Subject: Re: determining if a point is "inside" or "outside" a shape
Posted by Andrew Kraus on Mon, 18 Oct 1999 07:00:00 GMT
View Forum Message <> Reply to Message

A method I have used extensively in the past is less elegant, but works like a charm. Create a pixmap of the entire window. Paint it black (ERASE), or any intensity vaue for that matter, then POLYFILL your coordinates with a different color. You can then check the color of the point you are testing. Be sure to color the coordinates (boundary) as well so you know if the point lies on the boundary itself. Pixmaps and polyfill are 2 of the faster functions in IDL, so this goes very quickly. You can see how this handles any shape, except donuts require a bit more thought.

Best of luck,

AK

dmarshall@ivory.trentu.ca wrote in message ...
> I have an interesting problem that I'm betting someone has solved already
> in some form or another.
>
> I have a set of x,y coordinates that describe a polygon, 2d in my case.
> How do I tell if another x,y point is inside the boundary of my polygon?
>
> I had an idea to take the center of my polygon and extend a line from it to
> the point and beyond, seeing if I cross the perimeter an odd number of
> times.
>
> Any other suggestions?
>
> Dave.
>
> David Marshall   Physics Dept. Trent University
> dmarshall@remove.this.ivory.trentu.ca   Peterborough   Ontario   Canada
>

Subject: Re: determining if a point is "inside" or "outside" a shape
Posted by Med Bennett on Mon, 18 Oct 1999 07:00:00 GMT
View Forum Message <> Reply to Message

I have solved this problem with the included two functions. It works by first creating a point outside the polygon, then creating a line from the outside point
to the test point. Then, the algorithm counts the number of intersections between the test line and the lines making up the polygon. An odd number

indicates that the test point is inside the polygon, while an even number
indicates a point outside the polygon. This method ensures a correct result even
in the case of polygons with concave sides, etc. The function inpoly requires the
line_int function, and returns an array with the same number of elements as the

number of data points, with a 1 for points inside the polygon and 0 for points outside the polygon. Hope that this helps and that you can decipher my lousy code. There is a problem if two adjacent points have the same X coord (slope of one line segment is infinite) - this needs to be fixed!

```
=================================
function inpoly,polyxy,dataxy

; procedure to determine which points in dataxy
; fall inside a given polygon
; polygon is defined by series of vertices given
; in polyxy

polyxy = double(polyxy)
dataxy = double(dataxy)

;determine polygon min and max xand y values

pxmax = max(polyxy(0,*))
pxmin = min(polyxy(0,*))
pymax = max(polyxy(1,*))
pymin = min(polyxy(1,*))

;determine size of input arrays

dsize = size(dataxy)
if dsize(0) eq 1 then dpts = 1 else dpts = dsize(2)
datain = intarr(dpts)

psize = size(polyxy)
ppts = psize(2)
int = intarr(ppts)

; determine which points lie inside polygon bounding box

w = where((dataxy(0,*) lt pxmax) and (dataxy(0,*) gt pxmin) and $
  (dataxy(1,*) lt pymax) and (dataxy(1,*) gt pymin),c)

for i = 0,c-1 do begin ;loop through data

   testpt = [pxmin-(.1*(pxmax-pxmin)),pymax+(.1*(pymax-pymin))]
   l1 = [[testpt],[dataxy(*,w(i))]] ;line 1 is from testpt to data point
```

```
;loop through polygon edge segments - last point in polygon array must be same
as
first!

    for j = 0,ppts-2 do begin
      if ( polyxy(0,j) ne polyxy(0,j+1) or polyxy(1,j) ne polyxy(1,j+1) ) then
begin
        l2 = [[polyxy(*,j)],[polyxy(*,j+1)]] ;second line connects two vertices

        int(j) = line_int(l1,l2) ;calculate line intersection
      endif
    endfor

    nint = total(int) ;total no. of intersections

;even if point is outside polygon; odd if inside
    if (nint/2 eq fix(nint/2)) then datain(w(i)) = 0 $
                           else datain(w(i)) = 1

endfor
print,total(datain),' points inside polygon.'
return,datain

end

====================================
function line_int,l1,l2

; calculate line equations

; print,l1,l2

if ( l1(0,0) ne l1(0,1) ) then begin
  slope1 = (l1(1,1)-l1(1,0))/(l1(0,1)-l1(0,0))
  yint1 = l1(1,0) - slope1*l1(0,0)
;  print,'m1',slope1,'b1',yint1
endif else begin
  if ((l1(0,0) lt max(l2(0,*))) and (l1(0,0) gt min(l2(0,*)))) then return,1 $
  else return,0
endelse

if ( l2(0,0) ne l2(0,1) ) then begin
  slope2 = (l2(1,1)-l2(1,0))/(l2(0,1)-l2(0,0))
  yint2 = l2(1,0) - slope2*l2(0,0)
;  print,'m2',slope2,'b2',yint2
endif else begin
  if ((l2(0,0) lt max(l1(0,*))) and (l2(0,0) gt min(l1(0,*)))) then return,1 $
```

```
  else return,0
endelse

; if lines are parallel, no intersection

if (slope1 ne slope2) then begin

xintersect = (yint2-yint1)/(slope1-slope2)
;print,'xintersect',xintersect
endif else return,0

if ( (xintersect lt max(l1(0,*))) and (xintersect gt min(l1(0,*))) $
 and (xintersect lt max(l2(0,*))) and (xintersect gt min(l2(0,*)))) then
return,1
$
    else return,0


end
```
====================================

