Subject: Re: point inside polygon
Posted by Philippe Peeters on Wed, 01 Apr 1998 08:00:00 GMT
View Forum Message <> Reply to Message

Alex Schuster wrote:
>
> William Connolley wrote:
>
>> In article C0684CDB@oma.be, Philippe Peeters <philp@oma.be> writes:
>>> Does anybody knows of an IDL function to test whether a given point is
>>> inside a polygon?
>>
>> I needed to solve this recently (in a mapping context). The solution I came up
>> with works but its not elegant: use poly_fill to actually draw your polygon
>> (in a pixmap not the screen window if you prefer), then read off the pixel value
>> of your point to see if its in or out.
>>
>> This is grotesquely inelegant, but its very simple and it works. I can
>> post the code if you're interested. A better solution
>> would be to look at polyfill and see how it does the fill... but sadly
>> polyfill seems to be one of the few routines not written in IDL.
>
> POLYFILLV works similar, but does not need a pixmap. It just returns the
> subscripts of all points inside the polygon.
> This worked okay for me, but for floating point coordinates it might not
> be too accurate.

This is precisely my problem. POLYFILLV is ok to check regular grid
points within a given polygon. In my case I have a polygon with real
coordinates (a satellite pixel) and I want to check if a ground station
(lat,lon) is within the pixel.

I have tried to adapt a C code from Graphic Gems
http://www.acm.org/tog/GraphicsGems/
which is the CrossingMultiply from Haines
in C:

```
/* ======= Crossings Multiply algorithm
=================================== */

/*
 * This version is usually somewhat faster than the original published
in
 * Graphics Gems IV; by turning the division for testing the X axis
crossing
 * into a tricky multiplication test this part of the test became
faster,
 * which had the additional effect of making the test for "both to left
```

```
or
 * both to right" a bit slower for triangles than simply computing the
 * intersection each time.  The main increase is in triangle testing
speed,
 * which was about 15% faster; all other polygon complexities were
pretty much
 * the same as before.  On machines where division is very expensive
(not the
 * case on the HP 9000 series on which I tested) this test should be
much
 * faster overall than the old code.  Your mileage may (in fact, will)
vary,
 * depending on the machine and the test data, but in general I believe
this
 * code is both shorter and faster.  This test was inspired by
unpublished
 * Graphics Gems submitted by Joseph Samosky and Mark Haigh-Hutchinson.
 * Related work by Samosky is in:
 *
 * Samosky, Joseph, "SectionView: A system for interactively specifying
and
 * visualizing sections through three-dimensional medical image data",
 * M.S. Thesis, Department of Electrical Engineering and Computer
Science,
 * Massachusetts Institute of Technology, 1993.
 *
 */

/* Shoot a test ray along +X axis.  The strategy is to compare vertex Y
values
 * to the testing point's Y and quickly discard edges which are entirely
to one
 * side of the test ray.  Note that CONVEX and WINDING code can be added
as
 * for the CrossingsTest() code; it is left out here for clarity.
 *
 * Input 2D polygon _pgon_ with _numverts_ number of vertices and test
point
 * _point_, returns 1 if inside, 0 if outside.
 */
int CrossingsMultiplyTest( pgon, numverts, point )
double pgon[][2] ;
int numverts ;
double point[2] ;
{
register int j, yflag0, yflag1, inside_flag ;
register double ty, tx, *vtx0, *vtx1 ;
```

```
    tx = point[X] ;
    ty = point[Y] ;

    vtx0 = pgon[numverts-1] ;
    /* get test bit for above/below X axis */
    yflag0 = ( vtx0[Y] >= ty ) ;
    vtx1 = pgon[0] ;

    inside_flag = 0 ;
    for ( j = numverts+1 ; --j ; ) {

  yflag1 = ( vtx1[Y] >= ty ) ;
  /* Check if endpoints straddle (are on opposite sides) of X axis
   * (i.e. the Y's differ); if so, +X ray could intersect this edge.
   * The old test also checked whether the endpoints are both to the
   * right or to the left of the test point.  However, given the faster
   * intersection point computation used below, this test was found to
   * be a break-even proposition for most polygons and a loser for
   * triangles (where 50% or more of the edges which survive this test
   * will cross quadrants and so have to have the X intersection computed
   * anyway).  I credit Joseph Samosky with inspiring me to try dropping
   * the "both left or both right" part of my code.
   */
  if ( yflag0 != yflag1 ) {
      /* Check intersection of pgon segment with +X ray.
       * Note if >= point's X; if so, the ray hits it.
       * The division operation is avoided for the ">=" test by checking
       * the sign of the first vertex wrto the test point; idea inspired
       * by Joseph Samosky's and Mark Haigh-Hutchinson's different
       * polygon inclusion tests.
       */
      if ( ((vtx1[Y]-ty) * (vtx0[X]-vtx1[X]) >=
       (vtx1[X]-tx) * (vtx0[Y]-vtx1[Y])) == yflag1 ) {
   inside_flag = !inside_flag ;
      }
  }

  /* Move to the next pair of vertices, retaining info as possible. */
  yflag0 = yflag1 ;
  vtx0 = vtx1 ;
  vtx1 += 2 ;
    }

    return( inside_flag ) ;
}
```

I code it in IDL as

```
function ptpoly,pgonx,pgony,x,y

numverts=n_elements(pgonx)
if numverts ne n_elements(pgony) then message,'X & Y must have same
size'
if numverts lt 3 then message,'At least 3 vertex'

   tx = x
   ty = y

   vtx0x = pgonx[numverts-1]
   vtx0y = pgony[numverts-1]
   ; get test bit for above/below X axis
   yflag0 = ( vtx0y ge ty ) ;
   vtx1x = pgonx[0]
   vtx1y = pgony[0]

   inside_flag = 0
   for j = 1,numverts-1 do begin

 yflag1 = ( vtx1y ge ty ) ;

 if  yflag0 ne yflag1 then begin

    if ( ((vtx1y-ty) * (vtx0x-vtx1x) ge $
     (vtx1x-tx) * (vtx0y-vtx1y)) eq yflag1 ) then $
  inside_flag = not(inside_flag)

 endif

 yflag0 = yflag1
 vtx0x = vtx1x
 vtx0y = vtx1y
 vtx1x = pgonx[j]
 vtx1y = pgony[j]
   endfor

   return,inside_flag
end
```

I tried it with a square px=[0,1,1,0] and py=[0,0,1,1] and random points with 0<x<2 and 0<y<2. It works quite well with that polygon but fail if I rotate the polygon vertices using shift(px,1) and shift(px,2). The polygon is the same, only the vertices ordering has changed.

But...

Now I have tried Crossing algorithm. this one seems to work. The only thing is that it can gives a "divide by zero" when two vertices have the same Y coordinates. In my case of satellite pixels, align vertices are very improbable.

--

Philippe Peeters
-- ------------
Belgian Institute for Space Aeronomy Tel: +32-2-373.03.81
Institut d'Aeronomie Spatiale de Belgique Fax: +32-2-374.84.23
3 Avenue Circulaire Email: philp@oma.be
B-1180 Brussels, Belgium http://www.oma.be/BIRA-IASB/