CHF: A Scalable Topological Data Structure for Tetrahedral Meshes

Marcos Lage[1], Thomas Lewiner[1,2], Hélio Lopes[1] and Luiz Velho[3]

[1]PUC–Rio — Departamento de Matemática — Matmídia Project — Rio de Janeiro — Brazil
[2]INRIA — Géométrica Project — Sophia Antipolis — France
[3]IMPA — Visgraf Project — Rio de Janeiro — Brazil
{mlage,tomlew,lopes}@mat.puc-rio.br, lvelho@visgraf.impa.br

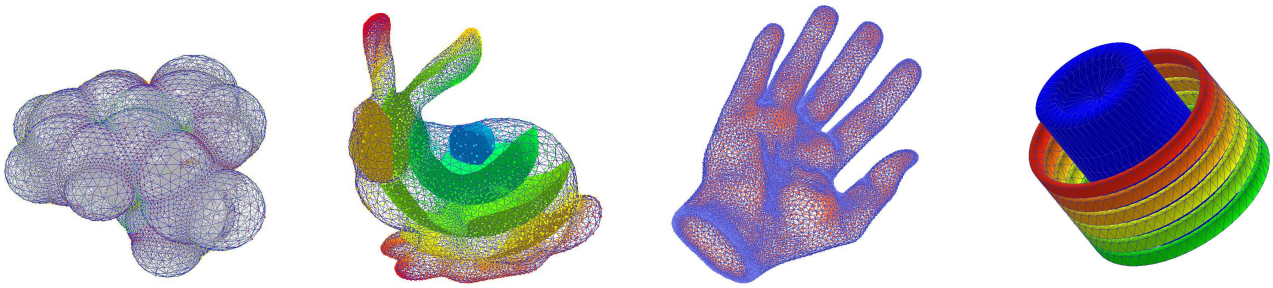

Figure 1. Visualization of tetrahedral meshes using CHF.

Abstract

This work introduces a scalable topological data structure for manifold tetrahedral meshes called Compact Half–Face *(*CHF*). It provides a high degree of scalability, since it is able to optimize the memory consumption / execution time ratio for different applications and data by using features of its different levels. An object–oriented API using class inheritance and virtual instantiation enables a unique interface for each function at any level.* CHF *requires very few memory, is simple to implement and easy to use, since it substitutes pointers by container of integers and basic bitwise rules.*

Keywords: *Geometric Modeling, Data Structures, Object Oriented Programming, Generic Containers.*

1. Introduction

Tetrahedral meshes constitute one of the fundamental representations for volumetric objects in computer graphics and geometric modeling. Although mesh–less models are very popular in visualization, meshes represent in a unique, local and explicit manner the underlying geometric space of an object. This is the main reason why these representations are omnipresent for finite element methods. For these applications and many others, the efficiency of the data structure is a crucial element. In particular, generation of meshes from point data is a very active area of research, where efficient data structures as CHF can help in both clarity and efficiency.

Nowadays, many geometric modeling and scientific visualization systems commonly have to deal with huge volumetric models [16, 20]. Several data structures and algorithm have been proposed to visualize and manipulate tetrahedral meshes [26, 18, 10, 21]. However, such structures consume considerable memory, requiring high scalability to handle large models without losing performance in memory access because of thrashing.

Contributions. This work introduces a scalable topological data structure for manifold tetrahedral meshes called Compact Half–Face (CHF). This data structure is an improvement for the *Handle–Face* data structure [18] and extends the *Corner–Table* data structure for surfaces [24]. The main advantages of CHF are threefold. First, it is scalable, since it is able to change the use of memory to improve execution time. The amount of information stored on the data structure can adapt to a specific application or model. Second, it requires very few memory, since it substitutes pointers by container of integers and basic bitwise rules. More-

over, it is very easy to implement and use. And last, its object–oriented API using class inheritance and virtual instantiation enables a unique interface of the same functions for every scale.

