Subject: Re: A distracting puzzle
Posted by Craig Markwardt on Mon, 17 Sep 2001 22:19:55 GMT
View Forum Message <> Reply to Message

JD Smith <jdsmith@astro.cornell.edu> writes:

>
> Given a polygon defined by the vertex coordinate vectors x & y, we've
> seen that we can compute the indices of pixels roughly within that
> polygon using polyfillv(). You can run the code attached to set-up a
> framework for visualizing this. It shows a 10x10 pixel grid with an
> overlain polygon by default, with pixels returned from polyfillv()
> shaded.
>
> You'll notice that polyfillv() considers only integer pixels, basically
> truncating any fractional part of the input polygon vertices (you can
> see this by plotting fix([x,x[0]]), etc.). For polygons on a fractional
> grid, this error can be significant.
>
> The problem posed consists of the following:
>
> Expand on the idea of the polyfillv algorithm to calculate and return
> those pixels for which *any* part of the pixel is contained within the
> polygon, along with the fraction so enclosed.
>
> For instance, the default polygon shown (invoked simply as
> "poly_bounds"), would have a fraction about .5 for pixel 34, 1 for
> pixels 33 & 43, and other values on the interval [0,1] for the others.
> Return only those pixels with non-zero fractions, and retain polygon
> vertices in fractional pixels (i.e. don't truncate like polyfillv()
> does).

Question: instead of making it a 10x10 image, could you make it a 100x100 image, or even a 1000x1000 image? Then you could resample back down using rebin, after converting to float of course, and get a reasonably accurate estimate of the area enclosed.

This is essentially performing an integral over a complex 2-d region. Another possibility is to do it by Monte Carlo. For example, cast a bunch of random 2-numbers onto the plane, and only accept those within the polygon (at least David has an IN_POLY routine, right?), and finally compute the fraction of accepted pairs.

If you want it exactly, then it sounds like you will be performing polygon intersections, which are non-trivial.

These ideas help?

Cheers,
Craig

--
---------- ----------
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
---------- ----------

Subject: Re: A distracting puzzle
Posted by David Fanning on Tue, 18 Sep 2001 04:29:29 GMT
View Forum Message <> Reply to Message

Craig Markwardt (craigmnet@cow.physics.wisc.edu) writes:

> These ideas help?

They help me. :-)

Cheers,

David

P.S. Let's just say this evening I finished reading
The Last Report on the Miracles at Little No Horse
by Louise Erdrich. I have the same feeling reading
this book that I have reading you two guys: I love it,
it's wonderful. I just don't see any way I can aspire
to it. But when I'm finished, I feel calm and I have
a sense that all is well with the world. That's
worthwhile (especially this week), even if I can't
always make out what the two of you are talking about. :-)

--
David W. Fanning, Ph.D.
Fanning Software Consulting
Phone: 970-221-0438, E-mail: david@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155

Subject: Re: A distracting puzzle
Posted by John-David T. Smith on Tue, 18 Sep 2001 16:05:35 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:

>
> JD Smith <jdsmith@astro.cornell.edu> writes:
>
>>
>> Given a polygon defined by the vertex coordinate vectors x & y, we've
>> seen that we can compute the indices of pixels roughly within that
>> polygon using polyfillv(). You can run the code attached to set-up a
>> framework for visualizing this. It shows a 10x10 pixel grid with an
>> overlain polygon by default, with pixels returned from polyfillv()
>> shaded.
>>
>> You'll notice that polyfillv() considers only integer pixels, basically
>> truncating any fractional part of the input polygon vertices (you can
>> see this by plotting fix([x,x[0]]), etc.). For polygons on a fractional
>> grid, this error can be significant.
>>
>> The problem posed consists of the following:
>>
>> Expand on the idea of the polyfillv algorithm to calculate and return
>> those pixels for which *any* part of the pixel is contained within the
>> polygon, along with the fraction so enclosed.
>>
>> For instance, the default polygon shown (invoked simply as
>> "poly_bounds"), would have a fraction about .5 for pixel 34, 1 for
>> pixels 33 & 43, and other values on the interval [0,1] for the others.
>> Return only those pixels with non-zero fractions, and retain polygon
>> vertices in fractional pixels (i.e. don't truncate like polyfillv()
>> does).
>
> Question: instead of making it a 10x10 image, could you make it a
> 100x100 image, or even a 1000x1000 image? Then you could resample
> back down using rebin, after converting to float of course, and get a
> reasonably accurate estimate of the area enclosed.
>
> This is essentially performing an integral over a complex 2-d region.
> Another possibility is to do it by Monte Carlo. For example, cast a
> bunch of random 2-numbers onto the plane, and only accept those within
> the polygon (at least David has an IN_POLY routine, right?), and
> finally compute the fraction of accepted pairs.
>
> If you want it exactly, then it sounds like you will be performing
> polygon intersections, which are non-trivial.