"Martin LUETHI GL A8.1 2-4092" wrote:

> Dear all
>
> Is there a simple and fast way to find the indices of array elements, which
> are inside of a boundary (in 2 dimensions). Let's say coord(npoints, 2) is an

> array of coordinates in the plane and bound(nbound, 2) is the array of a
> bounday polygon. The coordintes are not on a grid (otherwise one could use
> polyfillv (PV-Wave).
>
> Thank you for any suggestion!
>
> Martin
>
> --
> ==
> Martin Luethi Tel. +41 1 632 40 92
> Glaciology Section Fax. +41 1 632 11 92
> VAW ETH Zuerich
> CH-8092 Zuerich mail luthi@vaw.baum.ethz.ch
> Switzerland
> ==

dmarshall@ivory.trentu.ca wrote:

> I have an interesting problem that I'm betting someone has solved already

> in some form or another.
>
> I have a set of x,y coordinates that describe a polygon, 2d in my case.
> How do I tell if another x,y point is inside the boundary of my polygon?
>
> I had an idea to take the center of my polygon and extend a line from it to
> the point and beyond, seeing if I cross the perimeter an odd number of
> times.
>
> Any other suggestions?
>
> Dave.
>
> David Marshall   Physics Dept. Trent University
> dmarshall@remove.this.ivory.trentu.ca   Peterborough   Ontario   Canada

---

Subject: Re: determining if a point is "inside" or "outside" a shape
Posted by Job von Rango on Thu, 21 Oct 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Ther is another analytic way to solve the problem:

How to determine wether a point lies inside an arbitrary 2-dimensional
polygon
or not?

IDEA:
Look at the sum of all angles tended between all pairs of neighboured
vertics
the polygon and the given point.

IN DETAIL:
Assume we have the n vertices of polgons ordered at the positions:
v_1, v_2, ... ,v_n
The given point is denoted by p.

Now connect the given point p with all n neighboured
vertices, and get the n vectors beginning in p and ending
in the vertices:

vec_1 = vector(p_0, v_1)
...
vec_n = vector(p_0, v_n)

Now add all n angles tended between the 2 vectors vec_i and vec_(i+1):

angle_i = angle(vec_i,vec_(i+1))

and calculate the sum:

```
            n
anglesum = sum  angle_i
           i=1
```

(Use vec_1 again for vec_(n+1) in the last angle_n)

RESULT:

```
            _
           |   2*pi        inside
anglesum = -|         if p is       the polygon
           |_   0          outside
```

IMPORTANT:
Take into account the correct sign of the angles
and use the vertices in ascending order!
Use e.g. the vector crossproduct:

vec_i x vec_(i+1)

in order to retrieve the absolut value |...| for the angle:

|angle_i| = inv sin(|vec_i x vec_(i+1)|)

