Engineering with Computers
DOI 10.1007/s00366-015-0395-0

ORIGINAL ARTICLE

IBHM: index-based data structures for 2D and 3D hybrid meshes

Marcos Lage · Luiz Fernando Martha · J. P. Moitinho de Almeida · Hélio Lopes

Received: 26 December 2013 / Accepted: 22 December 2014
© Springer-Verlag London 2015

Abstract We propose new topological data structures for the representation of 2D and 3D hybrid meshes, i.e., meshes composed of elements of different types. Hybrid meshes are playing an increasingly important role in all fields of modeling, because elements of different types are frequently considered either because such meshes are easier to construct or because they produce better numerical results. The proposed data structures are designed to achieve a balance between their memory requirements and the time complexity necessary to answer topological queries while accepting cells (elements) of different types. Additionally these data structures are easy to implement and to operate, because they are based on integer arrays and on basic arithmetic rules. A comparison with other existing data structures regarding their memory requirements and of the time complexities for the algorithms to answer general topological queries is also presented. The comparison shows that the overhead required to accept arbitrary cell types is small.

M. Lage
Universidade Federal Fluminense, Niterói, Brazil
e-mail: mlage@ic.uff.br

L. F. Martha (✉) · H. Lopes
Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil
e-mail: lfm@tecgraf.puc-rio.br

H. Lopes
e-mail: lopes@inf.puc-rio.br

J. P. Moitinho de Almeida
Instituto Superior Técnico, Universidade de Lisboa, Lisbon, Portugal
e-mail: moitinho@civil.ist.utl.pt

Keywords Topological data structures · Hybrid meshes · Compact data structures · Index-based data structures · Local adjacencies

1 Introduction

According to Bern and Plassman [6], a structured mesh is one where all interior vertices are topologically alike, and an unstructured mesh is one in which the vertices have an arbitrarily varying local neighborhood. Unstructured meshes have been recognized as a practical representation for geometric modeling and computer simulations mainly because they can be created for complex geometries much more easily than structured meshes [32, 33]. However, because they lack regularity, unstructured meshes are more memory demanding than structured meshes. They are typically defined by a few sets of tables that store some adjacency and incidence relations between entities of the mesh. For example, finite element methods (FEMs) commonly use two tables, one referencing the type of element and the other representing the element-node incidences. An inconvenience of this approach is that some FEM simulations or post-processing procedures may require other topological relations, for example, the identification of the elements that are adjacent to a given node, which cannot be obtained efficiently without using additional tables.

In addition, the meshes used for FEM simulations are usually composed of elements of a single type, such as triangles or quadrangles in two-dimensional problems and tetrahedra or hexahedra for the three-dimensional problem [29]. However, in some situations it is necessary to use hybrid meshes, i.e., meshes composed of different types of elements. They may be deliberately chosen because they lead to more stable solutions [17], or they may be

Published online: 04 February 2015

introduced by the mesh generator, for example, to make a transition between regions where structured meshes are used [26]. Polygonal finite elements are another example of a formulation that naturally leads to meshes with many different element types [34, 35].

1.1 Contributions

The aim of this work is to propose two new Index-Based topological data structures to represent two-and three-dimensional Hybrid Meshes (IBHM), i.e., meshes composed of cells of different types. These are called the 2D-IBHM and 3D-IBHM data structures, respectively. Their main characteristic is that they use arrays and map containers (data-structures that work on many different data types without being rewritten for each one [2]), and simple integer arithmetic rules. They are designed with the object oriented paradigm in mind and they are strategically constructed in order to balance the memory × performance complexity required to answer any query about adjacencies and incidence relations between their topological entities (vertices, edges, faces, and solids).

1.2 Paper outline

Section 2 describes the basics concepts and establishes the terminology. Section 3 discusses related work. Section 4 provides an overview of the IBHM data structures, and Sects. 5 and 6 describe the IBHM data structures for the two- and three-dimensional hybrid mesh representations, respectively. Section 7 presents an analysis of memory and performance complexity, comparisons with other structures and the results of some experiments. Finally, Sect. 8 concludes this work and proposes future directions.

2 Basic concepts

This section defines some basic concepts of combinatorial topology in order to unify terminology from the areas of geometric modeling and discrete numerical methods for engineering simulations.

An affine cell in $\mathbb{R}^p$ is the convex hull of a finite set of points. The dimension of the affine space spanned by this set of points is the dimension of the cell. A cell of dimension p in $\mathbb{R}^m$ is the homeomorphic image of an affine cell in $\mathbb{R}^p$. This means that a cell can be non-convex but is always a simply connected region (see Fig. 1). The cells on the boundary of a given cell are called subcells. The empty cell is supposed to be a subcell of all cells.

An n-dimensional cell complex K is a finite collection of i-dimensional cells ($i = 0, \ldots, n$) in $\mathbb{R}^m$ under the following conditions:

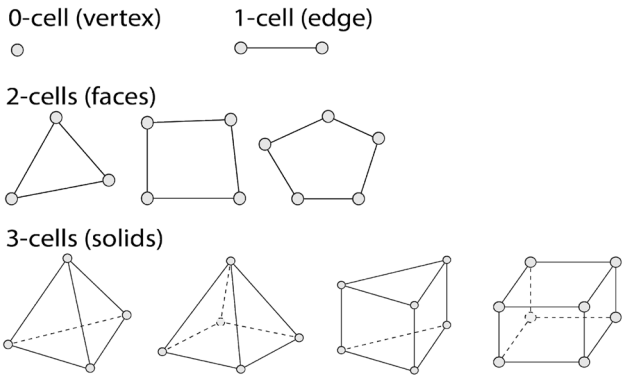

Fig. 1 Examples of topological models of different types of cells

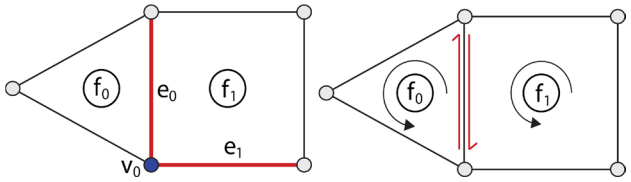

Fig. 2 The *left image* shows the incidence and adjacency relations of cells: f_0 and f_1 are adjacent; e_0 and e_1 are adjacent; e_0 is incident to f_0 and f_1; e_1 is incident to f_1; and v_0 is incident to e_0 and e_1. The *right image* shows the orientation induced by faces f_0 and f_1 on the center edge

1. If $\sigma \in K$ and τ is a subcell of σ, then $\tau \in K$.
2. If σ and $\tau \in K$, then $\sigma \cap \tau$ is a subcell of both σ and τ.

In this paper, a cell complex is referred to as a mesh, and the 0-, 1-, 2-, and 3-cells of a complex K are also called, respectively, the vertices, edges, faces, and solids of K. Considering a 2-dimensional mesh K, the above conditions for a valid mesh can be summarized as follows: the intersection of any two faces can only be an empty cell, a vertex or an entire edge, and the intersection of two edges can only be an empty cell or a vertex.

Two k-cells σ and $\rho \in K$ are adjacent when $\sigma \cap \rho \neq \emptyset$. If $\tau \in K$ is a subcell of $\sigma \in K$ then τ is said to be incident to σ and vice-versa. For example, in a two-dimensional mesh, as shown in Fig. 2, f_0 and f_1 are adjacent because they are 2-cells and share a common edge e_0. Likewise, edges e_0 and e_1 are adjacent by vertex v_0. Conversely, e_0 is incident to f_0 and f_1, and f_1 is incident to e_1.

If a collection of cells $L \subset K$ is a cell complex, then it is called a subcomplex of K. The star of a vertex $v \in K$, denoted by $\text{star}(v)$, is a subcomplex of K composed by the union of cells that are incident to v. The link of a vertex v, denoted by $\text{link}(v)$, is the boundary of $\text{star}(v)$. The open star of a vertex v, $\text{ostar}(v)$, is the set $\text{star}(v) - \text{link}(v)$.

A k-dimensional cell complex $M \subset \mathbb{R}^m$ is a k-manifold with boundary if the following conditions are satisfied:

1. A $(k-1)$-cell in M is incident to one or two k-cells of M.
2. The open star of a vertex in M is homeomorphic to an open subset of either $\mathbb{R}^k$ or $\mathbb{R}^k_+$.

The $(k-1)$-cells in a k-manifold M incident to only one k-cell and their subcells are called boundary cells. The collection of those cells forms the boundary of M and is denoted by ∂M. The boundary of a k-manifold is a $(k-1)$-manifold without a boundary, which means that the boundary of a 2-dimensional meshes is a collection of closed curves, and the boundary of a 3-dimensional meshes is a collection of 2-manifolds without a boundary. The cells that are not on the boundary are called interior cells.

A k-manifold is orientable when it is possible to choose a coherent orientation on its k-cells, i.e., two adjacent k-cells induce opposite orientations on their common $(k-1)$ -cells. In a combinatorial 2-manifold, any edge with a chosen induced 2-cell orientation is called a half-edge (see the left picture in Fig. 2, where the two half-edges of e_0 are illustrated). In the same way, any face with a chosen induced 3-cell orientation is called a half-face.

3 Related work

The study of topological data structures dedicated to mesh representation begins with the work of Baumgart [4], who proposed the Winged-Edge data structure for representing solids in $\mathbb{R}^3$. Several modifications have since been introduced in order to extend the range of objects to be modeled.

3.1 2-Manifolds

Examples of such extensions were proposed by Braid [7], who introduced the representation of non simply connected cells, and by Mäntylä [24], who defined the Half-Edge data structure. Guibas and Stolfi [14] also proposed a generalization to include non-orientable 2-manifolds implemented as the Quad-Edge data structure. Lopes [22] defined the Handle-Edge, a data structure that extends the Mäntylä's Half-Edge to explicitly represent the boundary of a manifold.

3.2 3-Manifolds

Dobkin and Laszlo [11] proposed the Facet-Edge, which extended the Quad-Edge to represent subdivision of a 3-manifold sphere. Lopes and Tavares [23] proposed the Handle-Face, which is an extension of the Handle-Edge to represent 3-manifolds with boundary.

3.3 n-Manifolds

Brisson [8] proposed a classification of topological data structures according to two categories, namely, those that, use an explicit or an implicit representation of cells. The Quad-Edge and the Facet-Edge do not explicitly represent each cell, while the Winged-Edge, the Half-Edge, the Handle-Edge, and the Handle-Face are examples of data structures with an explicit representation of cells. Another widely used dimension-independent data structure with an explicit representation is the Indexed Data Structure with Adjacencies proposed by Paoluzzi et al. [27].

