Subject: Understanding IDLanROI
Posted by KRDean on Fri, 08 Oct 2010 21:13:28 GMT
View Forum Message <> Reply to Message

I am attempting to use IDLanROI to determine if Shapefiles overlap another Shapefile.

Sometimes it works, sometimes it doesn't. Especially, if the polygon lies within the interior.

I put together a test program that retrieves the States and uses IDLanROI to determine if it lies within the US.

Why does Colorado not considered to be in the United States, among some other interior States?

Is IDLanROI not the IDL routine to use to perform this check?

Kelly Dean
Milliken, CO

```
;+
;
;
;
;
 ;------------------------------------------------------------ -----------
PRO ITT_PolyOverlap


states_path = FILEPATH('states.shp',SUBDIR=['resource', 'maps',
'shape' ])
country_path = FILEPATH('country.shp',SUBDIR=['resource', 'maps',
'shape' ])

PRINT, states_path
PRINT, country_path

oSHP = OBJ_NEW('IDLffShape', country_path )
ent1 = oSHP -> GetEntity( 34, /ATTRIBUTES ) ; USA
OBJ_DESTROY, oSHP

PRINT, ' -- Working with country : ', (*ent1.attributes).attribute_4

oSHP = OBJ_NEW('IDLffShape', states_path )
oSHP -> GetProperty, N_ENTITIES = num_ent
FOR staten = 0, num_ent DO BEGIN

  oSHP = OBJ_NEW('IDLffShape', states_path )
```

```
  ent0 = oSHP -> GetEntity( staten, /ATTRIBUTES )

  oROI = OBJ_NEW('IDLanROI', (*ent1.vertices)   )
  pttest = oROI -> ContainsPoints( (*ent0.vertices) )
  OBJ_DESTROY, oROI

  overlap = WHERE( ptTest GT 0, count )

  IF ( count GT 0 ) THEN BEGIN
    PRINT, ' === ' , (*ent0.attributes).attribute_1, ' inside ',
(*ent1.attributes).attribute_4, ' === ', count
  ENDIF ELSE BEGIN
    PRINT, ' <<< ' , (*ent0.attributes).attribute_1, ' outside ',
(*ent1.attributes).attribute_4, ' >>> ', count
  ENDELSE

ENDFOR

OBJ_DESTROY, oSHP

END
```

Subject: Re: Understanding IDLanROI
Posted by KRDean on Tue, 19 Oct 2010 15:02:43 GMT
View Forum Message <> Reply to Message

On Oct 8, 3:13 pm, kBob <krd...@gmail.com> wrote:
> I am attempting to use IDLanROI to determine if Shapefiles overlap
> another Shapefile.
>
> Sometimes it works, sometimes it doesn't. Especially, if the polygon
> lies within the interior.
>
> I put together a test program that retrieves the States and uses
> IDLanROI to determine if it lies within the US.
>
> Why does Colorado not considered to be in the United States, among
> some other interior States?
>
> Is IDLanROI not the IDL routine to use to perform this check?
>
> Kelly Dean
> Milliken, CO
>
> ;+
> ;
> ;

```
>  ;
>   ;------------------------------------------------------------ -----------
>  PRO ITT_PolyOverlap
>
>  states_path = FILEPATH('states.shp',SUBDIR=['resource', 'maps',
>  'shape' ])
>  country_path = FILEPATH('country.shp',SUBDIR=['resource', 'maps',
>  'shape' ])
>
>  PRINT, states_path
>  PRINT, country_path
>
>  oSHP = OBJ_NEW('IDLffShape', country_path )
>  ent1 = oSHP -> GetEntity( 34, /ATTRIBUTES ) ; USA
>  OBJ_DESTROY, oSHP
>
>  PRINT, ' -- Working with country : ', (*ent1.attributes).attribute_4
>
>  oSHP = OBJ_NEW('IDLffShape', states_path )
>  oSHP -> GetProperty, N_ENTITIES = num_ent
>  FOR staten = 0, num_ent DO BEGIN
>
>    oSHP = OBJ_NEW('IDLffShape', states_path )
>    ent0 = oSHP -> GetEntity( staten, /ATTRIBUTES )
>
>    oROI = OBJ_NEW('IDLanROI', (*ent1.vertices)   )
>    pttest = oROI -> ContainsPoints( (*ent0.vertices) )
>    OBJ_DESTROY, oROI
>
>    overlap = WHERE( ptTest GT 0, count )
>
>    IF ( count GT 0 ) THEN BEGIN
>      PRINT, ' === ' , (*ent0.attributes).attribute_1, ' inside ',
>  (*ent1.attributes).attribute_4, ' === ', count
>    ENDIF ELSE BEGIN
>      PRINT, ' <<< ' , (*ent0.attributes).attribute_1, ' outside ',
>  (*ent1.attributes).attribute_4, ' >>> ', count
>    ENDELSE
>
>  ENDFOR
>
>  OBJ_DESTROY, oSHP
>
>  END
```