Paper outline. Section 2 introduces some basic concepts of combinatorial structure. Section 3 reviews some related works. Section 4 presents the CHF data structure, while section 5 studies the complexity of the basic operations on the CHF. Section 6 compares CHF to some of the related works.

2. Combinatorial structures for geometrical objects

This sections introduces the basic concepts that will be modeled by our data structure. For more details on the following definitions, see [1].

Simplices. A *simplex* σ^p of dimension p is the closed convex hull of $p+1$ points $\{v_0, \ldots, v_p\}, v_i \in \mathbb{R}^m$, in general position, i.e., when the vectors $v_1 - v_0, v_2 - v_0, \ldots, v_p - v_0$ are linearly independent. For example, simplices of dimensions 3, 2, 1 and 0 are, respectively, tetrahedrons, triangles, edges and vertices. The points $v_0, \ldots, v_p$ are called the *vertices* of σ. A *face* of σ is the convex hull of some but not all of the vertices of σ and therefore is also a simplex. If σ is a face of a simplex τ then τ is said to be *incident* to σ and τ *bounds* σ. The *boundary* of a p–simplex σ, denoted by $\partial\sigma$, is the collection of all of its faces.

Simplicial Complex. Two k–simplices σ and $\rho \in K$ are *adjacent* when $\sigma \cap \rho \neq \emptyset$, and *independent* otherwise. A *simplicial complex* K is a finite set of simplices together with all their faces such that if σ and τ are simplices of K, then they either meet at a face λ, or are independent.

Stars. The *open star* of a simplex $\sigma \in K$, denoted by $\mathrm{star}\,(\sigma, K)$, is the union of all simplices in K having σ as a face. The *star* of $\sigma \in K$, denoted by $\overline{\mathrm{star}}\,(\sigma, K)$, is the closure of $\mathrm{star}\,(\sigma, K)$, i.e the union of all simplices in K having σ as a face (see figure 2).

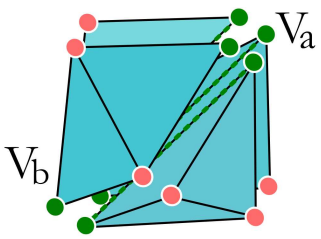

Figure 2. The star of an inner edge $\mathsf{v}_a\mathsf{v}_b$.

Manifolds. A simplicial complex M is a *combinatorial k–manifold* if the star of a vertex in M is homeomorphic either to $\mathbb{R}^k$ or to $\mathbb{R}^{k-1} \times \mathbb{R}_+$. In particular, if M is a manifold, then every $(k–1)$–simplex in M is bounding either one or two k–simplex.

A combinatorial k–manifold is *orientable* when it is possible to choose a coherent orientation for all of its simplices, where coherent means that two adjacent k–faces induce opposite orientations on their common $(k–1)$–face. From now on, a k–manifold will always mean an oriented combinatorial k–manifold and we will denote by $\mathbf{n}_k$ be the number of k–simplices in M.

Boundary. The $(k-1)$–simplices of a combinatorial k–manifold M that are incident to only one k–simplex are called *boundary simplices*. All faces of a boundary simplex are also called boundary simplices. The set of boundary simplices forms the *boundary* of M and is denoted by ∂M. The boundary of a combinatorial k–manifold is a combinatorial $(k–1)$–manifold without boundary. The simplices that are not on the boundary are called *interior simplices*. Figure 3 shows an example of interior and boundary cells of a combinatorial 3–manifold.

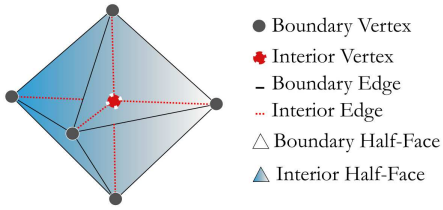

Figure 3. Example interior and boundary cells of a combinatorial 3–manifold.