3.4 Non-manifolds

Many applications require a non-manifold representation. Examples are objects with heterogeneous multi-regions, objects combining solid regions with degenerated parts, such as shells and wireframes; and FEM solid meshes with complete topological representation. Weiler [37] was the first to propose a non-manifold data structure, called Radial-Edge. Several modifications have since been suggested to address specific applications. Some substantial contributions to non-manifold representation include the works of Wu [38, 39], Yamaguchi and Kimura [40], Gursoz [15], Rossignac and O'Connor [30], Cavalcanti et al. [9], and Lee and Lee [21]. More recently, de Floriani and Hui [13] presented a very concise data structure for non-manifold manipulation, called NMIA, which is an extension of the Indexed data structure with adjacencies. Additionally, Pesco et al. proposed the Handle-Cell [28], which is an extension of the Handle-Edge, to address general 2-dimensional cell complexes. All of the above structures represent the cells explicitly.

3.5 Concise data structures

The data structures cited above are very suitable for representing unstructured meshes on finite element method implementations, but they are not efficient in terms of memory costs. Thus, the development of concise data structures has been recognized as an important challenge [10]. In this context, Rossignac et al. [31] proposed a concise data structure called Corner-Table, which uses only two tables of integers and a set of rules to represent triangular surfaces; Espinha et al. [10] introduced a concise data structure for tetrahedral meshes called Tops; Lage et al. [19] introduced the CHF, which is an extension of the Corner-Table to represent concisely tetrahedral

meshes; Gurungu and Rossignac [16] proposed the SOT data structure that represents in a very concise way tetrahedral meshes; and Remacle and Shephard [29] proposed the Algorithm Oriented Mesh Database (AOMD) to efficiently manage general mesh representations.

4 Overview of IBHM data structures

The 2D-IBHM and 3D-IBHM data structures were developed to address 2- and 3-manifold hybrid meshes, respectively. They are extensions of the compact half-edge (CHE) [18] and of the compact half-face (CHF) [19] data structures, adding the ability to represent meshes composed of hybrid geometry cells. They can also be considered a concise version of the Handle-Edge [22] and Handle-Face [23] mesh representations.

These new data structures use indexed arrays and map containers to represent a suitable set of adjacency and incidence relations between the cells and to obtain the remaining relations by the use of arithmetic rules. These characteristics are responsible for IBHM's most important feature: the balance between memory requirements and efficiency in terms of the computational complexity of topological queries. A limitation of these characteristics is observed when the mesh representation requires dynamic remeshing during the simulation, because they do not use pointers for dynamic memory allocation.

The 2D-IBHM and 3D-IBHM data structures are built from the typical cell-vertex incidence table usually adopted for Finite Element mesh representations. Additional arrays and containers are created from this basic information. To make it suitable for generic use, they are implemented using the object-oriented paradigm. The data provided by the user, hereafter called client information, is handled abstractly through generic methods that provide the cell-vertex incidence relations. Therefore, the IBHM's are encapsulated in abstract superclasses that store the additional arrays and map containers, which are used to obtain efficiently obtain the remaining incidence and adjacency relations. These additional data are referred to as derived information. The client of IBHMdata structures should instantiate subclasses of these superclasses, implementing the generic cell-vertex incidence methods in its own application.

Note that the cell-vertex incidence information and the local topology of each cell type are sufficient to represent hybrid meshes. For 2-manifold meshes, a cell is a simply connected polygon. Therefore, the client provides the number of vertices and their indices for each face in the mesh. For 3-manifold meshes, the client should first register the topology of all solid types (tetrahedra, hexahedra, pyramids, wedges, etc.) used in the mesh. This registration is performed through a special class that stores the local vertex incidence of the solid's faces. After that, the client may provide the vertex incidence of all solids in the mesh.

In a shared memory environment the proposed data structures can be directly accessed in parallel, because the stored information is not modified once the model is built. Therefore, for distributed memory environments, the procedures can also be directly used if a full copy of the adjacencies is kept at each node. Whenever some form of domain decomposition is implemented, it will be more convenient to store just the information related to the entities in each subdomain, together with a ghost layer surrounding it, similar to what is used, for example, in [5, 12, 25].

In many cases, the capabilities of the IBHM data structures will be excessive, because a given application will only require specific subsets of all possible types of cells and queries, implying that there will generally be an overhead associated with using an IBHM structure to access it. Nevertheless, when developing and testing new approaches that may potentially work with different topological entities, it is useful to know that the interface used to access the mesh is able to efficiently perform any query.

We envisage that our proposal will be particularly interesting for hybrid and hybridizable approaches, where discrete variables are assigned both to the cells and to their boundaries. In this case the assembly process requires the identification of the adjacencies of each cell, while recovery processes, focused on faces, edges or vertices, require their respective adjacencies. An example of a formulation potentially using most queries is the modeling of Kirchhoff plates using hybrid equilibrium elements [1].[1] This approach works on 2D meshes associating discrete variables with faces, edges and vertices. The process of assembling the global system is performed element-wise, requiring the identification of all the adjacencies of the faces. To post-process solutions, for example in a local recovery process, the procedures are centered on the vertices of the mesh, requiring information on the adjacent entities.

Sections 5 and 6 describe in detail the IBHM's data structures for 2D and 3D meshes, respectively.

5 The 2D-IBHM data structure

The oriented 2-manifold mesh representation provided by 2D-IBHM is based on the half-edge concept proposed by Mäntylä [24]. Note that a 2-manifold is orientable when it is possible to choose a coherent orientation on its faces, i.e., two adjacent faces induce opposite orientations on their common edges (see Fig. 2). A half-edge, here denoted as he, represents the edge orientation in the induced orientation of its incident

[1] In this context, Hybrid refers to a finite element model where additional fields are discretized on the interfaces of the elements.

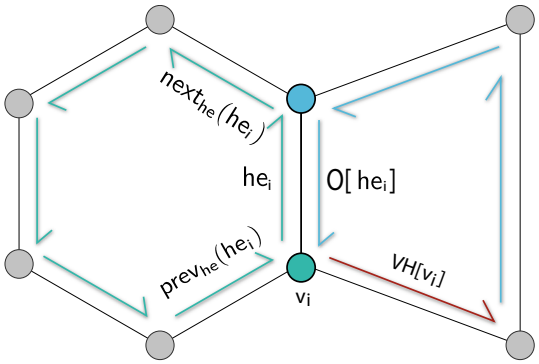

Fig. 3 A half-edge represents the edge orientation in a face. In its incident face the next and previous half-edges are obtained using integer arithmetic rules. The adjacency is encoded using the container of opposite half-edges. Two half-edges are opposites when they have the same vertices but with opposite orientation. To perform topological queries efficiently, the index of one incident half-edge is stored for each vertex of the mesh, as in the *red* boundary half-edge in the example (color figure online)

face in the mesh. A face is represented by a cycle of half-edges. In this cycle, each half-edge is associated with a single vertex, which corresponds to the origin of the half-edge (see Fig. 3). Any access to elements of a face is performed through its half-edges.

5.1 2D-IBHM client information

To build the 2D-IBHM data structure, the clients have to convey to the class object what are the indices of the vertices that compose each face of the mesh. The order of these indices on a face should be coherent with the mesh orientation. In the proposed data structure, this information is not replicated from the client application. This is achieved by providing the implementation of two abstract methods: the first returns the number of vertices of a given face c, called $\mathsf{n_v}(c)$, and the second returns the index of the kth vertex of a given face c, called $\mathsf{v}(c,k)$. This type of requirement is standard in mesh-based applications, such as the geometric modeling [31, 36] and finite element methods [3, 20].

5.2 2D-IBHM derived information

From this client information, which gives the face-vertex incidence relation, we derive other adjacency and incidence relations. Next, we will describe what these relations are and how they are built from the given client data.

5.2.1 The offset array

As we stated, the 2D-IBHM is based on the half-edge concept, and each half-edge is associated with its starting vertex within its 2-cell, naturally creating an ordering. Thus, the index of a half-edge is given by the sum of the number of vertices/half-edges on the previous faces plus the position (starting in 1) of its origin vertex in its face according to the client data. Consequently, the total number of half-edges is equal to the sum, for all faces, of their number of vertices.

2D-IBHM encodes the position of the first half-edge of each face in the list of half-edges using the Offset array, denoted as Of[], which has $\mathsf{size}(\mathsf{Of}[]) = \mathsf{n_c}$, where $\mathsf{n_c}$ denotes the number of cells on the mesh. It is computed directly from the information provided by the client:

$$\mathsf{Of}[1] = 0 \text{ and } \mathsf{Of}[\mathsf{c}] = \mathsf{Of}[\mathsf{c}-1] + \mathsf{n}_v(\mathsf{c}-1) \text{ for } \mathsf{c} \in 2 \ldots \mathsf{n_c}.$$

Using this array, the index of the ith half-edge that belongs to the face with index $\mathsf{c} \in \{1 \ldots \mathsf{n}_c\}$ is directly computed as $\mathsf{Of}[\mathsf{c}] + i$, with $i \in \{1 \ldots (\mathsf{n}_v(\mathsf{c}))\}$.

This array is used to determine the index of the face that contains a given half-edge, with index he, a question that is posed by any half-edge-based data structure. This query, denoted by $\mathsf{cell}_{\mathsf{he}}(\mathsf{he})$, is answered in our implementation using a binary search algorithm within the array Of[], whose code is detailed in Algorithm 1.

Algorithm 1 The cell of a half–edge

```
low ← 1; high ← size(Of[]);
while low < high do
    c = floor( 0.5 · (low + high + 1) );
    if he_i < Of[c] then
        high = c − 1;
    else
        low = c;
    end if
end while
if he_i < Of[c] then
    c = c − 1;
end if
return c;
```

This Offset array constitutes the main difference between 2D-IBHM and the original CHE data structure. It allows for a compact representation of faces that have an arbitrary number of vertices/half-edges, with the cost implied by the use of this search. Other schemes may be used, particularly the non-compact explicit storage of the face of each half-edge. In this case, the $\mathsf{cell}_{\mathsf{he}}(\mathsf{he})$ function will find the cell of a given half-edge in a constant time. From now on, this scheme will be called the explicit storage (ES) implementation, and the first scheme using Algorithm 1 will be called the binary search (BS) implementation.

Using the arrays and functions presented, the next and previous half-edges of a given half-edge he, all belonging

to the same face as illustrated in Fig. 3, can be obtained by applying the following arithmetic rules:

$$\begin{aligned} \mathsf{next_{he}(he, c)} &:= \mathsf{O[c]} + \mathit{circ}(\mathsf{he}+1, \mathsf{n_v(c)}), \\ \mathsf{prev_{he}(he, c)} &:= \mathsf{O[c]} + \mathit{circ}(\mathsf{he}-1, \mathsf{n_v(c)}). \end{aligned} \tag{1}$$

5.2.2 The opposite array

The adjacency relation between two faces is explicitly represented by the 2D-IBHM data structure using the Opposite array, denoted as O[], with dimension equal to the total number of half-edges.