To answer my own question ...

Yes, IDLanROI is NOT the IDL routine to perform this check.

After pondering on this for a couple of weeks, I figure what I needed to do.

IDLgrROI is the prefer IDL routine to use.

It is a tessellation thing, especially when dealing with State and Country Shapefiles that may contain gaps. You may get away with very simple polygons using IDLanROI when dealing with a handful of vertices, but IDLgrROI and IDLgrROIGroup are the routines you should be using along with incorporating IDLgrTessellator in the code.

The code below produces the desire results I was after.

Kelly Dean
Milliken, CO

P.S. I thought IDL 8.0 removed memory leaks? I had to destroy some Objects after a code run.

```
;+
;
; @file_comments
; <P>Looking for (State) polygons that overlap (Country) polygons.
;
 ;------------------------------------------------------------ -----------
PRO ITT_TessOverlap

COMPILE_OPT DEFINT32, STRICTARR

states_path = FILEPATH('states.shp',SUBDIR=['resource', 'maps',
'shape' ])
country_path = FILEPATH('country.shp',SUBDIR=['resource', 'maps',
'shape' ])
oUSA = OBJ_NEW('IDLffShape', country_path )
oUSA -> GetProperty, N_ENTITIES = num_ent
;entUSA = oUSA -> GetEntity( 2, /ATTRIBUTES ) ; CAN
entUSA = oUSA -> GetEntity( 34, /ATTRIBUTES ) ; USA
;entUSA = oUSA -> GetEntity( 73, /ATTRIBUTES ) ; MEX
PRINT, ' -- Working with country : ', (*entUSA.attributes).attribute_4
oROIGroup = OBJ_NEW( 'IDLgrROIGroup' )
oROI = OBJARR( entUSA.n_parts )
FOR ii = 0L, entUSA.n_parts-1 do begin
  IF (ii EQ entUSA.n_parts-1)THEN BEGIN
    startpoint = (*entUSA.parts)[ii]
    endpoint   = entUSA.n_vertices-1
  ENDIF ELSE BEGIN
    startpoint = (*entUSA.parts)[ii]
```

```
    endpoint   = (*entUSA.parts)[ii+1]-1
  ENDELSE
  data = (*entUSA.vertices)[*, startpoint:endpoint]
  oTess = OBJ_NEW('IDLgrTessellator')
  oTess -> AddPolygon, data
  x = oTess -> Tessellate(verts, polys)
  oROI[ii] = OBJ_NEW('IDLgrROI', data = verts )
  oROIGroup -> ADD, oROI[ii]
ENDFOR
oStates = OBJ_NEW('IDLffShape', states_path )
oStates -> GetProperty, N_ENTITIES = nstates
FOR staten = 0, nstates-1 DO BEGIN
  overlap = 0
  entState = oStates -> GetEntity( staten, /ATTRIBUTES )
  PRINT,  ' -- Working with state : ',
(*entState.attributes).attribute_1
  FOR ii = 0L, entState.n_parts-1 do begin
    IF (ii EQ entState.n_parts-1)THEN BEGIN
      startpoint = (*entState.parts)[ii]
      endpoint   = entState.n_vertices-1
    ENDIF ELSE BEGIN
      startpoint = (*entState.parts)[ii]
      endpoint   = (*entState.parts)[ii+1]-1
    ENDELSE
    data = (*entState.vertices)[*, startpoint:endpoint]
    oTess = OBJ_NEW('IDLgrTessellator')
    oTess -> AddPolygon, data
    x = oTess -> Tessellate(verts, polys)
    pttest = oROIGroup -> ContainsPoints( verts )
    npts = N_ELEMENTS( ptTest )
    FOR sym = 1, 3 DO BEGIN
      indx = WHERE( ptTest EQ sym, symcnt )
      IF ( symcnt GT 0 ) THEN BEGIN
        overlap = ( overlap GE 0 ) ? 1 : 0
      ENDIF
    ENDFOR
  ENDFOR
  IF ( overlap GT 0 ) THEN BEGIN
    PRINT, ' === ' , (*entState.attributes).attribute_1, ' inside ',
(*entUSA.attributes).attribute_4, ' === '
  ENDIF ELSE BEGIN
    PRINT, ' <<< ' , (*entState.attributes).attribute_1, ' outside ',
(*entUSA.attributes).attribute_4, ' >>> '
  ENDELSE
ENDFOR
OBJ_DESTROY, oROIGroup
FOR staten = 0, nstates-1 DO OBJ_DESTROY, oROI[staten]
PRINT, " -- That's all Folks!"
```

