Subject: Re: Polygon Clipping Algo in IDL
Posted by JD Smith on Wed, 14 Dec 2005 19:01:54 GMT
View Forum Message <> Reply to Message

On Wed, 14 Dec 2005 12:02:17 +1300, Mark Hadfield wrote:

> JD Smith wrote:
>> On Tue, 13 Dec 2005 14:38:42 +1300, Mark Hadfield wrote:
>>
>> Hey Mark... remind me what your version did different from mine? I
>> recall spending weeks trying to vectorize this algorithm, only to give
>> up and write it as a DLM. It's fairly slow in IDL. Lately I've been
>> exploring the clipping library GPC, linking it as a DLM.
>
> It's been a while. Didn't your POLYCLIP clip to a square? Whereas my
> routines clip to a single line (must be parallel to the X or Y axis for
> MGH_POLYCLIP and of arbitrary orientation for MGH_POLYCLIP2). Clipping to
> a square or rectangle therefore requires 4 applications. I settled on the
> line-clipping functions because they are more general and applying them 4
> times is not significantly slower that applying a rectangle-clipping form
> once.

That sounds about right.

> I spent a fair bit of time tweaking the code and achieved a modest
> speed-up over the code you sent me (perhaps 2x). The thing that made the
> difference was converting a couple of functions into in-line code.

I did the same thing, and got a similar speedup. I also lobbied for
RSI to give us a high-speed parallel polygon clipper as a native
routine. They thought about it a bit. My final tool is called
POLYFILLAA, which, like POLYFILLV, computes clipped pixels inside a
polygon, but unlike POLYFILLV, gives you the area of the pixel which
was clipped, and can (optionally) return the clipped pixel polygon
itself (AA stands for "anti-aliasing").

> I actually wanted this code for the following situation: I have a 2D
> grid of cells (usually rectangular, sometimes curvilinear) and some
> polygons which, for the sake of argument, we can consider to be land
> areas. For each cell in the grid, i want the fraction that is inside one
> of the polygons. So for each cell I clip every polygon and calculate the
> area (if any) in the clipped polygon. This approach is *not*
> particularly fast and implementing the polygon clipping in IDL slows it
> down further. I'm sure DLM-ed C code would be faster and am interested
> in your results.

I'll send you the code offline. I actually use a clever method of
auto-compiling the C code using MAKE_DLL, and if that fails, falling
back on the slower IDL code. I'm not actually using a DLM, just
CALL_EXTERNAL (which is probably slower for lots of small polygon
clips), but I do get another factor of 3 or so speedup. I toyed with
encoding GPC as a DLM, and stuffing much larger numbers of polygons
down its throat at once, for (I expect) another much larger speedup,
but that still on the TODO list. The main annoyance is encoding all
of the variously sized resultant pixel polygons on return. A
reverse-indices type setup should work.

JD