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