When dealing with 2-manifolds, each half-edge must be incident to one or two cells (see Fig. 3). Therefore, the edge-adjacency between neighboring elements is represented by associating to each half-edge, he, its opposite half-edge, O[he], which has the same vertices but the opposite orientation.

In Fig. 3, for example, $\mathsf{O[he}_i\mathsf{]} = \mathsf{he}_j$, and symmetrically, $\mathsf{O[he}_j\mathsf{]} = \mathsf{he}_i$. If the half-edge is on the boundary, then it does not have an opposite, which is encoded by setting $\mathsf{O[he]} = 0$, thus allowing to a direct check of whether a half-edge he is on the boundary or not.

Algorithm 2 shows how to build array O[] using the client functions $\mathsf{n_v}(c)$ and $\mathsf{v}(c,k)$. In this algorithm a map container,[2] named adjacency, is used to uniquely identify the edge with the given sorted pair of vertices.

Algorithm 2 O[] array

```
O[] ← 0 {Initializes the container}
he ← 0
map{v_i × v_j → he_k} adjacency
for c ∈ {1...n_c} do
    n_vc ← n_v(c)
    for k ∈ {1...n_vc} do
        he ← he + 1
        {Get the vertices pair of he}
        v_a ← v(c,k) ; v_b ← v(c, circ(k+1, n_vc)) ;
        v2 ← sort(v_a, v_b)
        if adjacency.find(v2) then
            O[he] ← adjacency[v2]
            O[O[he]] ← he {Found opposite half–edge}
            adjacency.erase(v2)
        else {Temporarily stores half–edge}
            adjacency[v2] ← he
        end if
    end for
end for
```

[2] A map container is a type of fast key lookup data structure that offers a flexible means of indexing into its individual elements. It stores key-value pairs and there cannot be duplicate keys. Maps are useful in situations where a key can be viewed as a unique identifier for the object.

5.2.3 The extra vertex array

2D-IBHM obtains the incidence and adjacency relations of the vertices of the mesh through the Extra vertex array, denoted by VH[], which associates each vertex v to a half-edge that has v as an origin. For internal vertices an arbitrary half-edge starting at v may be used; if the vertex is on the boundary, the stored half-edge should be the boundary one (see Fig. 3). The dimension of this array is equal to the number of vertices on the mesh. Algorithm 3 indicates how to build it, using the client functions and the O[] table.

Algorithm 3 VH[] array

```
VH[] ← 0 {Initializes the container}
he ← 0
for c ∈ {1...n_c} do
    n_vc ← n_v(c)
    for k ∈ {1...n_vc} do
        he ← he + 1
        if O[he] = 0 OR VH[v(c,k)] = 0 then
            VH[v(c,k)] ← he
        end if
    end for
end for
```

5.3 Topological queries

Using the proposed data structure, we can develop topological procedures to traverse the mesh elements. The query $R_{i,j}$ represents the adjacency (incidence) relation of a topological entity of dimension i with others of dimension j. For example, $R_{0,2}$ represents the query that, for a given vertex v, answers the set the faces incident to v. $R_{0,2}$ is also known as the vertex star.

The 2D-IBHM would answer relations $R_{0,2}(\mathsf{v})$ in time $O(\deg(\mathsf{v}))$, where $\deg(\mathsf{v})$ represents the valency of a vertex v, if not for the $\mathsf{cell_{he}(he)}$ queries, which imply a time that is $O(\log \mathsf{n_c})$, resulting in a time that is $O(\deg(\mathsf{v}) \cdot \log(\mathsf{n_c}))$. The search overhead is the price to pay when we save the explicit storage of the cell of each half-edge. The test results obtained show that this overhead is low and grows slowly with $\mathsf{n_c}$.

The algorithm used to compute the $R_{0,2}$ topological operation using 2D-IBHM is described in Algorithm 4.

Algorithm 4 The vertex star – 2D

```
he ← VH[v] //Gets the first incident half–edge
he_a ← he //Auxiliary half–edge
repeat
    he ← O[prev_he(he)]
    R_0,2.push(cell_he(he))
until he_a = he OR O[he] = 0
```

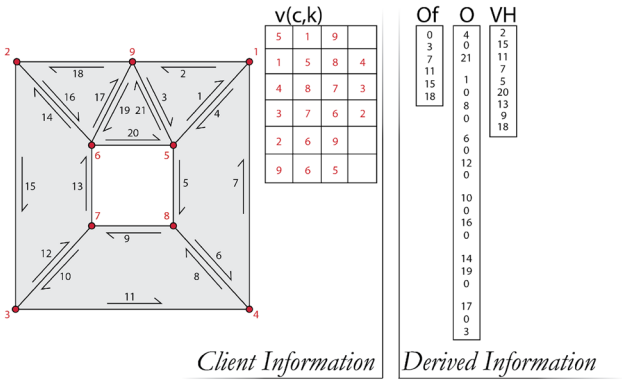

Fig. 4 Simple example of 2D-IBHM. On the *left* a mesh together with its half-edges and on the *right* the related 2D-IBHM arrays

5.4 Data structure example

Figure 4 illustrates the arrays used to manipulate the topological information: O[] has the opposite half-edge of each half-edge; VH[] stores a half-edge for each vertex of the mesh and Of[] has the offset of the position of the half-edges of a cell, with the global list of half-edges. Observe that the opposite of the boundary half-edges ($he = 2$, $he = 5$, $he = 7$, $he = 9$, $he = 11$, $he = 13$, $he = 15$, $he = 18$ and $he = 20$) are set to 0.

5.5 Summary

To build a 2-manifold mesh using the 2D-IBHM data structure, the user provides the client data, consisting of the indices of the vertices that compose each face of the mesh, which is provided by the implementation of two abstract methods: the first returns the number of vertices of a given face c, and the second returns the index of the kth vertex of a given face c. From this client data, three arrays are built in the following order: the offset array Of[], the opposite array O[] (both with dimensions equal to number of half-edges) and the extra vertex array VH[] (with dimension equal to the number of vertices).

6 The 3D-IBHM data structure

The volumetric mesh representation provided by 3D-IBHM uses the half-face concept, initially proposed by Weiler [37] and used by Lopes and Tavares [23]. A half-face, generically denoted as hf, represents the association of a 3-cell with one of its bounding faces. Any access to the elements of a 3-cell is performed through its half-faces. Figure 5 illustrates the half-face concept.

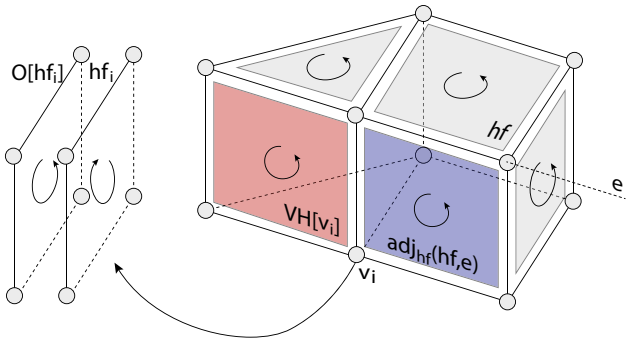

Fig. 5 Description of the half-faces of a 3-manifold. In a given cell, the adjacent-by-edge half-face is obtained using integer rules based on the topological description of each cell type. 3-Cell adjacency is encoded using the container of opposite half-faces. To perform topological queries efficiently, the index of one incident half-face is stored for each vertex of the mesh, for example the *red* boundary half-face (color figure online)

6.1 3D-IBHM client information

Similarly to what was presented in Sect. 5 for 2-dimensional meshes, 3D-IBHM requires from its clients the indices of the vertices that compose each solid of the mesh. Because this information is insufficient to uniquely define the topology of an arbitrary 3-cell, unlike what happens for faces in 2D-IBHM, the client must also explicitly indicate the type of each cell as well as the local topology of each type of 3-cell.

All this information is provided in two parts. First, the client must register each type of 3-cell, providing basic information on its topology. Section 6.2 will describe in detail how the client input this information to the 3D-IBHM data structure. Then, the client must also provide the implementation of two abstract methods: the first returns the type of a given 3-cell c, denoted by $t(c)$, and the second returns the index of the kth vertex of the 3-cell c, denoted by $v(c, k)$. These methods are then used to initialize the class object, building the derived information. Section 6.3 explains in detail how the derived information is obtained.

6.2 Template of a 3-cell

As mentioned, 3D-IBHM requires from its clients the description of each type of 3-cell used in the mesh, which, from the definition of its vertices, uniquely describes its topology.

This information is used to build the 3-cell template, which is composed of three matrices: the Vertex incidence matrix, the Face adjacency matrix, and the Half-edges matrix. These matrices are stored in the Template container, denoted by T[], which has dimension equal to the number of cell types. For example, if the mesh is only composed of hexahedra, then the number of cell types is 1, but if it is composed of tetrahedra, pyramids, prisms and hexahedra then the number of cell types is 4.

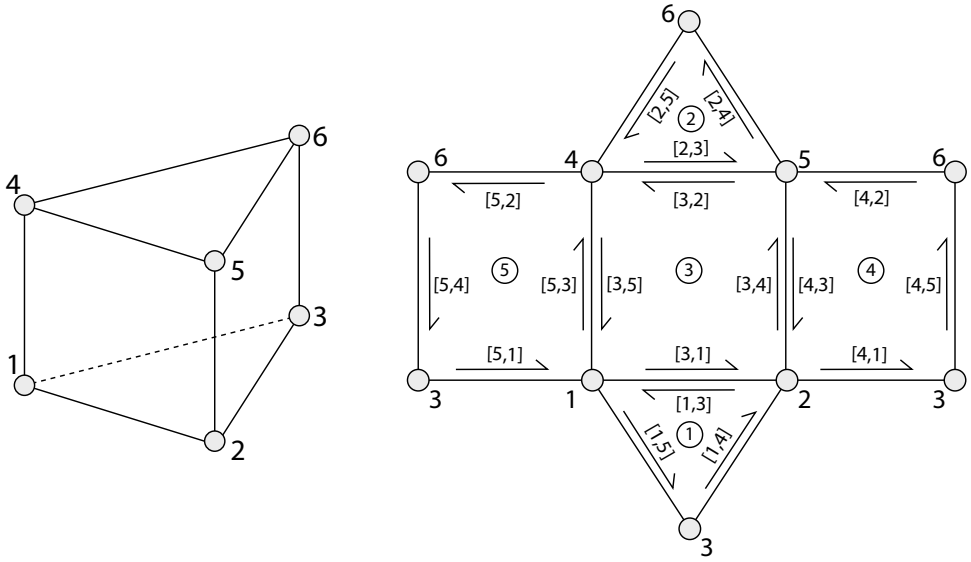

Fig. 6 Example of a cell type: the 3D view and a pacification of a wedge. Its topology is stored using the tables in Fig. 7

The Vertex incidence matrix of a 3-cell c with type $t(c)$, CV[][], has size(CV[][]) $= \mathsf{n_f}(t(c)) \times \overline{\mathsf{n_v}}(t(c))$, where $\overline{\mathsf{n_v}}(t(c))$ is the largest number of vertices of a face of the cell type. Each row of this matrix has the list of vertices of a face, padded with zeros for faces with less than $\overline{\mathsf{n_v}}(t(c))$ vertices. This is the only information that is explicitly provided by the user when registering a 3-cell type, and it constitutes the client information of the 3-cell template. The other two incidence matrices, which we describe next, are automatically built from the Vertex incidence matrix. They form the derived information of the 3-cell template.

