created, I could not retrieve the center or radius (major or minor) and cannot therefore compute using the ellipse equation. What I can do is use the undocumented getData method as I would for a polygon and then proceed as if it were a polygon.
>>>
>>> Still, a mask method would be a nice add to the FG.
>>>
>>> Cheers,
>>> Helder
>>
>> Hi Helder,
>> If you draw the ellipse by calling the ELLIPSE function, you should know everything. If you draw it by hand, you can get the rectangle containing the ellipse by doing (after selecting it):
>>   pos = GetWindows(/CURRENT).GetSelect().Position
>> then, center and axis lengthes.
>> alx.
>
> Hi Alx,
> thanks.
> The idea is that I have a widget window where I move and change my ellipse as I wish.
> Say I start with this:
>
> img = dist(600)
> w = window(dimensions=[500,500])
> im = image(img, current=w)
> el = ellipse(300,300,/data, minor=100, major=150, target=im)
>
> and then I modify the ellipse with the mouse (make it bigger, rotate, stretch,...).
> The changes that will be made I don't know "a priori", so by stretching and rotating I end up with a completely different ellipse.
>
> I can get the position (as you said) with
>
> pos = el.position
>
> and that gives me always the center as [(pos[0]+pos[2])/2.0,(pos[1]+pos[3])/2.0]. But I cannot retrieve the rotation or the axis and, as far as I understand, there are infinite ellipses that can fit inside the rectangle defined by the position property that have a different set of rotation/axis.
>
> I think that I will have to stick with the polygon type solution using IDLanROI.
>
> Thanks,
> Helder

How adventurous do you want to be? The transformation information is stored in the container of the polygon. The coordinates of the raw data don't actually change after the polygon is created, I would guess.

Let's say you've started a new IDL session and executed the above commands then you've selected and resized or repositioned it interactively. (I'm using IDL 8.4 so YMMV.)

Which object that was created is an IDLgrPolygon?

```
IDL> oall = obj_valid(/cast)
IDL> print, where(obj_isa(oall, 'idlgrpolygon'))
       262       550       568       586       604       621       640       657       676      1894
  2089
```

We have a bunch. One is likely the "main" polygon and the others the "selection" items for resizing and so forth.

Which one is the "main" polygon? Let's guess the first and change its color.

```
IDL> oall[262]->setproperty, color = [0, 255, 0] & el.refresh
```

Lucky guess.

Next, let's get its container, which is by definition a model.

```
IDL> oall[262]->getproperty, parent = omodel
```

Check out the transformation matrix.

```
IDL> omodel->getproperty, transform = t
IDL> print, t
     0.51712623     0.00000000     0.00000000      55.899910
     0.00000000      1.3790130     0.00000000     -120.23054
     0.00000000     0.00000000      1.0000000     0.00000000
     0.00000000     0.00000000     0.00000000      1.0000000
```

Of course, you'll have different values here.

Let's send the ellipse back "home":

```
IDL> t = identity(4)
IDL> omodel->setproperty, transform = t & el.refresh
```

Teasing out the rotation, translation, and scale from the quaternion is an exercise left for the reader.

In function graphics there's a whole lot of cruft, er, framework, that sits over the top of the most basic graphics atoms and models and knowing how the basic, documented, objects work will go a

long way to helping you get closer to what you want.

If the new graphics API doesn't expose exactly the features you want, there are ways to get and modify the data and behaviors, with the possible side effect of producing inconsistent internal states, particularly during interactive use.

Run with scissors.

Jim P.
"I work for Exelis VIS, but these are my own observations"