In case no one noticed, this is almost the same problem that font anti-aliasing and drawing smooth shapes with limited pixels present to graphics programmers. One approach is indeed over-sampling. If each pixel is over-sampled to a 16x16 pixel grid, and then something like polyfillv() is used on *that* grid with an appropriately scaled up

polygon, you can downsample the result (using, you guessed it, rebin()), and get an approximation (with a dynamic range of 256) to the area intercepted. The same guys also use stochastic sampling (aka Monte Carlo) to do the same thing, but with a smoother dithering. This might be especially good for strange shapes with difficult to calculate areas, but for straight-lined polygons, I had something more exact in mind.

The technique I was interested in is *area* sampling, so yes, the polygon intersections seem necessary for calculation. The reason is that I want much higher resolution than 100 or 256 levels of area, and ideally the algorithm would scale well to normal arrays, which typically have a much larger dimension than 10x10.

JD

Subject: Re: A distracting puzzle
Posted by air_jlin on Tue, 18 Sep 2001 19:18:45 GMT
View Forum Message <> Reply to Message

and a sense of awe and wonder. the feeling of "it's amazing someone understands that" and of seeing "wow, you can do that w/ idl?" is ultimately encouraging and inspiring :)

best,
-Johnny

Johnny Lin
CIRES, University of Colorado
Work Phone: (303) 735-1636
Web: http://cires.colorado.edu/~johnny/

David Fanning <david@dfanning.com> wrote in message news:<MPG.16107b955ffeabcc9896ad@news.frii.com>...
>
> P.S. Let's just say this evening I finished reading
> _The Last Report on the Miracles at Little No Horse_
> by Louise Erdrich. I have the same feeling reading
> this book that I have reading you two guys: I love it,
> it's wonderful. I just don't see any way I can aspire
> to it. But when I'm finished, I feel calm and I have
> a sense that all is well with the world. That's
> worthwhile (especially this week), even if I can't
> always make out what the two of you are talking about. :-)

Subject: Re: A distracting puzzle
Posted by Martin Downing on Tue, 18 Sep 2001 21:52:16 GMT
View Forum Message <> Reply to Message

Hi JD,

Since you are interested in high resolution, the relationship between pixels
and points is of interest.
I.e.: where in pixel (i,j) is point P(x=i, y=j)? Do you consider the pixel
to be centered on the point P(i,j) or P(i+0.5,j+0.5)?

Martin

--

Martin Downing,
Clinical Research Physicist,
Orthopaedic RSA Research Centre,
Woodend Hospital, Aberdeen, AB15 6LS.
Tel. 01224 556055 / 07903901612
Fax. 01224 556662

m.downing@abdn.ac.uk

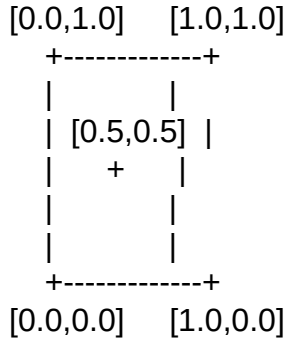
"JD Smith" <jdsmith@astro.cornell.edu> wrote in message
news:3BA770CF.E6EFDEB2@astro.cornell.edu...
> Craig Markwardt wrote:
>>
>> JD Smith <jdsmith@astro.cornell.edu> writes:
>>
>>>
>>> Given a polygon defined by the vertex coordinate vectors x & y, we've
>>> seen that we can compute the indices of pixels roughly within that
>>> polygon using polyfillv(). You can run the code attached to set-up a
>>> framework for visualizing this. It shows a 10x10 pixel grid with an
>>> overlain polygon by default, with pixels returned from polyfillv()
>>> shaded.
>>>
>>> You'll notice that polyfillv() considers only integer pixels,
basically
>>> truncating any fractional part of the input polygon vertices (you can
>>> see this by plotting fix([x,x[0]]), etc.). For polygons on a
fractional
>>> grid, this error can be significant.
>>>
>>> The problem posed consists of the following:
>>>
>>> Expand on the idea of the polyfillv algorithm to calculate and return
>>> those pixels for which *any* part of the pixel is contained within the