It is very important to notice that the indices provided to fill the CV[][] array represent a local numbering of the elements. These numbers do not necessarily correspond to their global numbering of the mesh.

The Face adjacency matrix of a cell, CF[][], has the same dimension as the Vertex incidence matrix. It stores the adjacency relation between the faces of the 3-cell. Each line of this matrix is associated with one face of the 3-cell and its columns, which are also zero padded, indicating the faces that are adjacent to the given face.

Finally, the Half-edge matrix of a cell, denoted as CE[][], has size(CE[][]) $= \mathsf{n_v}(\mathsf{c}) \times \mathsf{n_v}(\mathsf{c})$ and stores the adjacency relations between the vertices of the cell, which correspond to the oriented edges or the half-edges of the cell surface description. In practice, each line and column is associated with vertices of the cell and a non-zero entry, CE[v_i][v_j], designates the face that contains the oriented edge whose origin is the vertex v_i and whose destination is the vertex v_j. Consequently, symmetric elements on this matrix correspond to adjacent faces of the cell.

The Vertex incidence, Face adjacency and Half-edge matrices of a 3-cell template for a prism, illustrated in Fig. 6, are presented in Fig. 7.

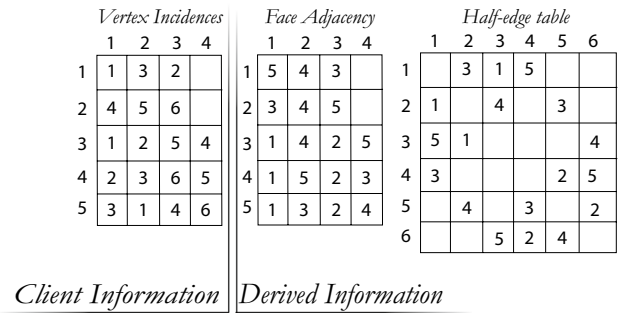

Fig. 7 Elements of the template for the wedge cell type. Only the Vertex incidence matrix, denoted by CV[][], has to be defined by the client. The Face adjacency and Half-edge matrices, denoted by CF[][] and CE[][] respectively, are built from the information in the Vertex incidence matrix

6.3 3D-IBHM derived information

From the client information and from the cell type template, we derive the other adjacency and incidence relations. We describe next what they are and how they are built.

6.3.1 The offset array

Each 3-cell c is represented, according to its type $t(c)$, by an array of $\mathsf{n}_f(t(c))$ half-faces that have a coherent orientation. Similarly to what is performed for indexing the half-edges of 2-manifold meshes, 3D-IBHM encodes the position of the first half-face of each 3-cell in the list of half-faces using the Offset array, Of[]. The details on the construction of this array and of the identification of the cell of a given half-face are exactly the same as those that were given in Sect. 5.2 for 2-manifold meshes.

To describe the adjacency relations between half-faces, an adjacent-by-edge operation is given by the following rule:

$$\mathsf{adj}_{\mathsf{hf}}(\mathsf{hf},\mathsf{e}) := \mathsf{Of}[\mathsf{cell}_{\mathsf{hf}}(\mathsf{hf})] + \mathsf{CF}[\mathsf{hf} - \mathsf{Of}[\mathsf{cell}_{\mathsf{hf}}(\mathsf{hf})]][\bar{\mathsf{e}}]. \quad (2)$$

This rule indicates, for a given half-face hf and one of its edges e, the half-face of the same 3-cell that is adjacent to the given half-face around the given edge. This relation is illustrated in Fig. 5.

For this equation, the Face adjacency matrix of the cell type, CF[][], is necessary. It is found on the template of the cell type, $\mathsf{n_c}(\mathsf{cell}_{\mathsf{hf}}(\mathsf{hf}))$. The local id of edge e within face hf, $\bar{\mathsf{e}}$, is obtained from a simple search using the Vertex Incidence, CV[][], which is found on the same template.

6.3.2 The opposite array

The adjacency relation between two 3-cells are again explicitly represented, also using an Opposite array, O[], which now relates adjacent half-faces, similarly to what the Opposite array does for half-edges on surface meshes. Its dimension is equal to the total number of half-faces.

Because we are dealing with 3-manifolds, each half-face must be incident to one or two 3-cells, as represented in Fig. 5. Therefore, the face-adjacency between neighboring elements is represented by associating to each half-face hf its opposite half-face O[hf], which has the same vertices but opposite orientation.

For example, in Fig. 5, the $\mathsf{O}[\mathsf{hf}_i] = \mathsf{hf}_j$, and, symmetrically, $\mathsf{O}[\mathsf{hf}_j] = \mathsf{hf}_i$. Again, in parallel to what is performed for two-dimensional meshes, a half-face hf on the boundary, does not have an opposite, which is encoded by setting $\mathsf{O}[\mathsf{hf}] = 0$. Thus, the value of O[hf] allows to a direct check of whether a half-face hf is on the boundary or not.

The construction of the Opposite array, using the client functions and the 3-cell template is very similar to what was presented in Algorithm 2 for 2-manifold meshes. Other than replacing he with hf, three main differences are noteworthy:

- the cycle in k, instead of looping on all of the vertices of the face, which uniquely identifies its half-edges in 2-manifold meshes, must now loop on all of the half-faces, as identified by the cell template;
- the identification of the vertices of a given half-face is obtained via the Vertex incidence matrix of the corresponding 3-cell template;
- the key v2, now used to identify a face, is obtained from the sorted list of vertices of each half-face.

Radial adjacencies of half-faces on an edge are obtained from the adjacency-by-edge rule, obtained from Eq. (2), with the information from this array:

$$\mathsf{radial}_{\mathsf{hf}}(\mathsf{hf},\mathsf{e}) := \mathsf{O}[\mathsf{adj}_{\mathsf{hf}}(\mathsf{hf},\mathsf{e})]. \quad (3)$$

6.3.3 The extra vertex array

The adjacencies of the vertices of the mesh are again explicitly represented in the Extra Vertex array, VH[], which for solid meshes associates each vertex v to a half-face. In case the vertex is on the boundary, the stored half-face should be a boundary one, as shown in Fig. 5.

The construction of this array is also very similar to Algorithm 3, noting that the identification of the half-faces to which a vertex belongs is more complex than the identification of the half-edges in surface meshes.

Algorithm 5 shows how to build the VH[] array using only the client functions and the incidence matrix of the 3-cell templates.

Algorithm 5 VH[] array

```
VH[] ← 0 {Initializes the array}
hf ← 0
for c ∈ {1...n_c} do
  n_fc ← n_fc(c)
  for i ∈ {1...n_fc} do
    hf ← hf + 1
    for k ∈ CV[i][] do
      if O[hf] = 0 OR VH[v(c,k)] = 0 then
        VH[v(c,k)] ← hf
      end if
    end for
  end for
end for
```

Algorithm 6 EH[] map

```
EH[] ← 0 {Initializes the container}
hf ← 0
for c ∈ {1...n_c} do
  n_fc ← n_fc(c)
  for i ∈ {1...n_fc} do
    hf ← hf + 1
    for k ∈ CV[i][] do
      e := (v(c,k), v(c, circ(k+1, n_v(CV[i][]))))
      if O[hf] = 0 OR not EH[].find(e) then
        EH[e] ← hf
      end if
    end for
  end for
end for
```

6.3.4 The edge map

Each edge is identified by an ordered pair of integers $\mathsf{e} := (\mathsf{v}_1, \mathsf{v}_2)$, where $\mathsf{v}_1 < \mathsf{v}_2$ are the indexes of the edge vertices, from which a unique identifier can be obtained.

The edges are then explicitly represented by a map container named EH[], linking each edge identifier (computed in our case as $\mathsf{v}_1 \cdot \mathsf{n}_0 + \mathsf{v}_2$, where n_0 corresponds to the number of vertices on the mesh) to one of its incident half-faces. If the edge lies on the boundary, the stored half-face will be the boundary half-face orientated from v_1 to v_2. This directly classifies whether the edge is in the interior or on the boundary of the solid.

The dimension of this map, which can be constructed using Algorithm 6, is equal to the total number of edges of the mesh.

6.4 Topological queries

Using the proposed data structure, we can develop topological procedures to traverse the mesh elements, in order to obtain, for example, vertex stars or edge stars, which are essential operations in volume modeling. Procedures that allow these $R_{0,3}$ and the $R_{1,3}$ topological operations to be efficiently performed using IBHM are described in Algorithms 7 and 8.

Details of the complexity analysis of the topological queries can be found in Sect. 7.