Operations on manifolds. This works aims at defining an optimized data structure for representing 3–manifolds with or without boundary. According to [10], the necessary operations on such data structures are the retrieval functions $R_{pq}(\sigma)$, which returns for a p–simplex σ the q–simplices τ that bound common simplices.

For example $R_{00}(\mathsf{v})$ returns all the vertices in the star of v. More generally, when $p < q$, $R_{pq}(\sigma)$ consists of the set of q–simplices in $\overline{\mathrm{star}}\,(\sigma)$. When $p > q$, $R_{pq}(\sigma)$ consists of the set of q–simplices that are faces of σ. Finally, when $p = q$, $R_{pq}(\sigma)$ consists of the set of q–simplices τ such that σ and τ are both faces of a common simplex of M. All the data structures described in the next section implement part

or all of the R_{pq} relations, each with a different trade–off between memory consumption and execution time.

3. Previous and related works.

Data structures for surfaces. Since Baumgart's *Winged–Edge* data structure [2] for representing solids in $\mathbb{R}^3$, several modifications have been proposed in order to extend the range of objects to be modeled. Among those, Braid [4] introduced the concept of *loop* and Mäntylä [19] defined the *Half–Edge* data structure. Guibas and Stolfi [11] proposed a generalization to include non–orientable 2–manifolds implemented as the *Quad–Edge* data structure. Lopes [17, 7] defined the *Handle–Edge*, a new data structure that represents explicitly the boundary of a combinatorial surface, which allows a direct description of its topology at each operation. Rossignac et al. [24] proposed a very concise version of the half–edge data structure, called *Corner–Table*, which only uses two array of integers and a set of rules to represent triangular surfaces. A first scalable data structure for 2–manifold, name *Direct–Edges* was proposed by Campagna et al. in [6].

Data structure for 3–manifolds. For three dimensional manifolds representation, other data structures have been developed. Dobkin and Laszlo [9], with their *Facet–Edge* data structure, extended Guibas and Stolfi's scheme in order to represent cell complexes that are subdivisions of the 3–dimensional sphere. After that, Lopes and Tavares [18] introduced the *Handle–Face* data structure that explicitly represents the boundaries.

Data structure for n–manifolds. Generalizing the idea of the *Quad–Edge* and *Facet–Edge*, some significant works were introduced to represent n–dimensional manifold. Among those, the *Cell–Tuple* data structure by Brisson [5], the *n–Generalized Maps* for simplicial quasi–manifolds by Lienhardt [15] and the *Hypermaps* by Bertrand and Dufourd [3] are to be cited.

Cell representation. According to Brisson [5], among all the above cited dimension–independent data structures, only the *Quad–Edge* and the *Facet–Edge* do not explicit represents each cell, while the *Winged–Edge*, the *Half–Edge*, the *Handle–Edge*, and the *Handle–Face* are examples of data structures with explicit representation of cells. Another widely used dimension–independent explicit representation is the *Indexed data structure with adjacencies* proposed by Paoluzzi et al. [22].

Non–manifold data structures. Traditional non–manifold models usually results from Boolean operations. Weiler [26] was the first to propose a non–manifold data structure, called *Radial–Edge*. After that, several modifications have been suggested to deal with specific applications. Some substantial contributions to non–manifolds representation include the works of Wu [27, 28], Yamaguchi and Kimura [29], Gursoz [12], Rossignac and O'Connor [25], Cavalcanti et al. [8], and Lee and Lee [14]. More recently, de Floriani and Hui [10] presented a very concise data structure for non–manifold manipulation, called *NMIA*, which is an extension of the *Indexed data structure with adjacencies*, and Pesco et al. proposed the *Handle–Cell* [23], which is an extension of the *Handle–Edge*, to deal with general 2–dimensional cell complexes. Finally, Nonato et al. [21] proposed an extension of the *Handle–Face* in order to deal with special cases of singularities, named *Singular Handle–Face*. All non–manifold structures cited above represent the cells explicitly.

