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
```