Algorithm 7 The vertex star – 3-Manifold

1: $\mathsf{hf}_a \leftarrow \mathsf{VH}[\mathsf{v}_a]$
2: $\mathsf{c}_a \leftarrow \mathsf{cell}_{\mathsf{hf}}(\mathsf{hf}_a)$
3: $R_{0,3}$.insert(c_a);
4: *CellStack*.init
5: **repeat**
6: find $i \in 1 \ldots \mathsf{n_v}(\mathsf{c}_a)$, so that $\mathsf{v}(\mathsf{c}_a, i) = \mathsf{v}_a$ {Locate vertex v_a in cell c_a}
7: $\mathsf{hf}s \leftarrow$ nonzeros($\mathsf{CE}[i][]$) {The half–faces of c_a adjacent to v_a}
8: **for** $\mathsf{hf} \in \mathsf{hf}s$ **do**
9: $\mathsf{hf}_b \leftarrow \mathsf{O}[\mathsf{Of}[\mathsf{c}] + \mathsf{hf}]$
10: **if** $\mathsf{hf}_b \neq 0$ **then**
11: $\mathsf{c}_b \leftarrow \mathsf{cell}_{\mathsf{hf}}(\mathsf{hf}_b)$
12: **if** $\mathsf{c}_b \notin R_{0,3}$ **then**
13: $R_{0,3}$.insert(c_b)
14: *CellStack*.push(c_b)
15: **end if**
16: **end if**
17: **end for**
18: $\mathsf{c}_a \leftarrow$ *CellStack*.pop
19: **until** isempty(c_a)

Algorithm 8 The edge star

1: $\mathsf{hf}_a \leftarrow \mathsf{EH}[\mathsf{e}_a]$
2: $\mathsf{c}_a \leftarrow \mathsf{cell}_{\mathsf{hf}}(\mathsf{hf}_a)$ {first cell}
3: $\mathsf{hf} \leftarrow \mathsf{hf}_a$; $\mathsf{c} \leftarrow \mathsf{c}_a$
4: **repeat**
5: $R_{1,3}$.insert(c); {insert to the result}
6: $\mathsf{hf} \leftarrow \mathsf{radial}_{\mathsf{hf}}(\mathsf{hf}, \mathsf{c})$ {radial face}
7: $\mathsf{c} \leftarrow \mathsf{cell}_{\mathsf{hf}}(\mathsf{hf})$ {next cell}
8: **until** $\mathsf{hf} = 0$ **or** $\mathsf{c} = \mathsf{c}_a$

6.5 Data structure example

Figures 8, 9 and 10 show one example of the proposed data structure for solid meshes. The IBHM client must give the nodal description of the each template cell by defining one CV[][] data for each cell type, as well as the nodal incidence of the mesh cells, information provided by $\mathsf{v}(c, k)$. Figure 10 shows also the derived topological information. Note that in this figure, the nodal description of the each template cell (through the definition of the two CV[][] arrays: one for the hexahedron 3-cell type and the other for the prism 3-cell type) indicates a local numbering of the vertices in the 3-cell template, while the $\mathsf{v}(c, k)$ indicates a global numbering of the vertices on the mesh. Using this example, the vertices whose global index are 15,16, and 18 of the 4th cell correspond to the vertices whose indices are 1, 3 and 2 of the second CV[][] table.

Container O[] stores the opposite half-faces. Observe that the opposites of the boundary half-faces ($\mathsf{hf} = 1$, $\mathsf{hf} = 2, \mathsf{hf} = 5, \mathsf{hf} = 6$, for example) are set to 0.

The VH[] container stores the one half-face incident to each vertex of the mesh. If we observe the entry VH[1], we see that because the vertex $\mathsf{v} = 1$ is a boundary vertex, its associated half-face is the one with id $\mathsf{hf} = 12$ since it is a boundary half-face incident to vertex $\mathsf{v} = 1$.

Finally, we show the container EH[], which indicates, for each edge, one of its incident half-faces. When the edge is at the boundary, we store a boundary half-face. In this example, all edges are on the boundary.

6.6 Summary

To build a 3-manifold mesh using the 3D-IBHM data structure, the user provides the client data composed of the client templates for each 3-cell type and the indices of the vertices that compose each 3-cell of the mesh. This is provided by the implementation of two abstract methods: the first returns the type of the given 3-cell c and the second returns the index of the kth vertex of a given 3-cell c according to the order in its corresponding template. From this client data, three arrays are built in the following order: the offset array Of[], the opposite array O[] (both with dimension equal to the number of half-faces); the extra vertex array VH[] (with dimension equal to the number of vertices) and the edge map EH[] (with dimension equal to the number of edges).

7 Memory and performance evaluation

In this section, we present a memory and performance evaluation of the proposed data structures. This is followed by timings obtained from a set of tests that measure the performance impact of using a binary search for the $\mathsf{cell}_{\mathsf{hf}}(\mathsf{hf})$ function.

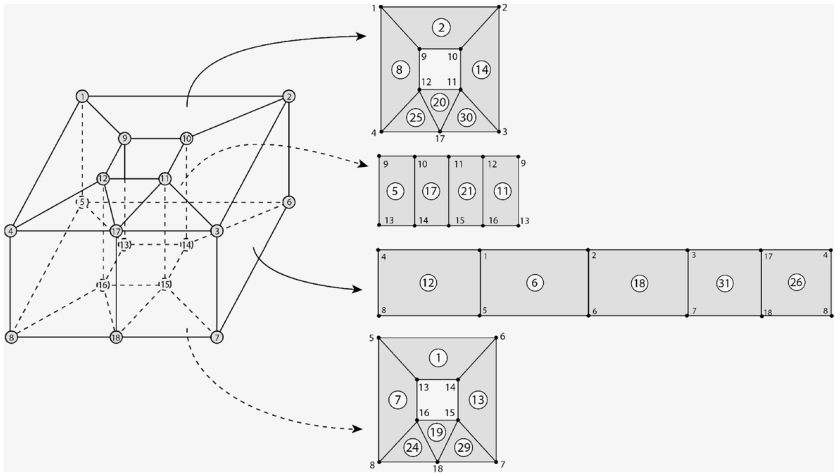

Fig. 8 Simple example of 3D-IBHM. Solid and numbering of the boundary half-faces. The numbering of the interior half-faces is presented in Fig. 9

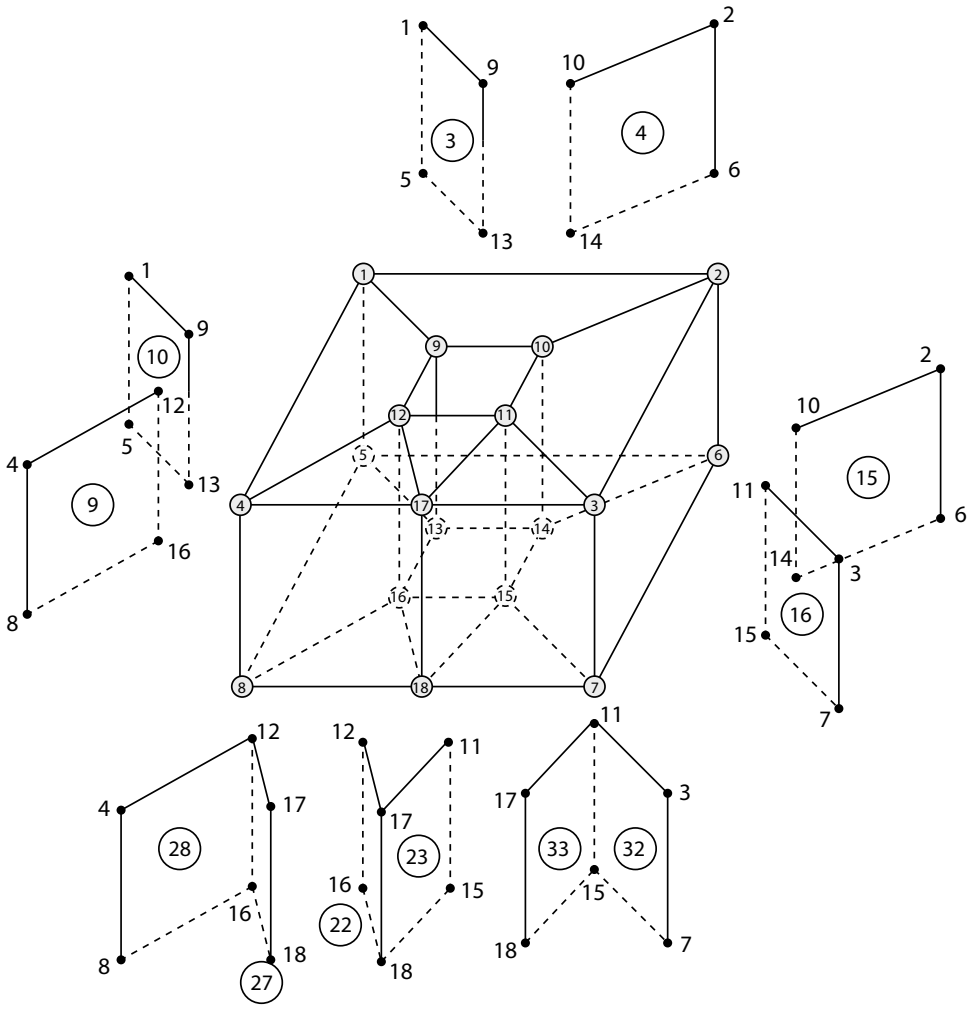

Fig. 9 Simple example of 3D-IBHM. Numbering of the interior half-faces. A solid example and the numbering of the boundary half-faces are presented in Fig. 8

These tests compare, for different situations, the time required to build the data structure and to obtain local adjacency information in relation to vertices and edges. For conciseness, only results for the solid representation are presented.

7.1 Memory and performance complexity comparisons

All data structures described in Sect. 3 provide different trade-offs between memory use and the time complexity of

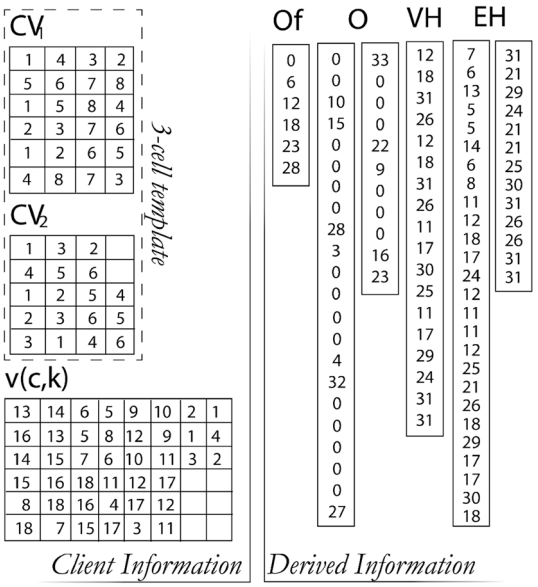

Fig. 10 Simple example of 3D-IBHM. Tables of the IBHM that describe the solid shown in Figs. 8 and 9

basic operations such as identifying an element, accessing its vertices, or computing its adjacencies. They reflect the balance between memory requirements and time complexity that a programmer always has to consider when handling huge data sets. We chose to compare the 3D-IBHM with the Handle-Face [23] and the CHF [19] data structures because 3D-IBHM is a concise representation of the first and an extension of the second, as CHF is a data structure for tetrahedral meshes only. From this point, we assume that the mesh to be represented has n_0 vertices, n_1 edges, n_2 triangular faces, and n_3 tetrahedra. For the memory comparisons, we will only consider the memory used to store the topological information.

