GMDS: A Generic Mesh Data Structure

F. Ledoux[1], J.-C. Weill[1], and Y. Bertrand[2]

[1] CEA/DAM Île de France, Arpajon, France
{franck.ledoux,jean-christophe.weill}@cea.fr
[2] Laboratoire IRCOM-SIC, Université de Poitiers, France
yves.bertrand@univ-poitiers.fr

1 Introduction

The development of a mesh data structure is a crucial task that needs to get the right balance between memory consumption and speed performances and to predict the useful and pertinent characteristics of the data structure. Today, working in a massive parallel context increases the difficulty of such a development. For software engineering reasons (code reusing, development cost) and technical reasons (mesh connectivity management, generality level, parallelism), the need for general meshing infrastructures has been recognized. Several development efforts have been done in the last few years [2, 3, 4, 5]. From our point of view, FMDB [5] is the most complete work. It is a C++ library providing a mathematically-founded mesh data structure which represents any cellular mesh in a memory distributed context. But, essentially due to an intensive use of STL containers, it consumes too much memory. In our applicative context, this drawback is very penalizing that is why we develop a new generic parallel C++ mesh data structure dedicated to mesh generation and modification. The proposed data structure, called Generic Mesh Data Structure (GMDS) is based on two mathematical models: a traditional cellular model [5] optimized in memory consumption, and the combinatorial map model [1] which allows us to handle global unpredictable topological modifications and provides a high level of genericity that you can not reach with traditional approachs.

This paper provides a snapshot of this work in progress. Sections 2 and3 describe our cellular model and the one based on combinatorial maps. Section 4 provides first memory consumption comparisons between our data structure, FMDB and MSTK [2].

2 Cellular Model

A n-dimensional cellular mesh model can be defined by the cells it handles and the available connectivities between these cells[3]. Such a description is topological, it gives an unambiguous shape-independent representation of the mesh. Some mesh models are given in Fig. 1. The description we use to modelize a mesh consists of a n-dimensional vector C and a $n \times n$ matrix[4] D. Vector C describes the stored cells and matrix D the stored connectivities. $D_{i,j} = 1$ (respectively 0) means that connectivities from i-cells to j-cells are stored (resp. not stored). A second $n \times n$ matrix I is computed from D and C to know which indirect adjacencies are available in the mesh model. Direct and indirect matrices of models M^3 and M^4 are given in Fig. 2. Indirect adjacencies are computed on the fly to satisfy a user request. A similar representation is used in FMDB where a single 4×4 matrix stores the available cells in the diagonal and cell adjacency otherwise. Adjacency between same dimension cells can not be stored while it is important in many meshing algorithms.

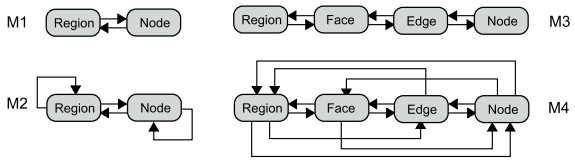

Fig. 1. Some mesh models. M^1 and M^2 are reduced models where only nodes and regions are present. Connectivities are modelized by arrows. In M^2, a region stores its nodes and some adjacent regions.

GMDS is a C++ library that intensively uses object-oriented concepts and generic programming. Generic classes are used to optimize storage consumption while interface classes provide flexibility and a user-friendly interface. every template classes has a parameter defining available cells and connectivities in the mesh. Direct and indirect matrices are computed from it. Generic programming techniques, like traits usage, optimize accesses and basic modifications throw exceptions without any test when an operation is not available for a specific mesh model. Generic cell classes have an extra parameter specifying the cell type (quad, triangle, hexahedron, tetrahedron,etc). This parameter coupled with the mesh model description allows us to optimize cell storage: Every cell class has a pointer C-style tabular whose size is defined at compile time considering mesh model and cell type. Downward connectivities are stored with the just necessary space while upward connectivities use two extra pointers to handle an extensible collection of connectivities. Let us

[3] Such a definition is not sharp enough to define a family of meshes. Generally, supported n-dimensional meshes are equivalent to n-dimensional quasi-manifolds.

