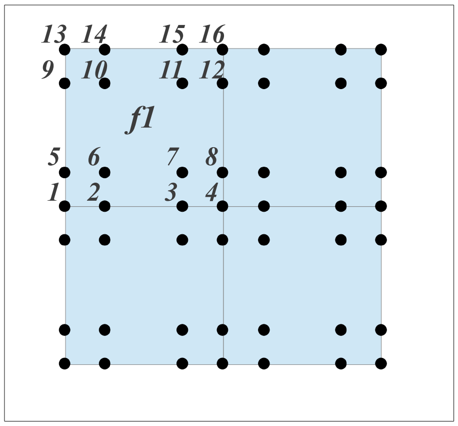
Note on Representation of Spectral Element Meshes



The Spectral Element Method (SEM) is a high-order method, using a polynomial Legendre interpolation basis with Gauss-Lobatto quadrature points, in contrast to the Lagrange basis used in (linear) finite elements [DEV02]. SEM obtains exponential convergence with decreasing mesh characteristic sizes, and codes implementing this method typically have high floating-point intensity, making the method highly efficient on modern CPUs. Most Nth-order SEM codes require tensor product cuboid (quad/hex) meshes, with each d-dimensional element containing $(N+1)^d$ degrees of freedom (DOFs). There are various methods for representing SEM meshes and solution fields on them; this document discusses these methods and the tradeoffs between them. The mesh parts of this discussion are given in terms of the iMesh mesh interface and its implementation by the MOAB mesh library [MOA12].

The figure above shows a two-dimensional 3rd-order SEM mesh consisting of four quadrilaterals. For this mesh, each quadrilateral has (N+1)^2=16 DOFs, with corner and edge degrees of freedom shared between neighboring quadrilaterals.

Representations

There are various representations of this mesh in a mesh database like MOAB, depending on how DOFs are related to mesh entities and tags on those entities. We mention several possible representations:

1) *Corner vertices, element-based DOFs:* Each quadrilateral is defined by four vertices, ordered in CCW order typical of FE meshes. DOFs are stored as tags on quadrilaterals, with size (N+1)^2 values, ordered lexicographically (i.e. as a 2D array tag(i,j) with i varying faster than j.) In the figure above, the connectivity for face 1 would be (1, 4, 16, 13), and DOFs would be ordered (1..16). Note that in this representation, tag values for DOFs shared by neighboring elements must be set multiple times, since there are as many copies of these DOFs as elements sharing them.
2) *High-order FE-like elements:* Each DOF is represented by a mesh vertex. Quadrilaterals each have (N+1)^2 vertices, ordered as they would be for high-order finite elements (corner vertices first, then mid-edge and mid-face elements; see [TAU10]). Mid -face, -edge, and -region vertices for a given edge/face/region would be ordered lexicographically, according to positive direction in a corresponding reference element. In the figure above, the connectivity array for face 1 would be (1, 4, 16, 13, 2, 3, 8, 12, 14, 15, 5, 9, 6, 7, 10, 11). DOF values are stored as tags on vertices. Since DOFs are uniquely associated with vertices and vertices are shared by neighboring elements, tag values only need to be set once. Full vertex-quadrilateral adjacencies are available, for all vertices.
3) *Linear FE-like elements, one vertex per DOF, array with DOF vertices:* Each quadrilateral is defined by four (corner) vertices, with additional vertices representing mid-edge and mid-face DOFs. An additional "DOF array" tag is assigned to each quadrilateral, storing the array of vertices representing the (N+1)^2 DOFs for the quadrilateral, ordered lexicographically. For the figure above, the connectivity array for face 1 would be (1, 4, 16, 13), and the DOF array would be (1..16), assuming that vertex handles are integers as shown in the figure. DOF values are stored as tags on vertices, and lexicographically-ordered arrays of DOFs can be retrieved using the DOF array tag as input to the tag_get_data function in MOAB. Adjacency functions would only be meaningful for corner vertices, but tag values would only need to be set once per DOF.
4) *High-order FE-like elements, array with DOF vertices:* This is a combination of options 2 and 3. The advantage would be full vertex-quad adjacency support and direct availability of lexicographically-ordered vertex arrays, at the expense of more memory.

5) *Convert to linear mesh:* Since a spectral element is a cuboid with higher-order vertices, it can always be converted to N^2 linear cuboids using the high-order vertices as corners of the finer quads/hexes. This is how readers in ParaView and VisIt typically import spectral meshes (CAM-SE also exports connectivity in this form).

As a convenience for applications, functions could also be provided for important tasks, like assembling the vertex handles for an entity in lexographic order (useful for option 2 above), and getting an array of tag values in lexicographic order (for option 3 above).

Tradeoffs

There are various competing tradeoffs in the various representation types. These include:

- **Adjacencies:** being able to retrieve the element(s) using a given (corner or higher-order) vertex.
- **Connectivity list:** being able to retrieve the connectivity of a given element, consisting of all (corner + higher-order) vertices in the element, usually in lexicographical order. This is closely linked with being able to access the connectivity list as a const*, i.e. using the list straight from memory without needing to copy it.
- **Memory vs. time:** There is a memory vs. execution time tradeoff between duplicating interface vertex solution/tag variables in neighboring elements (more memory but more time-efficient and allows direct access to tag storage by applications) versus using vertex-based tags (less memory but requires assembly of variables into lexicographically-ordered arrays, and prevents direct access from applications).

The lower-memory option (storing variables on vertices and assembling into lexicographically-ordered arrays for application use) usually ends up costing more in memory anyway, since applications must allocate their own storage for these arrays. On the other hand, certain applications will always choose to do that, instead of sharing storage with MOAB for these variables. In the case where applications do share memory with MOAB, other tools would need to interpret the lexicographically-ordered field arrays specially, instead of simply treating the vertex tags as a point-based field.

MOAB Representation

In choosing the right MOAB representation for spectral meshes, we are trying to balance a) minimal memory usage, b) access to properly-ordered and -aligned tag storage, and c) maximal compatibility with tools likely to use MOAB. The solution we propose is to use a representation most like option 2) above, with a few optional behaviors based on application requirements.

In brief, we propose to represent elements using the linear, FE-ordered connectivity list (containing only corner vertices from the spectral element), with field variables written to either vertices, lexicographically-ordered arrays on elements, or both, and with a lexicographically-ordered array of all (corner+higher-order) vertices stored on elements. In the either/or case, the choice will be evident from the tag size and the entities on which the tag is set. In the both case, the tag name will have a "-LEX" suffix for the element tags, and the size of the element tag will be (N+1)^2 times that of the vertex-based tag. Finally, the file set containing the spectral elements (or the root set, if no file set was input to the read) will contain a "SPECTRAL_ORDER" tag whose value is N.

References

[DEV02] M. O. Deville, P. F. Fischer, and E. H. Mund, *High-order methods for incompressible fluid flow*. Cambridge, UK; New York: Cambridge University Press, 2002.
[MOA12] T. J. Tautges, "MOAB Wiki." [Online]. Available: http://trac.mcs.anl.gov/projects/ITAPS/wiki/MOAB. [Accessed: 30-Oct-2012].
[TAU10] T. J. Tautges, "Canonical numbering systems for finite-element codes," *International Journal for Numerical Methods in Biomedical Engineering*, vol. 26, no. 12, pp. 1559–1572, 2010.