END

Subject: Re: Understanding IDLanROI
Posted by David Fanning on Tue, 19 Oct 2010 17:38:28 GMT
View Forum Message <> Reply to Message

kBob writes:

> To answer my own question ...
>
> Yes, IDLanROI is NOT the IDL routine to perform this check.
>
> After pondering on this for a couple of weeks, I figure what I needed
> to do.
>
> IDLgrROI is the prefer IDL routine to use.
>
> It is a tessellation thing, especially when dealing with State and
> Country Shapefiles that may contain gaps. You may get away with very
> simple polygons using IDLanROI when dealing with a handful of
> vertices, but IDLgrROI and IDLgrROIGroup are the routines you should
> be using along with incorporating IDLgrTessellator in the code.

Can you elaborate a bit on why this "is a tessellation thing"?
I don't really understand what that means. And, supposing
I did understand what it means, what would it have to do
with choosing IDLgrROI over IDLanROI. Isn't IDLgrROI a
subclass of IDLanROI?

Cheers,

David

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

Subject: Re: Understanding IDLanROI
Posted by KRDean on Tue, 19 Oct 2010 20:19:09 GMT
View Forum Message <> Reply to Message

On Oct 19, 11:38 am, David Fanning <n...@dfanning.com> wrote:

> kBob writes:
>> To answer my own question ...
>
>> Yes, IDLanROI is NOT the IDL routine to perform this check.
>
>> After pondering on this for a couple of weeks, I figure what I needed
>> to do.
>
>> IDLgrROI is the prefer IDL routine to use.
>
>> It is a tessellation thing, especially when dealing with State and
>> Country Shapefiles that may contain gaps. You may get away with very
>> simple polygons using IDLanROI when dealing with a handful of
>> vertices, but IDLgrROI and IDLgrROIGroup are the routines you should
>> be using along with incorporating IDLgrTessellator in the code.
>
> Can you elaborate a bit on why this "is a tessellation thing"?
> I don't really understand what that means. And, supposing
> I did understand what it means, what would it have to do
> with choosing IDLgrROI over IDLanROI. Isn't IDLgrROI a
> subclass of IDLanROI?
>
> Cheers,
>
> David
>
> --
> David Fanning, Ph.D.
> Fanning Software Consulting, Inc.
> Coyote's Guide to IDL Programming:http://www.dfanning.com/
> Sepore ma de ni thui. ("Perhaps thou speakest truth.")

In Object Graphics, polygons that are overlapping, disjointed, or self-intersecting are not handled very well by routines like IDLgrPolygon, thus IDLgrTessellator is used to convert polygons that are more suitable for IDLgrPolygon. From the IDL documentation, it converts the concave polygon into a convext polygon. Tessellation is the process of tiling a plane so there are no overlaps or gaps.

When I tried to use the US shapefile in IDLanROI (and IDLgrROI) there was a line running across the United States from the Northeast to Hawaii. Using IDLanROI, some States above this line were considered exterior polygons. See my original post. :(After tessellating, the line from the Northeast to Hawaii is gone, so all the States that fall in the United States polygon are considered interior polygons. :)

Your right, IDLgrROI inherits IDLanROI, thus allowing IDLgrROI to use IDLanROI classes, like ->ContainsPoints().