>>> polygon, along with the fraction so enclosed.
>>>
>>> For instance, the default polygon shown (invoked simply as
>>> "poly_bounds"), would have a fraction about .5 for pixel 34, 1 for
>>> pixels 33 & 43, and other values on the interval [0,1] for the others.
>>> Return only those pixels with non-zero fractions, and retain polygon
>>> vertices in fractional pixels (i.e. don't truncate like polyfillv()
>>> does).
>>
>> Question: instead of making it a 10x10 image, could you make it a
>> 100x100 image, or even a 1000x1000 image? Then you could resample
>> back down using rebin, after converting to float of course, and get a
>> reasonably accurate estimate of the area enclosed.
>>
>> This is essentially performing an integral over a complex 2-d region.
>> Another possibility is to do it by Monte Carlo. For example, cast a
>> bunch of random 2-numbers onto the plane, and only accept those within
>> the polygon (at least David has an IN_POLY routine, right?), and
>> finally compute the fraction of accepted pairs.
>>
>> If you want it exactly, then it sounds like you will be performing
>> polygon intersections, which are non-trivial.
>
> In case no one noticed, this is almost the same problem that font
> anti-aliasing and drawing smooth shapes with limited pixels present to
> graphics programmers. One approach is indeed over-sampling. If each
> pixel is over-sampled to a 16x16 pixel grid, and then something like
> polyfillv() is used on *that* grid with an appropriately scaled up
> polygon, you can downsample the result (using, you guessed it, rebin()),
> and get an approximation (with a dynamic range of 256) to the area
> intercepted. The same guys also use stochastic sampling (aka Monte
> Carlo) to do the same thing, but with a smoother dithering. This might
> be especially good for strange shapes with difficult to calculate areas,
> but for straight-lined polygons, I had something more exact in mind.
>
> The technique I was interested in is *area* sampling, so yes, the
> polygon intersections seem necessary for calculation. The reason is
> that I want much higher resolution than 100 or 256 levels of area, and
> ideally the algorithm would scale well to normal arrays, which typically
> have a much larger dimension than 10x10.
>
> JD

Subject: Re: A distracting puzzle
Posted by John-David T. Smith on Wed, 19 Sep 2001 13:28:13 GMT
View Forum Message <> Reply to Message

Martin Downing wrote:
>
> Hi JD,
>
> Since you are interested in high resolution, the relationship between pixels
> and points is of interest.
> I.e.: where in pixel (i,j) is point P(x=i, y=j)? Do you consider the pixel
> to be centered on the point P(i,j) or P(i+0.5,j+0.5)?
>
> Martin

This choice is somewhat arbitrary, but my convention has always been the latter: pixels centered at the 1/2 pixel. E.g. pixel [0,0] has center [0.5,0.5], and its lower left edge corresponds to [0.0,0.0]:



In case anyone is actually trying this for real, the correct answers for the 10x10 array and the default polygon given are (using my horribly slow algorithm):

Pix	Frac
11	0.3295
12	0.1284
21	0.3765
22	0.9866
23	0.4890
31	0.0567
32	0.9669
33	1.0000
34	0.5000
42	0.6706
43	1.0000
44	0.9006
45	0.0861
52	0.3176
53	0.8559

| 54 0.1299 |
| 62 0.0282 |
| 63 0.0876 |
+============+

JD

Subject: Re: A distracting puzzle
Posted by Stein Vidar Hagfors H[1] on Tue, 25 Sep 2001 16:11:10 GMT
View Forum Message <> Reply to Message

If what's being sought here is only to distinguish which pixels have *some* area inside the polygon and which do not, wouldn't it be sufficient to check the corners? I.e., in a continuum of pixel coordinates, given corners with coordinates [0,0], [1,0], [1,1], [0,1], it can be checked whether each of those are inside versus outside any defined polygon. If one or more of the corners is inside, then some area is also inside..

I have included some simple-minded routines I wrote some years ago to check whether a point is inside or outside a polygon...

Stein Vidar

```
;; $Id: vectorangle.pro,v 1.1 1999/06/02 16:24:14 steinhh Exp $
;;The angle between vector A & B
;; The angle that vector A needs to be rotated (counterclockwise) in order
;; to be parallell to B

FUNCTION vectorangle,x1,y1,x2,y2,zerovalue=zerovalue

  default,zerovalue,0.0

  dp = x1*x2 + y1*y2
  cp = x1*y2 - x2*y1

  ix = where(dp EQ 0 AND cp EQ 0)
  IF ix(0) EQ -1L THEN return,atan(cp,dp)*!radeg

  dp(ix) = 1.0
  res = atan(cp,dp)*!radeg
  res(ix) = zerovalue
  return,res

END
```

```
----------------------

;; $Id: insidepolygon.pro,v 1.2 1999/06/02 16:25:59 steinhh Exp $
;; Return true if the given point is inside the
;;

;; Poly == [2,N]

FUNCTION insidepolygon,ip,x,y,$
                    edge_is_inside=edge_is_inside

  IF size(ip,/type) NE 4 AND size(ip,/type) NE 5 THEN p = float(ip) $
  ELSE BEGIN
    copyback = 1
    p = temporary(ip)
  END

  np = (size(p))(2)

  x1 = p(0,*)-x
  y1 = p(1,*)-y
  x2 = shift(p(0,*),0,-1)-x
  y2 = shift(p(1,*),0,-1)-y

  zeroval = 1e5
  theta = vectorangle(x1,y1,x2,y2,zerovalue=zeroval)

  ix = where(theta EQ zeroval  $
           OR abs(theta-180.0d) LT 1e-4 $
           OR abs(theta+180.0d) LT 1e-4,count)
  IF count GT 0 THEN BEGIN
    result = keyword_set(edge_is_inside)
    GOTO,finished
  END

  ;; Test for those....

  result = abs(total(theta)) GT 180.0

finished:

  IF copyback THEN ip = temporary(p)

  return,result
END
```