The valid sign for angle_i is given by looking at the scalar product
of the crossproduct from above and a fixed vector vec_perp, perpendicular
to the area of the polygon:

```
           vec_perp * (  vec_i x vec_(i+1)  )
sign_i =   ------------------------------------
          | vec_perp * (  vec_i x vec_(i+1)  ) |
```

Now we get the correct value for all angles:

angle_i = sign_i * |angle_i|

REMARK:
Examine the case of a given point inside the polygon, but very near
to the edge between two vertices v_i and v_(i+1). The contributing
angle will be:

angle_i = + pi - epsilon

(where epsilon denotes a small positive angle.)
If we shift p over the edge to the outside of the polygon
(but very near again to the edge)
the contributing angle will be:

angle_i = - pi + epsilon'

The switch of the sign is the reason for the discontinuity (2*pi <-> 0)
of anglesum for points moving from inside to the outside of the polygon!

-- --
Job v. Rango   Medizinische Klinik I
Dipl.-Phys.   D-52074 Aachen
Pauwelsstrasse 30
Tel. : 049 / (0)241 / 80-89832
Fax : 049 / (0)241 / 88-88414
Email: jran@pcserver.mk1.rwth-aachen.de

Subject: Re: determining if a point is "inside" or "outside" a shape
Posted by Job von Rango on Thu, 21 Oct 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Ther is another analytic way to solve the problem:

How to determine wether a point lies inside an arbitrary 2-dimensional
polygon
or not?

IDEA:
Look at the sum of all angles tended between all pairs of neighboured
vertics
the polygon and the given point.

IN DETAIL:
Assume we have the n vertices of polgons ordered at the positions:
v_1, v_2, ... ,v_n
The given point is denoted by p.

Now connect the given point p with all n neighboured
vertices, and get the n vectors beginning in p and ending
in the vertices:

vec_1 = vector(p_0, v_1)
...
vec_n = vector(p_0, v_n)

Now add all n angles tended between the 2 vectors vec_i and vec_(i+1):

```
angle_i = angle(vec_i,vec_(i+1))
```

and calculate the sum:

```
             n
anglesum = sum  angle_i
            i=1
```

(Use vec_1 again for vec_(n+1) in the last angle_n)

RESULT:

```
              _
             |   2*pi        inside
anglesum = -|           if p is        the polygon
             |_   0          outside
```

IMPORTANT:
Take into account the correct sign of the angles
and use the vertices in ascending order!
Use e.g. the vector crossproduct:

```
vec_i x vec_(i+1)
```

in order to retrieve the absolut value |...| for the angle:

```
|angle_i| = inv sin( |vec_i x vec_(i+1)| )
```

The valid sign for angle_i is given by looking at the scalar product
of the crossproduct from above and a fixed vector vec_perp, perpendicular
to the area of the polygon:

```
              vec_perp * (  vec_i x vec_(i+1)  )
sign_i =   -------------------------------------
            | vec_perp * (  vec_i x vec_(i+1)  ) |
```

Now we get the correct value for all angles:

```
angle_i = sign_i * |angle_i|
```

REMARK:
Examine the case of a given point inside the polygon, but very near
to the edge between two vertices v_i and v_(i+1). The contributing

angle will be:

angle_i = + pi - epsilon

(where epsilon denotes a small positive angle.)
If we shift p over the edge to the outside of the polygon
(but very near again to the edge)
the contributing angle will be:

angle_i = - pi + epsilon'

The switch of the sign is the reason for the discontinuity (2*pi <-> 0)
of anglesum for points moving from inside to the outside of the polygon!