The CHF proposal. The CHF data structure has been created to represent 3–manifolds. It extends the *Corner–Table* [24] to volumes and at the same time can be considered a concise version of the *Handle–Face* [18], since its volumetric cells are simplices. It uses generic containers instead of pointers or static arrays. Similarly to the *Corner–Table*, it explicitly represents a few adjacencies and incidence relations between cells and uses a set of rules to obtain the others. However, CHF has a very different and important characteristic that is its scalability, since it can change in size according to the application.

4. The CHF Data Structure

The objective of this section is to introduce the CHF data structure for the representation of 3–dimensional manifolds with or without boundary. There are actually four levels of structures, each one completing the previous in order to accelerate the execution time, but consuming a little more memory at each step. Those levels are implicit to the programmer by virtual inheritance: a C++ feature that avoids the programmer to care about which structure level he is using. The compiler simply generates at the beginning of each function a piece of code like:

Operation $R_{pq}(\sigma)$ at level i;
if has_level i+1 **then**
  RETURN Operation $R_{pq}(\sigma)$ at level i+1;
end if
$(\dots)$

This section describes the four levels of the CHF data structure, with their constructions from the previous level. The next section will describe the main operations on each of those levels.

4.0. Level 0: representing tetrahedrons by the Vertex container

The CHF uses the concept of *half–face* (see figure 4) to represent the association of a tetrahedron with one of its bounding triangles, or equivalently the association of this triangle with its apex. Any access to the elements of a tetrahedron is performed through its half–faces. The level 0 if the CHF represents only the tetrahedral soup (see figure 5) by storing the apex of each half–faces.

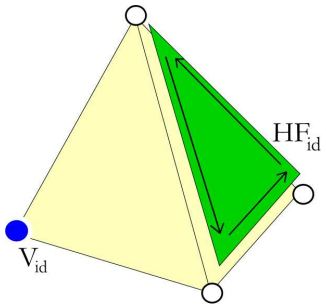

Figure 4. One half–face of a tetrahedron.

In the CHF data structure, the half–faces, the vertices and the tetrahedrons are indexed by non–negative integers. Each tetrahedron is represented by 4 consecutive half–faces that define its orientation. For example, half–faces 0, 1, 2 and 3 correspond to the first tetrahedron, the half–faces 4, 5, 6 and 7 correspond to the second tetrahedron and so on...

Vertex geometry. The representation of the mesh geometry is done by the use of a container, named $\mathsf{G}[]$, that stores the geometry (coordinates, normals,...) of the $\mathbf{n}_0$ vertices in M. For example, the geometry of a vertex with index V_{id} v is obtained by accessing $\mathsf{G}[\mathsf{v}]$.

Half–Faces' rules. A half–face with index HF_{id} hf is associated with the tetrahedron of index $\lfloor \mathsf{hf}/4 \rfloor$. Therefore, the indexes of the four half–faces that belong to the tetrahedron with index T_{id} t are $4\mathsf{t}, 4\mathsf{t}+1, 4\mathsf{t}+2$, and $4\mathsf{t}+3$. The next, middle and previous half–faces of a given half–face