[4] In this paper, we do not mathematically define what a cellular mesh is.

consider models M^1 and M^2 of Fig. 1. Connectivities of a tetrahedron in M^1 are stored inside a 4-size tabular while connectivities of the same tetrahedron in M^2 are stored in a 8-size tabular, the four first item store node adjacency while four last items store region adjacency. In M^2, hexahedron connectivities are stored using a 16-size tabular.

$$D^3 = \begin{pmatrix} 0&1&0&0 \\ 1&0&1&0 \\ 0&1&0&1 \\ 0&0&1&0 \end{pmatrix} \; I^3 = \begin{pmatrix} 1&\times&1&1 \\ \times&0&1&2 \\ 1&\times&1&\times \\ 2&2&\times&2 \end{pmatrix} \; D^4 = \begin{pmatrix} 0&1&1&1 \\ 1&0&1&1 \\ 1&1&0&1 \\ 1&1&1&0 \end{pmatrix} \; I^4 = \begin{pmatrix} 1&\times&\times&\times \\ \times&0&\times&\times \\ \times&\times&1&\times \\ \times&\times&\times&2 \end{pmatrix}$$

Fig. 2. Direct and indirect matrices of mesh models M^3 and M^4 given in Fig. 1. With matrix I^3 we know that to get the adjacent nodes of a face we have to go through edges ($I^3_{2,0} = 1$ and $I^3_{1,0} = \times$).

3 Combinatorial Map Model

Combinatorial maps define a general framework to encode any subdivision of n-dimensional topological spaces orientable or non-orientable with or without boundaries. Traditional cells are replaced by algebraic definitions where topology is built from a set of abstract elements, the darts, and applications α_i, with $0 < i \leq n$, defined on these darts. In 3D, we get the following definition which can be generalized to any dimension.

Definition 1 *A combinatorial 3-map M is a 4-tuple $(D, \alpha_1, \alpha_2, \alpha_3)$ where α_1 is a permutation*[5] *on D and α_2 and α_3 are involutions*[6] *on D.*

Usual mesh cells can be built as dart subsets obtained using α_i applications. For instance, on Fig. 3, the quad face can be described by dart d and application α_1. Note that a cell is a particular case of orbit[7]. An example of orbit is a face inside a region which allows you to attach properties to the two sides of a face. Combinatorial map model provides us a generic data structure which eases algorithm development: orbit notion gives flexibility to attach new types of properties; the management of any 3D cell is direct as cells are implicitly defined; all topological traversals and adjacency retrievals are possible. Moreover, it is a performing issue to handle dual meshes, block-structured meshes and non-conforming meshes.

Combinatorial maps are defined using three classes for maps, darts and orbits. Parameters defining a map-based mesh are the mesh dimension and

[5] A permutation f on X is a one-to-one mapping on X such that $\forall x \in X, \exists k \geq 1, f^k(x) = x$.

[6] An involution f on X is a one-to-one mapping on X such that $f^2 = id$.

[7] In 3D, there exist 16 orbits and 4 cells.

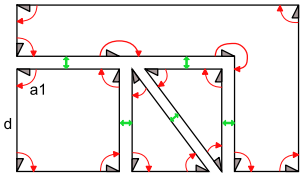

Fig. 3. A 2-dimensional combinatorial map. Darts are modelized by arrows as they are oriented. α_1 (resp. α_2) links darts belonging to adjacent edges (resp. faces).

the available orbits. Adjacency relations have not to be specified since they intrinsically exist in the combinatorial map model. The storage is very simple since the only entities to store are darts and orbits. Darts are defined by pointers given relation α_1, α_2 (and α_3 in 3D) and pointers to the orbits containing them. Orbits are defined by a pointer on a dart to access to the topology and an id defining it.