IDLgrTesselator is an IDLgr object class, thus I recommend the use of the IDLgr classes IDLgrROI and IDLgrROIGroup. I haven't tried, but I may at a later date to see if just the IDLanROI and IDLanROIGroup will give the desired results.

Check out the example below that I modified from IDL Help, it tries to draw an intersecting polygon in Object Graphics, with and without tessellation.

```
PRO TessAux, DoTess = DoTess

  x = [0,1,0,1]
  y = [0,0,1,1]

  colors = [[0,255,0],[0,255,0],[0,64,0],[0,64,0]]

  IF ( KEYWORD_SET( DoTess ) ) THEN BEGIN

    oTess = OBJ_NEW('IDLgrTessellator')
    colors = [[0,255,0],[0,255,0],[0,64,0],[0,64,0]]
    oTess -> AddPolygon, x, y, AUXDATA=colors
    result = oTess -> Tessellate(v, c, AUXDATA=aux)

    oPoly = OBJ_NEW('IDLgrPolygon', v, POLYGONS=c, VERT_COLORS=aux,
SHADING=1)

  ENDIF ELSE BEGIN

    oPoly = OBJ_NEW('IDLgrPolygon', x, y )

  ENDELSE

  XOBJVIEW, oPoly, /BLOCK
  IF ( OBJ_VALID( oTess ) ) THEN OBJ_DESTROY, oTess
  IF ( OBJ_VALID( oPoly ) ) THEN OBJ_DESTROY, oPoly

END
;---------------------------------------------------------
```

Kelly Dean
Milliken, CO

```
;+
;
;
;-----------------------------------------------------
PRO TessAux, DoTess = DoTess
```

```
  x = [0,1,0,1]
  y = [0,0,1,1]

  IF ( KEYWORD_SET( DoTess ) ) THEN BEGIN

    oTess = OBJ_NEW('IDLgrTessellator')
    colors = [[0,255,0],[0,255,0],[0,64,0],[0,64,0]]
    oTess -> AddPolygon, x, y, AUXDATA=colors
    result = oTess -> Tessellate(v, c, AUXDATA=aux)

    oPoly = OBJ_NEW('IDLgrPolygon', v, POLYGONS=c, VERT_COLORS=aux,
SHADING=1)

  ENDIF ELSE BEGIN

    oPoly = OBJ_NEW('IDLgrPolygon', x, y )

  ENDELSE

  XOBJVIEW, oPoly, /BLOCK
  IF ( OBJ_VALID( oTess ) ) THEN OBJ_DESTROY, oTess
  IF ( OBJ_VALID( oPoly ) ) THEN OBJ_DESTROY, oPoly

END
```

Subject: Re: Understanding IDLanROI
Posted by David Fanning on Tue, 19 Oct 2010 21:02:22 GMT
View Forum Message <> Reply to Message

Kelly writes:

> When I tried to use the US shapefile in IDLanROI (and IDLgrROI) there
> was a line running across the United States from the Northeast to
> Hawaii. Using IDLanROI, some States above this line were considered
> exterior polygons. See my original post. :(After tessellating, the
> line from the Northeast to Hawaii is gone, so all the States that fall
> in the United States polygon are considered interior polygons. :)

Ah, ha! Some real world data has somehow slipped into
the normally quite vanilla example data sets! Good to know.
Thanks for tracking this down.

> Your right, IDLgrROI inherits IDLanROI, thus allowing IDLgrROI to use
> IDLanROI classes, like ->ContainsPoints().
>
> IDLgrTesselator is an IDLgr object class, thus I recommend the use of

> the IDLgr classes IDLgrROI and IDLgrROIGroup. I haven?t tried, but I
> may at a later date to see if just the IDLanROI and IDLanROIGroup will
> give the desired results.

I didn't have time to check this today, but my
understanding is that IDLanROI is the "brains" of the
outfit, and IDLgrROI is the "muscle". In other words,
one is use for analysis, and the other is used for graphics.
I would be surprised if IDLanROI is not perfect for the job,
once the tesselation issue is out of the way.

Thanks for the update. Very informative and helpful.