-- --
Job v. Rango   Medizinische Klinik I
Dipl.-Phys.   D-52074 Aachen
Pauwelsstrasse 30
Tel. : 049 / (0)241 / 80-89832
Fax : 049 / (0)241 / 88-88414
Email: jran@pcserver.mk1.rwth-aachen.de

Subject: Re: determining if a point is "inside" or "outside" a shape
Posted by Randall Frank on Fri, 22 Oct 1999 07:00:00 GMT
View Forum Message <> Reply to Message

I would suggest you read the Graphics FAQ on this issue and also check Graphics Gem (I think volume 1) for a more detailed explanation of this problem. The upshot is that there really are three core methods and many variants. In general, you can sum angles, sum signed areas or clip a line. There are good code examples of all these approaches on the net which can be coded into IDL very quickly. It also depends on how you intend to use the function. If, you are going to repeatedly test many points, you are better off using one of the sorted variants of the line clipping techniques. In general, the line clipping techniques are the fastest on the average, but have poor worst case performance without the sorting overhead. The angle sum is one of the slowest methods unless you can get creative and avoid the transcendentals (and you can). The area sum approach generally falls inbetween. In IDL code, I believe you can vectorize the latter with some setup overhead, making it the fastest for .pro code when testing multiple points with one point per call.

FWIW.

Job von Rango wrote:

```
>   Ther is another analytic way to solve the problem:
>
>   How to determine wether a point lies inside an arbitrary 2-dimensional
> polygon
>   or not?
>
>   IDEA:
>     Look at the sum of all angles tended between all pairs of neighboured
> vertics
>     the polygon and the given point.
>
>   IN DETAIL:
>     Assume we have the n vertices of polgons ordered at the positions:
>        v_1, v_2, ... ,v_n
>     The given point is denoted by p.
>
>     Now connect the given point p with all n neighboured
>     vertices, and get the n vectors beginning in p and ending
>     in the vertices:
>
>         vec_1 = vector(p_0, v_1)
>         ...
>         vec_n = vector(p_0, v_n)
>
>     Now add all n angles tended between the 2 vectors vec_i and vec_(i+1):
>
>         angle_i = angle(vec_i,vec_(i+1))
>
>     and calculate the sum:
>                       n
>         anglesum = sum  angle_i
>                      i=1
>
>     (Use vec_1 again for vec_(n+1) in the last angle_n)
>
>   RESULT:
>                       _
>                      |   2*pi          inside
>         anglesum = -|          if p is          the polygon
>                      |_    0           outside
>
>   IMPORTANT:
>     Take into account the correct sign of the angles
>     and use the vertices in ascending order!
>     Use e.g. the vector crossproduct:
>
```

>   vec_i x vec_(i+1)
>
>   in order to retrieve the absolut value |...| for the angle:
>
> |angle_i| = inv sin(|vec_i x vec_(i+1)|)
>
>   The valid sign for angle_i is given by looking at the scalar product
>   of the crossproduct from above and a fixed vector vec_perp, perpendicular
>   to the area of the polygon:
>
>   vec_perp * (vec_i x vec_(i+1))
>   sign_i =   -------------------------------------
> | vec_perp * (vec_i x vec_(i+1)) |
>
>   Now we get the correct value for all angles:
>
>   angle_i = sign_i * |angle_i|
>
> REMARK:
>   Examine the case of a given point inside the polygon, but very near
>   to the edge between two vertices v_i and v_(i+1). The contributing
>   angle will be:
>
>   angle_i = + pi - epsilon
>
>   (where epsilon denotes a small positive angle.)
>   If we shift p over the edge to the outside of the polygon
>   (but very near again to the edge)
>   the contributing angle will be:
>
>   angle_i = - pi + epsilon'
>
>   The switch of the sign is the reason for the discontinuity (2*pi <-> 0)
>   of anglesum for points moving from inside to the outside of the polygon!
>
> -- --
> Job v. Rango   Medizinische Klinik I
> Dipl.-Phys.   D-52074 Aachen
>   Pauwelsstrasse 30
> Tel. : 049 / (0)241 / 80-89832
> Fax : 049 / (0)241 / 88-88414
> Email: jran@pcserver.mk1.rwth-aachen.de

--
rjf.

Randall Frank | Email: rjfrank@llnl.gov
Lawrence Livermore National Laboratory | Office: B4525 R8019

P.O. Box 808, Mailstop:L-560 | Voice: (925) 423-9399
Livermore, CA 94550 | Fax: (925) 422-6287