Subject: IDLy approach to splatting points on a grid?
Posted by Jonathan Dursi on Thu, 23 Nov 2006 11:46:43 GMT
View Forum Message <> Reply to Message

I've been reading fairly carefully this group recently and many of the resources suggested, but I'm unable to figure out a loopless (or fast) way to do the following:

I have a collection of points in space, each contributing some value locally over some region (how big a region depends on the point), and I'm trying to produce a 1/2/3d grid of the sum of their contributions. In general, the points contribute over a small number of grid cells compared to the whole grid. (For those who are interested, the application here is analysis of Smoothed Particle Hydrodynamics simulations.) So for some field rho I want to calculate rho[i,j] = \sum_k {W_k(i,j) m_k} where the summation is over particles and there's a particle-dependant kernel W which for the time being is Gaussian, although later it'd likely be something tabulated.

For a given particle, I can looplessly/fairly quickly add the contribution to this particle to the grid. The issue I'm having is that I can't get rid of the *outer* loop, a loop over particles. If each particle contributed to the same number of grid cells, I think this would be straightforward -- although I don't konw how to do it offhand -- but that's not the case; each particle has it's own `size'. (If all the particles were very small compared to grid spacing of course this would just reduce to doing a 1/2/3d histogram...)

So right now I'm stuck doing something like (in, say, 2d)

```
;;loop over particles
;;find the cell the particles centre is in, cc, and
;; the number of grid cells in each direction that the particle
interacts with ndx/ndy
;;then:
;;   build `interaction list' where cc is the cell the particle is
located
        jlist = indgen(2*ndy+1)-ndy
        ilist = indgen(2*ndx+1)-ndx
        ntot = (2*ndx+1)*(2*ndy+1)

        allj = reform(rebin(jlist,(2*ndy+1),(2*ndx+1)),ntot)
        alli = rebin(ilist,ntot)

        coordlist = [[alli],[allj]] + transpose(rebin(cc[0:1],2,ntot))
        ;; limit this to cells that are within the appropriate ranges
        w = where((coordlist[*,0] ge 0) and (coordlist[*,1] ge 0) and
(coordlist[*,0] lt np[0]) and (coordlist[*,1] lt np[1]), c)
```

;; then calculate the contributions on each of coordlist[w,*]

There's doubtless optimizations that could be done on this inner `loop' but it's much faster than
how I had it before. The larger issue remains -- I'm looping over this little bit of code on order a million times, and the actual computation within this loop is usually fairly modest (because most particles interact only with a modest number of grid cells).

It might be that I'm thinking about this all wrong, and there's another way entirely to go about it. None of the obvious approaches seem immediately useful, though; for instance, I could go about things from the opposite end, going through all the grid points and summing up contributions from the appropriate particles. Again, that makes the inner `loop' easily IDL-ified/vectorized, but the outer loop remains over the mesh cells, and in general both the # of mesh cells and # of particles are going to be large. Note too that this is is different from the usual unstructured-data interpolation stuff -- triangulate/trigrid, etc. aren't directly helpful, because the situation isn't that a field is irregularly sampled, it's that the contributions to the field are irregularly located.

I could trade looping for computational inefficiency and just use a fixed size for all particles -- that corresponding to the largest one -- and compute lots of zero contributions. The problem there is that the size of the particles varies widely, and this would be a huge, huge hit.

Is this just hopeless? Do I do this in a low-level language and spit out the data in some format IDL can read for vis/analysis?

  Jonathan
--
Jonathan Dursi
ljdursi@gmail.com

Subject: Re: IDLy approach to splatting points on a grid?
Posted by JD Smith on Mon, 27 Nov 2006 18:05:38 GMT
View Forum Message <> Reply to Message

On Thu, 23 Nov 2006 03:46:43 -0800, Jonathan wrote:

> I've been reading fairly carefully this group recently and many of the
> resources suggested, but I'm unable to figure out a loopless (or fast) way
> to do the following:
>
> I have a collection of points in space, each contributing some value

> locally over some region (how big a region depends on the point), and I'm
> trying to produce a 1/2/3d grid of the sum of their contributions.
> In general, the points contribute over a small number of grid cells
> compared to the whole grid. (For those who are interested, the
> application here is analysis of Smoothed Particle Hydrodynamics
> simulations.) So for some field rho I want to calculate rho[i,j] =
> \sum_k {W_k(i,j) m_k} where the summation is over particles and there's a
> particle-dependant kernel W which for the time being is Gaussian, although
> later it'd likely be something tabulated.
>
<snip>
> Is this just hopeless? Do I do this in a low-level language and spit out
> the data in some format IDL can read for vis/analysis?

Shouldn't be hopeless. This is highly akin to the "Drizzle" method
often used for linear restoration of dithered astronomical images,
except each of your points has a varying region of influence which
doesn't depend on geometry (well, normal non-SPH geometry anyway).
Search the archive for information on the method. The reigning champ
(and one I still use all the time) is a dual histogram method. The
first histogram finds all contributing "input" pixels in a given
output pixel. The second one finds all output pixels with 1, 2, 3, 4,
.... up to n contributing input pixels. Then you can loop efficiently
over bin count and operate on large chunks of data at a time (to avoid
feeling the loop penalty).

See also:

http://www.dfanning.com/code_tips/drizzling.html

For you (actually for everyone doing this type of thing in IDL), the
problem will be efficiently calculating the "region of influence" of
all your input particles. Since it is highly variable depending on
particle position and size, it's somewhat difficult to vectorize
(though perhaps you have "bins" of influence radius and can attack it
that way). If you do write a DLM, the clipping operation is the best area
to concentrate effort.

JD