4 Memory Consumption Comparison

We have compared memory consumption between our data structure, FMDB and MSTK to evaluate the potential gain of our approach. This comparison is based on source code analysis by keeping comparable information. For instance, in FMDB, we do not consider elements relative to parallelism and geometric classification. To estimate "cell cost", we consider the type of attributes (`int`, `long`, `T*`), container strategies and potential virtual table pointers. The cost of a mesh is then computed using the next formula given in [2, 3]:

$$Storage(M) = \sum_{d=0}^{3} \sum_{i=1}^{N_d} (S_{cell} + \sum_{q=0}^{3} |\{M_i^d\{M^q\}\}| \times S_{adj})$$

where N_d is the number of d-dimensional cells, S_{ent} is the amount of memory each cell uses, S_{adj} is the amount of memory each adjacency uses, and $|\{M_i^d\{M^q\}\}|$ is the number of q-cells adjacent to d-cell M_i^d and stored in the mesh. The formula is different for combinatorial maps:

$$Storage(M) = \sum_{d=0}^{3} \sum_{i=1}^{N_d} S_{cell} + \sum_{k=0}^{N_k} S_{dart}$$

where N_k is the number of darts and S_{dart} is the amount of memory each dart uses[8]. We get results of Tab. 1 where the cost of mesh is the specified number of bytes $\times$ the number of mesh nodes. For any models M^1, M^2, M^3 and M^4, we consider tetrahedral (T) and hexahedral (H) meshes. The current GMDS cellular model consumes less memory than FMBD and MSTK while

[8] That is, 2 or 3 pointers to store α_i and a pointer by available orbit type.

combinatorial maps are expensive for reduced models. The low cost of cellular models is due to the fact we do no store downward adjacencies with generic containers. The high cost of combinatorial maps is due to the presence of all connectivities whatever the mesh model it is. Note this overcost is not prohibitive for full models.

Table 1. Memory Consumption Comparison on 64 bit machines (in bytes).

	M^1 (T)	M^1 (H)	M^2 (T)	M^2 (H)	M^3 (T)	M^3 (H)	M^4 (T)	M^4 (H)
FMDB	193	66	×	×	882	284	1247	408
MSTK	145	50	203	70	513	168	822	272
GMDS(cell)	71	28	117	44	325	112	506	178
GMDS(map)	642	254	642	254	1015	392	1015	392

5 Conclusion and Future Works

In this note, we have shortly introduced a mesh data structure based on a generic cellular model and combinatorial maps. This combination will allow us to represent any kind of meshes while optimizing memory consumption for cellular models. Combinatorial maps are memory expensive but they provide a high genericity level allowing us to easily handle complex kinds of meshes. Moreover it is a perfect framework to prototype new algorithms before converting them into an memory-optimized mesh model. The development of this data structure just begins and several studies and developments are expected in little time. Other memory consumption comparisons and speed comparisons will be then done. Some other developments are also planned: conversion from a model to another one, managment of memory-distributed applications, user-friendly mecanism to develop algorithms above GMDS and so on.

References

1. Lienhardt P (1991) Topological models for boundary representation: a comparison with n-dimensional generalized maps. Computer-Aided Design 23(1):59–82
2. Garimella RV (2002) Mesh data structure selection for mesh generation and FEA applications. International Journal for Numerical Methods in Engineering 55:451–478
3. Remacle JF and Karamete BK and Shephard MS (2003) Algorithm Oriented Mesh Database. International Journal for Numerical Methods in Engineering 58:349–374
4. Tautges T, Ernst C., Merkley K, Meyers R and Stimpson C (2005) Mesh Oriented datABase (MOAB). `http://cubit.sandia.gov/cubit`
5. Seol ES (2005) FMDB: Flexible Distributed Mesh Database for Parallel Automated Adaptative Analysis. Thesis, Faculty of Renssealer Polytechnic Institute