To estimate the memory consumption of the model in terms of the number of vertices we can use the relation between the number of entities in a tetrahedral mesh provided in [29], which states that for a tetrahedral mesh with n_0 vertices, the estimated number of edges, faces and tetrahedra are, respectively: $n_1 \approx 7n_0$, $n_2 \approx 12n_0$ and $n_3 \approx 6n_0$, respectively. According to the authors [29], those relations are asymptotically correct for sufficiently large meshes but are surely incorrect for coarse meshes.

To represent the topology of a tetrahedral mesh, the Handle-Face uses five pointers per tetrahedron, six pointers per triangle, six pointers per edge and five pointers per vertex. Using the relations mentioned above, the *Handle-Face* uses approximately $(5 \cdot 6 + 6 \cdot 12 + 7 \cdot 6 + 5) \cdot n_0 = 149 \cdot n_0$ pointers or $4 \cdot 149 \cdot n_0 = 596 \cdot n_0$ bytes.

The CHF data structure, in its Level 2 version, describes the mesh using $4 \cdot n_3$ integers for O[], n_0 for VH[], $2n_1$ for EH[] (since it is a map container), and $4 \cdot n_3$ for V[]. Using the same assumptions described in the previous analysis, this structure spends approximately $4 \cdot (24 + 1 + 14 + 24) \cdot n_0 = 252 \cdot n_0$ bytes.

In addition to the CHF data structure, 3D-IBHM requires n_3 integers for Of[]. Consequently, it spends approximately $4 \cdot (24 + 1 + 14 + 24 + 6) \cdot n_0 = 276 \cdot n_0$ bytes. To improve the time complexity performance, one can implement the function $\mathsf{cell}_{\mathsf{hf}}(\mathsf{hf})$ using an Explicit Storage (ES) instead of using a Binary Search (BS). In so doing, more $4 \cdot (4 - 1) \cdot 6 \cdot n_0 = 72 \cdot n_0$ bytes will be used.

One can estimate the memory size of the mesh by the numbers presented in Table 1 and adding the memory used for the vertices coordinates and other properties or physical attributes stored on the data structure.

Table 1 Memory requirements for the mesh topology: comparison of the related data structures

	Memory consumption (bytes)
Handle-Face	$596 \cdot n_0$
CHF	$252 \cdot n_0$
3D-IBHM using BS	$276 \cdot n_0$
3D-IBHM using ES	$348 \cdot n_0$

Table 1 summarizes the comparison of the memory requirements of these three data structures and Table 2 shows the order of time complexity required to answer incidence and adjacency relations queries $R_{i,j}$ between the topological entities of the mesh. When analyzing these values we must consider that both Handle-Face and 3D-IBHM are able to represent hybrid meshes, while CHF is restricted to meshes of tetrahedra. The performance tests presented in Sect. 7.2 indicate that the additional computational complexity is not very high, but the memory savings are.

These results indicate that the proposed 3D-IBHM gives a suitable balance between memory requirements and time complexity for hybrid mesh representation, because it can represent arbitrary hybrid meshes and answer topological queries with a competitive performance while requiring a reasonable amount of memory.

7.2 Build and adjacency information performance tests

The data structure was implemented in MATLAB®, using additional containers to represent physical edges and faces. To minimize the dependency of the performance results on this specific implementation, the measured build times are divided by the total number of cells and the measured local adjacency times are divided by the total number of corresponding topological entities.

Figure 11 depicts a generic mesh adopted for the performance tests. A regular global grid with $N \times N \times N$ hexahedral grid-cells is used. Several mesh sizes, parameterized

Table 2 Time complexity for topological queries: comparison of the related data structures

Relation	Handle-Face	CHF	3D-IBHM using BS	3D-IBHM using ES
$R_{0,0}(\mathsf{v})$	$O(\deg(\mathsf{v}))$	$O(\deg(\mathsf{v}))$	$O(\deg(\mathsf{v}) \cdot \log(\mathsf{n}_3))$	$O(\deg(\mathsf{v}))$
$R_{0,1}(\mathsf{v})$	$O(\deg(\mathsf{v}))$	$O(\deg(\mathsf{v}))$	$O(\deg(\mathsf{v}) \cdot \log(\mathsf{n}_3))$	$O(\deg(\mathsf{v}))$
$R_{0,2}(\mathsf{v})$	$O(\deg(\mathsf{v}))$	$O(\deg(\mathsf{v}))$	$O(\deg(\mathsf{v}) \cdot \log(\mathsf{n}_3))$	$O(\deg(\mathsf{v}))$
$R_{0,3}(\mathsf{v})$	$O(\deg(\mathsf{v}))$	$O(\deg(\mathsf{v}))$	$O(\deg(\mathsf{v}) \cdot \log(\mathsf{n}_3))$	$O(\deg(\mathsf{v}))$
$R_{1,0}(\mathsf{e})$	$O(\deg(\mathsf{e}))$	$O(\deg(\mathsf{e}))$	$O(\deg(\mathsf{e}) \cdot \log(\mathsf{n}_3))$	$O(\deg(\mathsf{e}))$
$R_{1,1}(\mathsf{e})$	$O(\deg(\mathsf{e}))$	$O(\deg(\mathsf{e}))$	$O(\deg(\mathsf{e}) \cdot \log(\mathsf{n}_3))$	$O(\deg(\mathsf{e}))$
$R_{1,2}(\mathsf{e})$	$O(\deg(\mathsf{e}))$	$O(\deg(\mathsf{e}))$	$O(\deg(\mathsf{e}) \cdot \log(\mathsf{n}_3))$	$O(\deg(\mathsf{e}))$
$R_{1,3}(\mathsf{e})$	$O(\deg(\mathsf{e}))$	$O(\deg(\mathsf{e}))$	$O(\deg(\mathsf{e}) \cdot \log(\mathsf{n}_3))$	$O(\deg(\mathsf{e}))$
$R_{2,0}(\mathsf{hf})$	$O(1)$	$O(1)$	$O(\log(\mathsf{n}_3))$	$O(1)$
$R_{2,1}(\mathsf{hf})$	$O(1)$	$O(1)$	$O(\log(\mathsf{n}_3))$	$O(1)$
$R_{2,2}(\mathsf{hf})$	$O(1)$	$O(1)$	$O(\log(\mathsf{n}_3))$	$O(1)$
$R_{2,3}(\mathsf{hf})$	$O(1)$	$O(1)$	$O(\log(\mathsf{n}_3))$	$O(1)$
$R_{3,0}(\mathsf{t})$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
$R_{3,1}(\mathsf{t})$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
$R_{3,2}(\mathsf{t})$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
$R_{3,3}(\mathsf{t})$	$O(1)$	$O(1)$	$O(1)$	$O(1)$

by N, are adopted in the performance tests. To verify the efficiency in dealing with hybrid cell (element) topologies, each grid-cell may be one hexahedron, two wedges, three pyramids, six tetrahedra, or a combination of two tetrahedra and one octahedron, as shown in Fig. 11.

One key point in the proposed data structure for hybrid meshes is that it requires an auxiliary function $\mathsf{cell}_{\mathsf{hf}}(\mathsf{hf})$ to determine the index of a cell that contains a given half-face (or a given half-edge for a surface representation). Our implementation of this function uses a binary search, detailed in Algorithm 1 for the case of surface representation. To verify the efficiency of function $\mathsf{cell}_{\mathsf{hf}}(\mathsf{hf})$, we compare the build and adjacency query performance results with the results of an implementation with non-compact explicit storage of the cell of each half-face. Of course, adjacency queries perform better with explicit storage. On the other hand, in addition to more memory use, the data structure construction (build) time increases with explicit storage.

This may be observed in Fig. 12, which shows the build time results divided by the number of mesh cells (elements). The type of cell (hexahedron, wedge, pyramid, tetrahedron, and tetrahedron + octahedron) adopted in a mesh is indicated for each curve in the graphs. The tetrahedron +