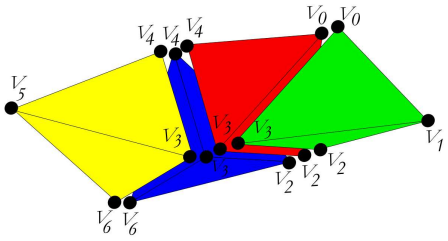

Figure 5. CHF **level 0.**

Half–Face	< Orientation >
HF_{id} $4\mathsf{t}$	$< \mathsf{V}[4\mathsf{t}+1], \mathsf{V}[4\mathsf{t}+2], \mathsf{V}[4\mathsf{t}+3] >$
HF_{id} $4\mathsf{t}+1$	$< \mathsf{V}[4\mathsf{t}+2], \mathsf{V}[4\mathsf{t}], \mathsf{V}[4\mathsf{t}+3] >$
HF_{id} $4\mathsf{t}+2$	$< \mathsf{V}[4\mathsf{t}+3], \mathsf{V}[4\mathsf{t}], \mathsf{V}[4\mathsf{t}+1] >$
HF_{id} $4\mathsf{t}+3$	$< \mathsf{V}[4\mathsf{t}], \mathsf{V}[4\mathsf{t}+2], \mathsf{V}[4\mathsf{t}+1] >$

Table 1. Orientation of the triangles of a generic tetrahedron T_{id} t.

HF_{id} hf on the tetrahedron $\lfloor \mathsf{hf}/4 \rfloor$ can be obtained by the use of the following rules:

$$\begin{aligned} \mathrm{next}_{\mathsf{hf}}(\mathsf{hf}) &:= 4\lfloor \mathsf{hf}/4 \rfloor + (\mathsf{hf}+1)\,\%4, \\ \mathrm{mid}_{\mathsf{hf}}(\mathsf{hf}) &:= 4\lfloor \mathsf{hf}/4 \rfloor + (\mathsf{hf}+2)\,\%4, \\ \mathrm{pref}_{\mathsf{hf}}(\mathsf{hf}) &:= 4\lfloor \mathsf{hf}/4 \rfloor + (\mathsf{hf}+3)\,\%4. \end{aligned}$$

Note that the arithmetic operations above can be coded efficiently with bitwise rules: $4\mathsf{t} := \mathsf{t} \ll 2$, $\lfloor \mathsf{hf}/4 \rfloor := \mathsf{hf} \gg 2$, $\mathsf{hf}\%4 := \mathsf{hf}\&3$ and $4\lfloor \mathsf{hf}/4 \rfloor := \mathsf{hf}\&(\sim 3)$.

The Vertex container. The association of each half-face HF_{id} hf to its apex is stored in a container of integers, named the Vertex container and denoted by $\mathsf{V}[]$. The integer $\mathsf{v} = \mathsf{V}[\mathsf{hf}]$ is the index of the apex of half-face hf (see figure 6). The size of $\mathsf{V}[]$ is $4\mathbf{n}_3$, and each entry of $\mathsf{V}[]$ varies from 0 to $\mathbf{n}_0 - 1$. Table 1 and figure 7 define the triangle orientation for each half–face of a tetrahedron with index t.

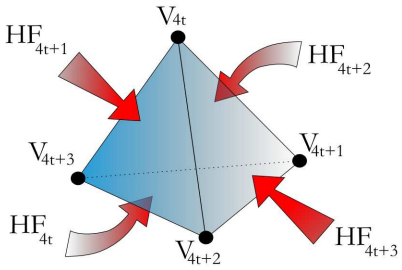

Figure 6. Insides of half-faces and vertices.

The Half–Edges. Similarly to the *Handle-Face* data structure, each half–face in the CHF is bounded by a cycle of three *half–edges* (see figure 4). However, in the CHF the half–edges are implicitly represented. The half–edge represents the association between a vertex of a half–face and the half–face itself. Table 1 fixes the orientation of the half–edge cycle inside a generic tetrahedron. For example, the cycle of half–edges inside half–face HF_{id} $4\mathsf{t}$ can be read as: $\mathsf{V}[4\mathsf{t}+1] \rightarrow \mathsf{V}[4\mathsf{t}+2], \mathsf{V}[4\mathsf{t}+2] \rightarrow \mathsf{V}[4\mathsf{t}+3]$, and $\mathsf{V}[4\mathsf{t}+3] \rightarrow \mathsf{V}[4\mathsf{t}+1]$.