Cheers,

David

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

Subject: Re: Understanding IDLanROI
Posted by guillermo.castilla.ca on Wed, 20 Oct 2010 17:01:32 GMT
View Forum Message <> Reply to Message

kBob writes:
> I am attempting to use IDLanROI to determine if Shapefiles overlap
> another Shapefile...
> When I tried to use the US shapefile in IDLanROI (and IDLgrROI) there
> was a line running across the United States from the Northeast to
> Hawaii.

I think what happened is that when you created those ROI objects, you
provided the entire set of vertices that define the shape of the US as
a single part, and then the last vertex of the ring defining the
conterminous US got connected with the 1st vertex in Hawaii, hence the
line you mention. I think the IDLanROIGroup::ContainsPoints method
should work properly if you create the corresponding object by adding
sequentially the parts in the US shapefile entity, providing neither
of the parts are holes (e.g., imagine that Colorado had successfully
opted out of the Union; the ring defining its shape would be a hole,
and its vertices would be listed counterclockwise in the shapefile
entity). Since afaik there are no holes in the US territory, the above
method should work without having to resort to the tessellator or

other graphic object classes.

But in a more general application (i.e., where there are polygons with holes), I wonder what would be the way to find out whether a point is inside a polygon that contain holes. The IDLanROIGroup cannot be used for that, as there is no way to specify that a given ROI within the groups is actually a hole. The easiest solution would be that the ITTVIS folks implement the IDLffShape::ContainsPoints method. Now that there is a larger interest to integrate image analysis with vector GIS, there is some hope that this will happen in a future release...

Guillermo

Subject: Re: Understanding IDLanROI
Posted by KRDean on Fri, 22 Oct 2010 14:16:01 GMT
View Forum Message <> Reply to Message

On Oct 20, 11:01 am, Guillermo
<guillermo.castilla.castell...@gmail.com> wrote:
> kBob writes:
>> I am attempting to use IDLanROI to determine if Shapefiles overlap
>> another Shapefile...
>> When I tried to use the US shapefile in IDLanROI (and IDLgrROI) there
>> was a line running across the United States from the Northeast to
>> Hawaii.
>
> I think what happened is that when you created those ROI objects, you
> provided the entire set of vertices that define the shape of the US as
> a single part, and then the last vertex of the ring defining the
> conterminous US got connected with the 1st vertex in Hawaii, hence the
> line you mention. I think the IDLanROIGroup::ContainsPoints method
> should work properly if you create the corresponding object by adding
> sequentially the parts in the US shapefile entity, providing neither
> of the parts are holes (e.g., imagine that Colorado had successfully
> opted out of the Union; the ring defining its shape would be a hole,
> and its vertices would be listed counterclockwise in the shapefile
> entity). Since afaik there are no holes in the US territory, the above
> method should work without having to resort to the tessellator or
> other graphic object classes.
>
> But in a more general application (i.e., where there are polygons with
> holes), I wonder what would be the way to find out whether a point is
> inside a polygon that contain holes. The IDLanROIGroup cannot be used
> for that, as there is no way to specify that a given ROI within the
> groups is actually a hole. The easiest solution would be that the
> ITTVIS folks implement the IDLffShape::ContainsPoints method. Now that
> there is a larger interest to integrate image analysis with vector

> GIS, there is some hope that this will happen in a future release...
>
> Guillermo

Your right, the Tessellator is not required. I just have to perform
the ContainsPoints() on the individual shapefile parts.

To find the polygons that are holes, I use the IDL routine POLY_AREA.
The /SIGNED keyword returns the area as either positive or negative,
which tells you if the vertices are counterclockwise or clockwise. All
the State polygons from states.shp are clockwise.

Kelly

Subject: Re: Understanding IDLanROI
Posted by Matt[2] on Mon, 25 Oct 2010 16:17:50 GMT
View Forum Message <> Reply to Message

On Oct 22, 8:16 am, kBob <krd...@gmail.com> wrote:

> To find the polygons that are holes, I use the IDL routine POLY_AREA.
> The /SIGNED keyword returns the area as either positive or negative,
> which tells you if the vertices are counterclockwise or clockwise. All
> the State polygons from states.shp are clockwise.
>

Kelly,

Thanks so much for this tip. It has saved me. So other people out
there. Is this the preferred method to determine the vertex's
direction? internal vs external?

