Subject: Re: Is a point in a 4 vertex polygon?
Posted by Mark Hadfield on Mon, 15 Feb 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Nando Iavarone wrote in message <36C4D843.D9CA658A@acsys.it>...
> does anyone know a function that check if
> a given point is inside a 4 vertex polygon?

Function POLY_INSIDE (below) does this, for an n-vertex polygon.
I don't know how robust it is; please test it thoroughly before relying on
it.

--
Mark Hadfield, m.hadfield@niwa.cri.nz http://www.niwa.cri.nz/~hadfield/
National Institute for Water and Atmospheric Research
PO Box 14-901, Wellington, New Zealand

```
;+
; NAME:
;   POLY_INSIDE
;
; PURPOSE:
;   This function determines if points are inside/outside a polygon.
;
; CATEGORY:
;   Geography, Geometry.
;
; CALLING SEQUENCE:
;   Result = POLY_INSIDE(X, Y, XP, YP)
;
; INPUTS:
;   X,Y:       X, Y position(s) defining the point(s) to be tested. Can
;              be vectors.
;
;   XP,YP:     Vectors of X, Y positions defining the polygon.
;
; INPUT KEYWORDS:
;   EDGE:      Set this keyword to accept edge (& vertex) points as
;              "inside". Default is to reject them. Note that this feature
;              is still not implemented properly.
;
; OUTPUTS:
;   The function returns 0 if the point is outside the polygon, 1 if
;   it is inside the polygon.
;
; PROCEDURE:
;   This routine uses the fact that the sum of angles between
;   lines from a point to successive vertices of a polygon is 0
```

```
;   for a point outside the polygon, and +/- 2*pi for a point
;   inside. A point on an edge will have one such succesive angle
;   equal to +/- pi.
;
; EXAMPLE:
;
; MODIFICATION HISTORY:
;   Mark Hadfield, June 1995:
;       Written based on ideas in MATLAB routine INSIDE.M in the WHOI
;       Oceanography Toolbox  v1.4 (R. Pawlowicz, 14 Mar 94,
;       rich@boreas.whoi.edu).
;-
function POLY_INSIDE, X, Y, XP, YP, EDGE_INSIDE=edge

    ; Check geometry of the array of points. Note that this array
    ; is treated as a 1-D array internally to allow matrix operations,
    ; but the array structure and shape is restored on output.
    n = n_elements(X)  &  s = size(x)

    ; Make a local copy of polygon data
    ; If necessary, add a last point to close the polygon
    xpp = xp(*)  &  ypp = yp(*)  &  npp = n_elements(xpp)
    if xpp(npp-1) ne xpp(0) or ypp(npp-1) ne ypp(0) then begin
        xpp = [xpp,xpp(0)]  &  ypp = [ypp,ypp(0)]  &  npp = npp+1
    endif

    case npp of

        1:  if keyword_set(edge) then $
                return, fix(x eq xpp(0) and y eq ypp(0)) $
            else $
                return, reproduce(0,x)

     else:  begin

                ; The following matrix operations return (npp,n) arrays
containing
                ; distances from points to vertices
                dx = xpp#replicate(1D0,n) - replicate(1D0,npp)#x(*)
                dy = ypp#replicate(1D0,n) - replicate(1D0,npp)#y(*)

                angles = (atan(dy,dx))
                angles = angles(1:npp-1,*)-angles(0:npp-2,*)
                oor = where(angles le -!dpi,count)
                if count gt 0 then angles(oor) = angles(oor) + 2.*!dpi
                oor = where(angles gt !dpi,count)
                if count gt 0 then angles(oor) = angles(oor) - 2.*!dpi
```

```
        return,round(total(angles/!dpi,1)) ne 0

      end

   endcase

end
```