Subject: POLYFILLV weirdness
Posted by dktr.ted on Fri, 17 Nov 2006 03:29:32 GMT
View Forum Message <> Reply to Message

Hi all,

I've been having problems with POLYFILLV. Consider the case below:

```
IDL> verts = TRANSPOSE([ [ 2, 3.25, 2, 0.75, 2 ], [ 0.75, 2, 3.25, 2,
0.75 ] ])
IDL> dim = [ 6, 6 ]
IDL> inside = POLYFILLV(verts(0,0:3), verts(1,0:3), dim(0), dim(1))
```

Verts defines a diamond of width 2.5, centered at [2, 2], which
according to the POLYFILLV documentation should be the center of pixel
[2, 2]. The results of the POLYFILLV call should give me 5 pixels,
specifically [2, 3], [1, 2], [2, 2], [3, 2], and [2, 1]. However,
instead I get 6 pixels, those listed plus [3, 3]. From the POLYFILLV
documentation I can't for the life of me figure out how this extra
pixel would be regarded as being inside the diamond.

Has anyone else encountered any funny effects at polygon edges when
using POLYFILLV? Any advice or commiseration would be much
appreciated. Thanks!

Ted

Subject: Re: POLYFILLV weirdness
Posted by dktr.ted on Fri, 17 Nov 2006 23:52:08 GMT
View Forum Message <> Reply to Message

> PolyFillV is not using the provided polygons coordinates but a "fix()"
> of them.... which induce this extra line on the left and at the bottom
> (and a few missing pixels on the right side and on the top If I remember
> well). I personnaly used a round() over my polygon coordinates and it
> was returning much better results... though still not perfect!

This practice is particularly horrifying to me considering I frequently
use ROIs defined in physical coordinates and convert them to array

coordinates (commonly fractional) before running POLYFILLV. Is there
anywhere I can have a look at the actual algorithm used in IDL for this
routine? The documentation references the scan line coordinate system
defined in Rogers, Procedural Elements of Computer Graphics, 1985, but
I'm reluctant to hunt down this out of print text without confirmation
that I will get something useful out of it.

Ted

Subject: Re: POLYFILLV weirdness
Posted by Jean H. on Sun, 19 Nov 2006 22:43:00 GMT
View Forum Message <> Reply to Message

dktr.ted@gmail.com wrote:
>> PolyFillV is not using the provided polygons coordinates but a "fix()"
>> of them.... which induce this extra line on the left and at the bottom
>> (and a few missing pixels on the right side and on the top If I remember
>> well). I personnaly used a round() over my polygon coordinates and it
>> was returning much better results... though still not perfect!
>
>
> This practice is particularly horrifying to me considering I frequently
> use ROIs defined in physical coordinates and convert them to array
> coordinates (commonly fractional) before running POLYFILLV. Is there
> anywhere I can have a look at the actual algorithm used in IDL for this
> routine? The documentation references the scan line coordinate system
> defined in Rogers, Procedural Elements of Computer Graphics, 1985, but
> I'm reluctant to hunt down this out of print text without confirmation
> that I will get something useful out of it.
> Ted

Hi Ted,

the code is not available... but you can have a look here:
http://www.ittvis.com/services/techtip.asp?ttid=3539
The process is a bit more explained...

Jean

Subject: Re: POLYFILLV weirdness
Posted by JD Smith on Mon, 20 Nov 2006 20:31:47 GMT
View Forum Message <> Reply to Message

On Sun, 19 Nov 2006 15:43:00 -0700, Jean H. wrote:

> dktr.ted@gmail.com wrote:
>>> PolyFillV is not using the provided polygons coordinates but a "fix()"
>>> of them.... which induce this extra line on the left and at the bottom
>>> (and a few missing pixels on the right side and on the top If I remember
>>> well). I personnaly used a round() over my polygon coordinates and it
>>> was returning much better results... though still not perfect!
>>
>>
>> This practice is particularly horrifying to me considering I frequently
>> use ROIs defined in physical coordinates and convert them to array
>> coordinates (commonly fractional) before running POLYFILLV. Is there
>> anywhere I can have a look at the actual algorithm used in IDL for this
>> routine? The documentation references the scan line coordinate system
>> defined in Rogers, Procedural Elements of Computer Graphics, 1985, but
>> I'm reluctant to hunt down this out of print text without confirmation
>> that I will get something useful out of it.
>> Ted
>
> Hi Ted,
>
> the code is not available... but you can have a look here:
> http://www.ittvis.com/services/techtip.asp?ttid=3539
> The process is a bit more explained...

This is a crummy old algorithm. I've lobbied unsuccessfully for
RSI/ITTVIS to put real polygon clipping into IDL, either something
simple like Sutherland-Hodgeman, or a full-up "holes and degenerate
edges" Greiner-Hormann algorithm or other method, e.g. something
like gpc:

http://www.cs.man.ac.uk/~toby/alan/software/

Any of these can clip an arbitrary polygon against another polygon (or
just a grid in the Sutherland-Hodgeman case), compute the exact area
of overlap as well as the actual overlap polygon itself. That latter two
can even deal with holes and other weird polygon forms. I have a slow IDL
Sutherland-Hodgeman implementation, as well as a C DLM for the same, but
it sure would be nice not to have to go to that. If you share this
concern, let your ITTVIS representatives know.

JD

Subject: Re: POLYFILLV weirdness
Posted by badjelly.witch on Mon, 20 Nov 2006 22:49:30 GMT
View Forum Message <> Reply to Message

I second JD's response. There are many routines in IDL that are disturbingly vague about where the pixel edges and pixel centres lie. Perhaps this is because of IDL's emphasis on image processing, where the number of pixels is large and the edge effects don't matter (much).

A few years ago I wrote a reasonably versatile--but not very fast--pollyfillv-replacement routine using Sutherland-Hodgman clipping code I got from JD. (I think the problem I was addressing at the time was to calculate the land fraction for each cell in an ocean-model grid given coastline information.) Just the other day I had to revive this stuff and after 20 minutes of intense mental anguish I established that it works as it should and on a modern computer it's acceptably fast. The routine handles arbitray polygons projected onto rectilinear or curvilinear 2D grids. I'm happy to give you a copy, or you might want to try JD's implementation.