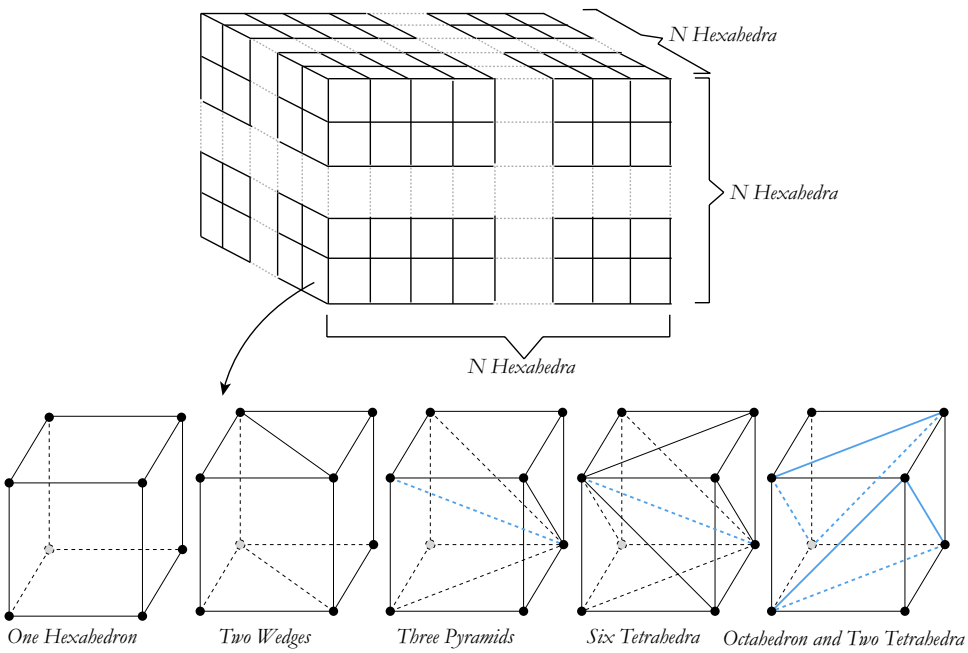

Fig. 11 Meshes used in the benchmark tests

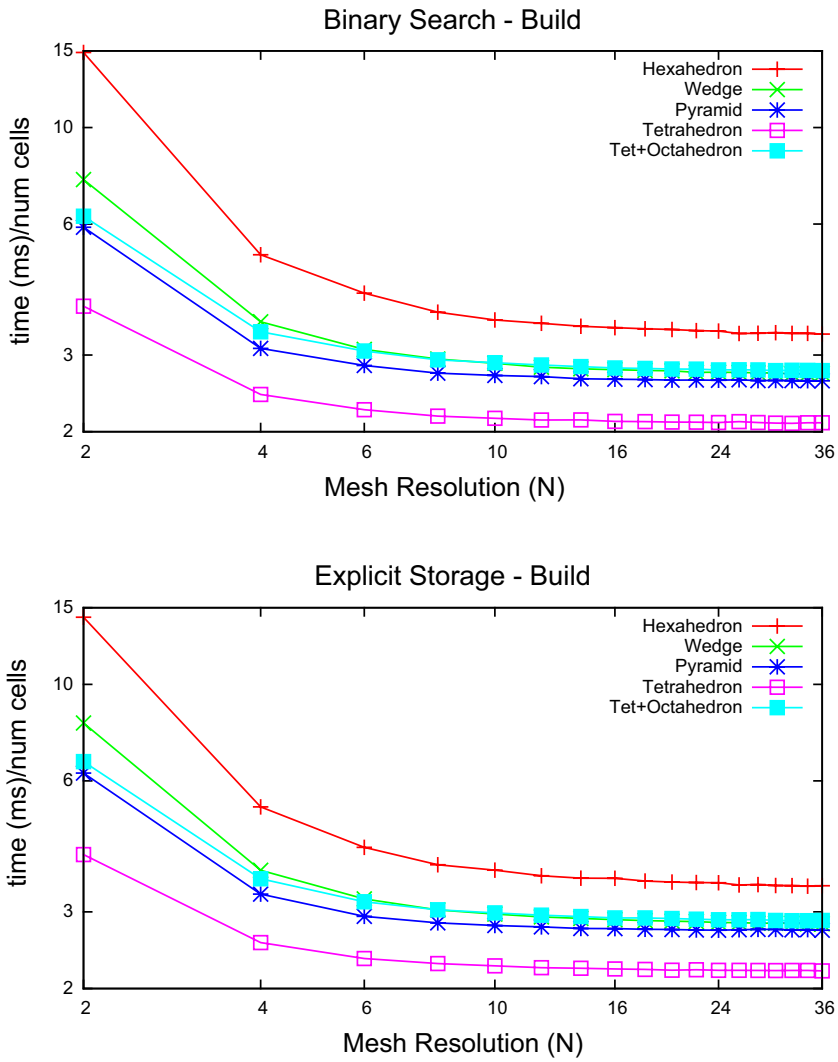

Fig. 12 Time to build the solid data structures

octahedron type corresponds to the hybrid mesh in which each grid-cell contains two tetrahedra and one octahedron. There are two graphs in Fig. 12: one with title "binary search", which corresponds to the results of function $\mathsf{cell}_{\mathsf{hf}}(\mathsf{hf})$ implemented using a binary search, and another with title "explicit storage", which corresponds to the results in which this function is implemented with explicit storage of the cell of each half-face. The global grid size varies from $2 \times 2 \times 2$ to $36 \times 36 \times 36$ grid-cells and the number of elements in the meshes varies from 8 to 279,936.

The build times shown in Fig. 12 do not consider the time to generate the original (conventional cell-vertex incidence) mesh, which is already in memory when the data structure is built. Therefore, these results report the time necessary to build the additional derived vectors and container maps that compose the proposed hybrid mesh compact representation.

The results of Fig. 12 show that there is an overhead to build a mesh, which is noticeable only for meshes with small sizes. As the mesh size increases, the ratio of build time to the number of cells tends toward a constant value for each type of mesh. This clearly indicates the efficiency of the data structure construction . It may be observed that, although the data structure construction time is higher for the case of explicit storage of the cell of each half-face, the difference in build time, when compared with the binary search, is negligible.

Figures 13 and 14 plot the time required for a local adjacency query, as a function of the size of the global grid. This time is computed by adding the time to obtain the local adjacency information for all topological entities (vertices or edges) in the mesh divided by the total number of entities queried. Similar graphs could be shown for adjacency queries in relation to faces and cells. However, these queries consist of direct access to data structure tables and practically do not depend on function $\mathsf{cell}_{\mathsf{hf}}(\mathsf{hf})$.

The relative cost of the vertex adjacency query for the two approaches considered is plotted in Fig. 15. For the

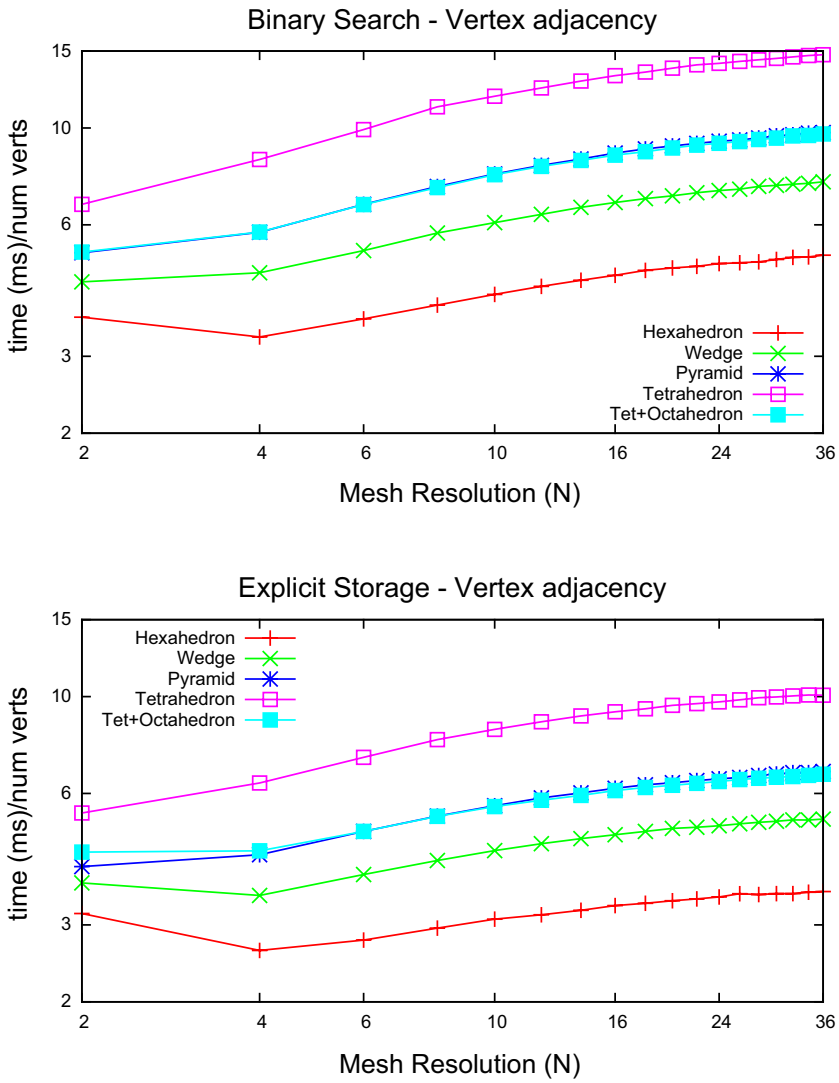

Fig. 13 Time to query the local adjacencies of a vertex

considered mesh resolutions, the overhead introduced by the use of the binary search is always below 50 %.

These figures demonstrate the efficiency of the proposed data structure in providing complete local adjacency information in a hybrid mesh. As predicted, the cost is higher for the binary search implementation of function $\mathsf{cell}_{\mathsf{hf}}(\mathsf{hf})$ compared with the implementation with explicit storage of the cell of each half-face. Furthermore, the relative cost grows when increasing the number of elements. However, a rough extrapolation of the plot in Fig. 15 indicates that an increase of 50 % would be obtained for $N \approx 150$ and that the relative cost would double for $N \approx 1{,}000{,}000$, a mesh dimension ($\approx 10^{18}$ vertices) that we do not plan to consider in the near future.

8 Conclusion and future works

We have shown that the IBHM data structures accomplish their designed goal: to represent hybrid meshes, with elements of user-defined types while achieving a balanced compromise between memory requirements and the computational cost required to answer arbitrary topological queries.

In the limit, the flexibility considered in the definition of the element types allows these data structures to represent a mesh where every element is different. In such cases a substantial memory overhead will be required to store each template, but the operations involved in topological queries will nevertheless remain local.

More efficient alternatives may be considered when all elements have a similar topology, and/or when only specific selected topological queries are required. The paradigm of this situation is the conventional, node-based, finite element formulations, where vertex-based description of the cells is sufficient and particularly efficient. However, there is a new trend for concise representations, like the SOT data structure for tetrahedral meshes. It is very efficient in terms of memory and has also a good time complexity