This is more out of interest than need. The poly_area is working like
a charm for me.

Thanks,
Matt

--
Matthew Savoie - Manager of Science Programmers
National Snow and Ice Data Center
(303) 735-0785 http://nsidc.org

Subject: Re: Understanding IDLanROI
Posted by guillermo.castilla.ca on Wed, 27 Oct 2010 15:44:50 GMT

On Oct 25, 1:17 pm, Matt <sav...@nsidc.org> wrote:
> Is this the preferred method to determine the vertex's
> direction? internal vs external?

The below function will tell you whether a ring (i.e., a closed chain
of vertices defined by their x y coordinates) is a hole (i.e., is
ordered counterclockwise). It is based on the same principle than
Kelly's trick, but it uses only the first 3 vertices of the ring
(which form a triangle from which area can be computed), and therefore
it should be a little more efficient than the other method.

```
;+
; :Description: Determines whether a list of vertices forming a ring
is a hole
; (i.e., is ordered counterclockwise) or not (and then is clockwise)
;
; :Params:
;    xy : in, required, type="dblarr(2\, nVertices)"
;
; :Returns: 1, if the ring is a hole; 0, otherwise
; :Categories: IDLffShape
;-
FUNCTION ishole, xy

dims = SIZE(xy, /DIMENSIONS)
IF dims[0] NE 2 THEN MESSAGE, 'The xy array must have two columns'
IF dims[1] LE 2 THEN MESSAGE, 'xy must contain at least three
vertices'

answer = (xy[0,2] - xy[0,0]) * (xy[1,1] - xy[1,0]) - $
(xy[0,1] - xy[0,0]) * (xy[1,2] - xy[1,0]) LT 0 ? 1 : 0

RETURN, answer

END
```

Cheers

Guillermo

Subject: Re: Understanding IDLanROI
Posted by guillermo.castilla.ca on Mon, 01 Nov 2010 13:23:22 GMT

I made some tests with a real dataset (a shapefile with 3 million

vertices distributed in 25k rings, 6k of which are holes), and noticed
that the above function does not always return the same result as the
signed POLY_AREA. The reason is that when the first three vertices in
the ring correspond to a concave portion, the triangle they define
lies outside the area enclosed by the ring. Therefore the triangle is
traversed in the opposite direction, and the function returns the
wrong answer (e.g., that the ring is a hole when is not). So I have
modified the ISHOLE function (below) so that it selects, rather than
the 1st 3 vertices in the ring, the northernmost (i.e., top) vertex
and its predecessor and successor. This way the triangle is guaranteed
to lie inside the ring. If the top vertex happens to be also the
first, then ISHOLE has to make sure that the last vertex is not at the
same height, otherwise the three selected vertices could be aligned
and no triangle would be formed.

After these changes, the function is not much more efficient than
using the SIGNED kw of POLY_AREA (0.39 s for the above dataset, while
using POLY_AREA took 0.48 s), so I don't think is really worthy. But
since I took the pain of writing it, here it goes, in case there is
someone out there who has to deal with larger shapefiles for which
this would make a difference.

Cheers

Guillermo

```
;+
; :Description: Determines whether a list of vertices forming a
ring
; is a hole(i.e., is ordered counterclockwise) or not (and then the
; ring is ordered clockwise)
;
; :Params:
;    verts : in, required, type="dblarr(2\, nVertices)"
;
; :Returns: 1, if the ring is a hole; 0, otherwise
; :Categories: IDLffShape
; :History: G.Castilla, Nov 2010. Written as a more efficient
; alternative to using the SIGNED keyword of the POLY_AREA
; function for the same purpose
;-
FUNCTION ishole, verts

dims = SIZE(verts, /DIMENSIONS)
IF dims[0] NE 2 THEN MESSAGE, $
'The verts array must have two columns'
IF dims[1] LE 3 THEN MESSAGE, $
'The input array must contain at least four rows'
  ; NB. Since the 1st and last vertices in the list are the
```

```
  ; same, verts must have at least 4 rows to form an area

ymax = MAX(verts[1,*],top)
IF top NE 0 THEN xy = verts[*,top-1:top+1] $
  ; NB. By using the top vertex, the resulting triangle
  ; is guaranteed to lie inside the ring

ELSE BEGIN ; if the the top vertex is the 1st vertex, make
          ; sure that the selected vertices form a triangle
  pos= dims[1]-2
  last = verts[*,pos]
  WHILE last[1] EQ ymax DO BEGIN
    pos=pos-1
    IF pos NE 0 THEN last = verts[*,pos] ELSE $
    MESSAGE, 'All vertices are in a horizontal line'
  ENDWHILE
  xy = [[last],[verts[*,0:1]]]
ENDELSE

answer = (xy[0,2] - xy[0,0]) * (xy[1,1] - xy[1,0]) - $
(xy[0,1] - xy[0,0]) * (xy[1,2] - xy[1,0]) LT 0 ? 1 : 0
  ; NB. The above expression is (minus) the determinant of the
  ; triangle coordinates, and is negative if the vertices of
  ; the triangle are listed in counterclockwise order

RETURN, answer

END
```

