

Re: Representing structures for simulations

Source: <http://coding.derkeiler.com/Archive/Fortran/comp.lang.fortran/2005-04/msg00822.html>

- *From:* Brooks Moses <bmoses-nospam@xxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Mon, 25 Apr 2005 15:28:52 -0700
-

justabeginner wrote:

> Hello. I am planning on programming my own small smooth particle
> hydrodynamics code. I have no prior experience with programming such
> code, or even other meshed methods. This SPH code will be interfaced
> with another program I have written to study hypervelocity impact.
>
> I guess the main question I have at the moment is how I should
> represent structures in my code. How is it normally approached, in both
> meshless and mesh-free methods?

I presume you mean "meshed" and meshless methods -- meshless and mesh-free are the same thing.

In meshed methods, there are two types of meshes: structured and unstructured. (Note that this distinction is approximate; there are meshes that are somewhat vaguely both.)

A structured mesh is, roughly speaking, one that can be stored in a multidimensional array. This does not imply fixed resolution, or even a cartesian grid; the grid can be distorted in various ways (e.g., a polar grid) to adjust the resolution in various regions.

An unstructured mesh is one that's made up of individual grid elements that can be randomly aligned; these are usually triangles or quadrilaterals in 2D, or tetrahedrons or six-faced blocks in 3D. These would typically be stored as a 1D array of grid cells, with the adjacency information also stored. (Depending on arrangement, the storage may be a 1D array of vertices with pointers to the cells that contain them, and a 1D array of grid cells which contain pointers to the relevant vertices). Because the adjacency information is precomputed and stored, it is not especially CPU-intensive to handle.

For large complicated geometries where a pure structured mesh may not be appropriate, one common approach is a "block" method, where the geometry is broken up into subdomains, and each subdomain is discretized in a structured mesh. Again, the adjacency information between subdomains is stored.

Re: Representing structures for simulations

Now, in mesh-free methods, things become trickier — and simpler. In the purest sense, because there is no mesh, there is no topological structure, and therefore the particles must be stored in a long 1D array.

As can be learned from the unstructured meshed methods, however, long 1D arrays are not CPU intensive if you have a good way of handling the adjacency data. The problem is complicated, however, by the fact that the particles are generally not fixed, and so the adjacency information changes. Thus, some cleverness is called for. I can't claim to be anywhere near an expert on the matter, but here are some things that occur to me from having read a few papers on SPH and related subjects:

One simple method is storing, with each particle, a list of all of the particles within X distance from it. At each timestep (or after each suitable number of timesteps), you iterate through the particles, and check all its neighbors and see if they're still within that distance. Additionally, you check all the neighbors of its neighbors, and see if they've become neighbors of it. And then you update the lists, and continue. This, of course, requires that particles not be changing positions too quickly, or you might not notice some new neighbors. Also, it starts becoming very CPU intensive unless the neighbor lists are small, or if particles interact with other particles that aren't on their neighbor lists.

Another method is to overlay the method with a grid of some sort (generally a structured mesh), and with each grid cell store a list of particles that are in that cell. Then, you need only do adjacency checks between particles in the grid cell and particles in that cell or adjacent cells.

There are undoubtedly many more methods, and means of refining these methods so that they work orders of magnitude better than naive approaches might; I second the recommendation of another poster to look into papers in the field. In this day and age, with access to decent online databases (if you're at a university, they should have access to something like the "ISI Web of Science"; failing that, Google Scholar is probably sufficient though much poorer), it shouldn't take more than a day or two to find much of the relevant literature in the field.

Beyond all the program design questions, there's the programming-level issue of storage. Unless you're going with a structured mesh (where the adjacency calculations are trivial), you need to store two kinds of information: particle adjacency information, and particle state information. Those need different kinds of storage.

Particle state information will virtually always be stored in a 1D array or set of 1D arrays, indexed by particle number. (There may be a strategy for re-using particle numbers if a particle goes out of the domain; I won't get into that. There may also be a strategy for how you

Re: Representing structures for simulations

choose your particle numbers to make memory accesses work better, but you absolutely should not be worrying about that at this stage.) I would suggest defining a type called ParticleState or something like that, containing named elements for each component of the state; alternately, you could use a 2D array with the second dimension being an index for the state components, but that's rather less clear.

Particle adjacency information is a different matter. This is where the lists, trees, graphs, lattices, and/or other more-complicated things that Jon Harrop mentioned are useful. Functionally, they probably get implemented in Fortran as a lot of arrays containing integers that "point" to other elements in the array or to elements in the particle state array, though they may contain actual Fortran pointers to other arrays, depending on how you implement your lists.

You may also find it useful to read (or at least skim) a book about data structures, so that you get an idea of what sorts of things are possible.

– Brooks

—

The "bmoses-nospam" address is valid; no unmunging needed.

.

- **References:**

- ◆ **[Representing structures for simulations](#)**

- ◆ *From:* justabeginner

- Prev by Date: **[Re: Does F90 do type checking when calling module subroutines?](#)**

- Next by Date: **[Re: Representing structures for simulations](#)**

- Previous by thread: **[Re: Representing structures for simulations](#)**

- Next by thread: **[Problem with array size](#)**

- Index(es):

- ◆ **[Date](#)**

- ◆ **[Thread](#)**