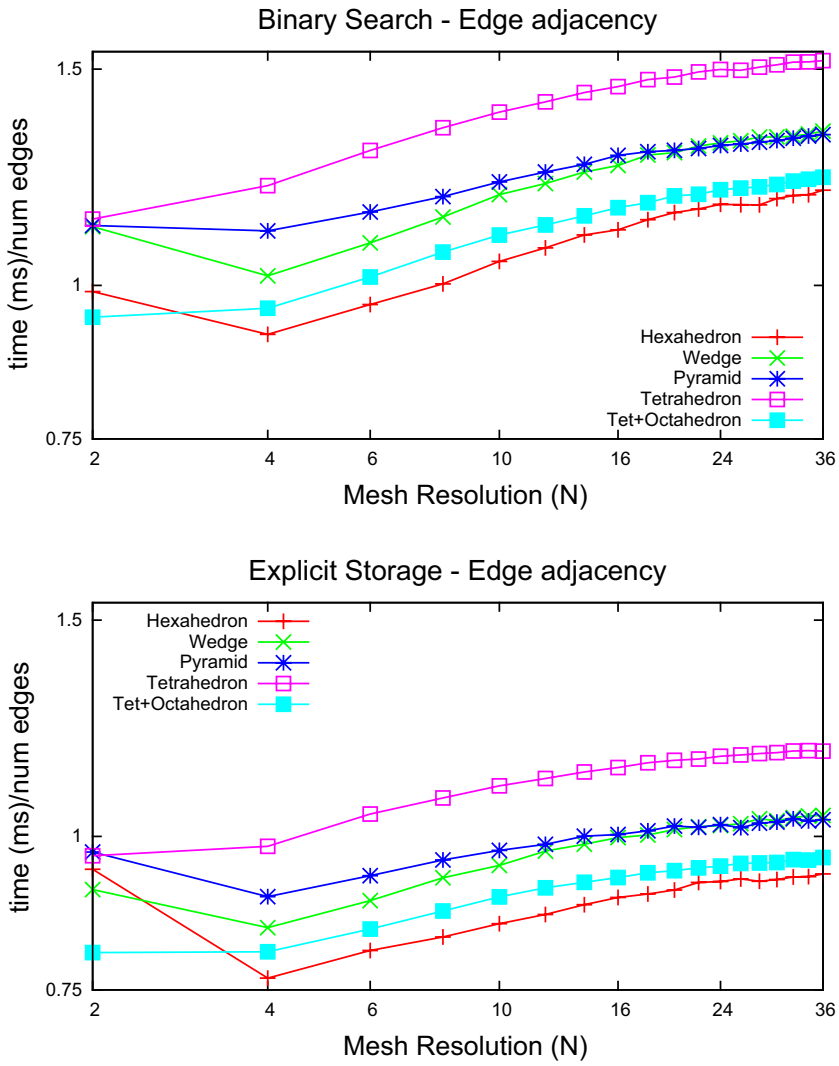

Fig. 14 Time to query the local adjacencies of an edge

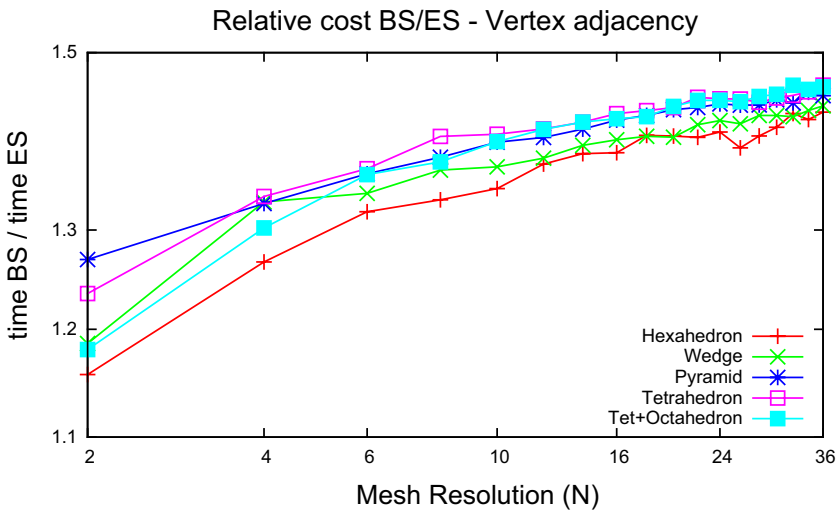

Fig. 15 Relative time to query the local adjacencies of an edge, using the two alternative approaches

performance, but it works only for tetrahedral meshes. In the future, data structure like that will have an important place in simulation.

We consider that alternative meshes and formulations, either with cells of arbitrary topology and/or requiring more than cell/vertex adjacency information, are the potential users of these data structures. We emphasize that because the proposed data structures are quite general, they may be particularly interesting in the development and testing of such innovative approaches, where every topological query can potentially be asked and there may be interest in testing the use of any type of cell.

These data structures have been implemented as two object-oriented MATLAB® classes. The preliminary results indicate that their performance corresponds to the expectations: query times are only marginally affected by the additional cost of the global search used to identify the cell of a given half-face. The corresponding codes will be published and made available for general use.

We envisage extending the IBHM data structures to include efficient local mesh modification procedures and explicit boundary representations.

Acknowledgments Part of the work described in this paper was developed during a sabbatical period of the second author at Instituto Superior Técnico, during the first semester of 2012, which was possible thanks to the support by PUC-Rio (Pontifícia Universidade Católica do Rio de Janeiro), CNPq (Brazilian National Council for Scientific and Technological Development), and the European Union through project NUMSIM—"Numerical simulation in technical sciences" of the Marie Curie Actions (FP7-PEOPLE-2009-IRSES), Project No. 246977. Hélio Lopes would like to thank CNPq and FINEP (Brazilian Funding Authority for Studies and Projects) for supporting his research. Marcos Lage would like to thank FAPERJ (Rio de Janeiro Research Funding Agency) for supporting his research.

References

1. Almeida JPM, Maunder EAW (2013) A general degree hybrid equilibrium finite element for Kirchhoff plates. Int J Numer Methods Eng 94(4):331–354
2. Austern MH (1999) Generic programming and the STL: using and extending the C++ standard template library. Addison-Wesley, Boston
3. Bangerth W, Hartmann R, Kanschat G (2007) Deal.ii a general-purpose object-oriented finite element library. ACM Trans Math Softw (TOMS) 33(4):24
4. Baumgart BG (1975) A polyhedron representation for computer vision. AFIPS Natl Comput Conf 44:589–596
5. Beghini LL, Pereira A, Espinha R, Menezes IF, Celes W, Paulino GH (2014) An object-oriented framework for finite element analysis based on a compact topological data structure. Adv Eng Softw 68:40–48
6. Bern MW, Plassmann PE (1997) Mesh generation. Pennsylvania State University, Department of Computer Science and Engineering, College of Engineering. http://www.cs.berkeley.edu/~jrs/meshpapers/BernPlassmann.pdf
7. Braid IC, Hillyard RC, Stroud IA (1980) Stepwise construction of polyhedra in geometric modeling. In: Brodlie KW (ed) Mathematical methods in computer graphics and design. Academic Press, San Diego, pp 123–141
8. Brisson E (1993) Representing geometric structures in d dimensions: topology and order. Discrete Comput Geom 9:387–426
9. Cavalcanti PR, Carvalho PCP, Martha LF (1997) Non-manifold modelling: an approach based on spatial subdivision. Comput Aided Des 29(3):209–220
10. Celles W, Paulino G, Espinha R (2005) A compact adjacency-based topological data structure for finite element mesh representation. Int J Numer Methods Eng 64:1529–1556
11. Dobkin DP, Laszlo MJ (1989) Primitives for the manipulation of threedimensional subdivisions. Algorithmica 4:3–32
12. Espinha R, Celes W, Rodriguez N, Paulino GH (2009) Partops: compact topological framework for parallel fragmentation simulations. Eng Comput 25(4):345–365
13. de Floriani L, Hui A (2003) A scalable data structure for three-dimensional non-manifold objects. In: Symposium on geometry processing. ACM, pp 72–82
14. Guibas LJ, Stolfi J (1985) Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. Trans Graph 4:74–123
15. Gursoz EL, Choi Y, Prinz FB (1990) Vertex-based representation of non-manifold boundaries. In: Turner JU, Wozny MJ, Preiss K (eds) Geometric modeling for product engineering. Elsevier, North-Holland, pp 107–130
16. Gurung T, Rossignac J (2009) Sot: compact representation for tetrahedral meshes. In: 2009 SIAM/ACM joint conference on geometric and physical modeling. ACM, pp 79–88
17. Ito Y (2013) Challenges in unstructured mesh generation for practical and efficient computational fluid dynamics simulations. Comput Fluids 85:47–52
18. Lage M, Lewiner T, Lopes H, Velho L (2005) CHE: a scalable topological data structure for triangular meshes. Technical report, Pontifical Catholic University of Rio de Janeiro
19. Lage M, Lewiner T, Lopes H, Velho L (2005) CHF: a scalable topological data structure for tetrahedral meshes. In: Computer graphics and image processing, 2005. SIBGRAPI 2005. 18th Brazilian symposium. IEEE, pp 349–356
20. Lage M, Lopes H, Carvalho MS (2011) Flows with suspended and floating particles. J Comput Phys 230(20):7736–7754. doi:10.1016/j.jcp.2011.06.031
21. Lee SH, Lee K (2001) Partial entity structure: a compact non-manifold boundary representation based on partial topological entities. In: Hoffman C, Bronsvort W (eds) Solid modeling and applications. ACM, pp 159–170
22. Lopes H, Pesco S, Tavares G, Maia M, Xavier A (2003) Handle-body representation for surfaces and its applications to terrain modeling. Int J Shape Model 9(1):61–77
23. Lopes H, Tavares G (1997) Structural operators for modeling 3-manifolds. In: Hoffman C, Bronsvort W (eds) Solid modeling and applications. ACM, pp 10–18
24. Mäntylä M (1988) An introduction to solid modeling. Computer Science Press, Rockville
25. Mubarak M, Seol S, Lu Q, Shephard MS (2013) A parallel ghosting algorithm for the flexible distributed mesh database. Sci Program 21(1):17–42
26. Ovcharenko A, Chitale K, Sahni O, Jansen K, Shephard M, Tendulkar S, Beall M (2012) Parallel adaptive boundary layer meshing for cfd analysis. In: Proceedings of the 21st international meshing roundtable. Springer, pp 437–455
27. Paoluzzi A, Bernardini F, Cattani C, Ferrucci V (1993) Dimension-independent modeling with simplicial complexes. Trans Graph 12(1):56–102. doi:10.1145/169728.169719

28. Pesco S, Lopes H, Tavares G (2004) A stratification approach for modeling 2-cell complexes. Comput Graph 28(2):235–247
29. Remacle JF, Shephard MS (2003) An algorithm oriented mesh database. Int J Numer Methods Eng 58(2):349–374. doi:10.1002/nme.774
30. Rossignac J, O'Connor MA (1990) SGC : a dimension independent model for pointsets with internal structures and incomplete boundaries. In: Turner JU, Wozny MJ, Preiss K (eds) Geometric modeling for product engineering. Elsevier, North-Holland, pp 145–180
31. Rossignac J, Safonova A, Szymczak, A (2001) 3D compression made simple: edgebreaker on a corner-table. In: Shape modeling international. IEEE, pp 278–283
32. Seol ES, Shephard MS (2006) Efficient distributed mesh data structure for parallel automated adaptive analysis. Eng Comput 22(3–4):197–213
33. Shephard M, Jansen K, Sahni O, Diachin L (2007) Parallel adaptive simulations on unstructured meshes. In: Journal of physics: conference series, vol 78. IOP Publishing, p 012053
34. Talischi C, Paulino G, Pereira A, Menezes I (2012) Polymesher: a general-purpose mesh generator for polygonal elements written in matlab. Struct Multidiscip Optim 45(3):309–328. doi:10.1007/s00158-011-0706-z
35. Talischi C, Paulino G, Pereira A, Menezes I (2012) Polytop: a matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. Struct Multidiscip Optim 45(3):329–357. doi:10.1007/s00158-011-0696-x
36. Vieira AW, Lewiner T, Velho L, Lopes H, Tavares G (2004) Stellar mesh simplification using probabilistic optimization. Comput Graph Forum 23(4):825–838. doi:10.1111/j.1467-8659.2004.00811.x
37. Weiler KJ (1986) Topological structures for geometric modeling. Ph.D. thesis, Rensselaer Polytechnic Institute, New York, USA
38. Wu ST (1989) A new combinatorial model for boundary representation. Comput Graph 13(4):477–486
39. Wu ST (1992) Non-manifold data models: implementation issue. In: CAD/CAM—MICAD. Computer graphcis and computer aided technologies, pp 37–56
40. Yamaguchi Y, Kimura F (1995) Non-manifold topology based on coupling entities. Comput Graph 15(1):42–50