Subject: Re: Understanding IDLanROI
Posted by guillermo.castilla.ca on Mon, 01 Nov 2010 13:24:16 GMT
View Forum Message <> Reply to Message

I made some tests with a real dataset (a shapefile with 3 million vertices distributed in 25k rings, 6k of which are holes), and noticed that the above function does not always return the same result as the signed POLY_AREA. The reason is that when the first three vertices in the ring correspond to a concave portion, the triangle they define lies outside the area enclosed by the ring. Therefore the triangle is traversed in the opposite direction, and the function returns the wrong answer (e.g., that the ring is a hole when is not). So I have modified the ISHOLE function (below) so that it selects, rather than the 1st 3 vertices in the ring, the northernmost (i.e., top) vertex and its predecessor and successor. This way the triangle is guaranteed to lie inside the ring. If the top vertex happens to be also the first, then ISHOLE has to make sure that the last vertex is not at the same height, otherwise the three selected vertices could be aligned

and no triangle would be formed.

After these changes, the function is not much more efficient than
using the SIGNED kw of POLY_AREA (0.39 s for the above dataset, while
using POLY_AREA took 0.48 s), so I don't think is really worthy. But
since I took the pain of writing it, here it goes, in case there is
someone out there who has to deal with larger shapefiles for which
this would make a difference.

Cheers

Guillermo

```
;+
; :Description: Determines whether a list of vertices forming a
ring
; is a hole(i.e., is ordered counterclockwise) or not (and then the
; ring is ordered clockwise)
;
; :Params:
;    verts : in, required, type="dblarr(2\, nVertices)"
;
; :Returns: 1, if the ring is a hole; 0, otherwise
; :Categories: IDLffShape
; :History: G.Castilla, Nov 2010. Written as a more efficient
; alternative to using the SIGNED keyword of the POLY_AREA
; function for the same purpose
;-
FUNCTION ishole, verts

dims = SIZE(verts, /DIMENSIONS)
IF dims[0] NE 2 THEN MESSAGE, $
'The verts array must have two columns'
IF dims[1] LE 3 THEN MESSAGE, $
'The input array must contain at least four rows'
  ; NB. Since the 1st and last vertices in the list are the
  ; same, verts must have at least 4 rows to form an area

ymax = MAX(verts[1,*],top)
IF top NE 0 THEN xy = verts[*,top-1:top+1] $
  ; NB. By using the top vertex, the resulting triangle
  ; is guaranteed to lie inside the ring

ELSE BEGIN ; if the the top vertex is the 1st vertex, make
          ; sure that the selected vertices form a triangle
  pos= dims[1]-2
  last = verts[*,pos]
  WHILE last[1] EQ ymax DO BEGIN
    pos=pos-1
```

```
    IF pos NE 0 THEN last = verts[*,pos] ELSE $
    MESSAGE, 'All vertices are in a horizontal line'
  ENDWHILE
  xy = [[last],[verts[*,0:1]]]
ENDELSE

answer = (xy[0,2] - xy[0,0]) * (xy[1,1] - xy[1,0]) - $
(xy[0,1] - xy[0,0]) * (xy[1,2] - xy[1,0]) LT 0 ? 1 : 0
  ; NB. The above expression is (minus) the determinant of the
  ; triangle coordinates, and is negative if the vertices of
  ; the triangle are listed in counterclockwise order

RETURN, answer

END
```