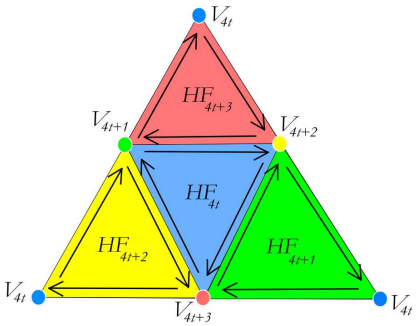

Figure 7. Orientation of the triangles of a generic tetrahedron T_{id} t.

Define the index of a half–edge in half–face HF_{id}hf as $(\mathsf{hf}, \mathsf{he})$, where he is the index of the half–face opposed to the initial vertex of the half–edge (denoted V_{he} on figures 8 and 9). Fixing a half–face hf, we can define its initial half–edge as $HE_{id0}\,(\mathsf{hf}) := (\mathsf{hf}, \mathrm{next}_{\mathsf{hf}}\,(\mathsf{hf}))$. With this notation, the next and previous half-edges of $(\mathsf{hf}, \mathsf{he})$ inside hf are, respectively:

$$\mathrm{next}_{\mathsf{he}}\,(\mathsf{hf}, \mathsf{he}) := (\mathsf{hf}, HE_{id0}\,(\mathsf{hf}) + N[\mathsf{he}\%4][\mathsf{hf}\%4]),$$
$$\mathrm{prev}_{\mathsf{he}}\,(\mathsf{hf}, \mathsf{he}) := (\mathsf{hf}, HE_{id0}\,(\mathsf{hf}) + P[\mathsf{he}\%4][\mathsf{hf}\%4]).$$

$$where : N = \begin{bmatrix} - & 3 & 1 & 2 \\ 2 & - & 3 & 0 \\ 3 & 0 & - & 1 \\ 1 & 2 & 0 & - \end{bmatrix} \; and \; P = N^t.$$

On one hand, the *mate* of a half–edge lives on the half–face opposed to its previous vertex:

$$\mathrm{mate}_{\mathsf{he}}\,(\mathsf{hf}, \mathsf{he}) = (\mathrm{prev}_{\mathsf{he}}\,(\mathsf{hf}, \mathsf{he})\,, \mathrm{next}_{\mathsf{he}}\,(\mathsf{hf}, \mathsf{he}))\,.$$

On the other hand, the *radial* of a half–edge lives on the opposite half–face, that will be defined in section 4.1 or that can be constructed by algorithm 1:

$$\mathrm{radial}_{\mathsf{he}}\,(\mathsf{hf}, \mathsf{he}) = (\mathsf{O}[\mathsf{hf}]\,, \mathrm{next}_{\mathsf{he}}\,(\mathsf{hf}, \mathsf{he}))\,.$$

With these four rules on half–edge, one can derive an algorithm similar to Weiler [26] to traverse all half–faces around an edge.

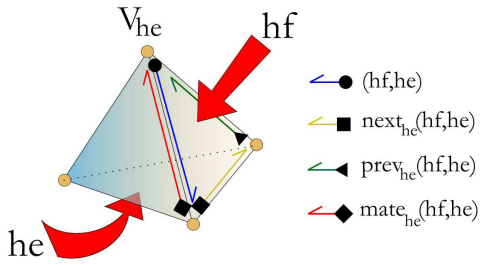

Figure 8. Mate half–edge.

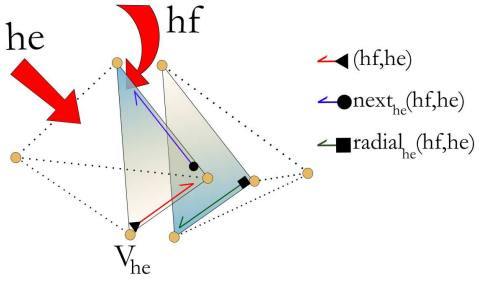

Figure 9. Radial half–edge.

4.1. Level 1: representing the adjacencies among tetrahedrons through the Opposite container

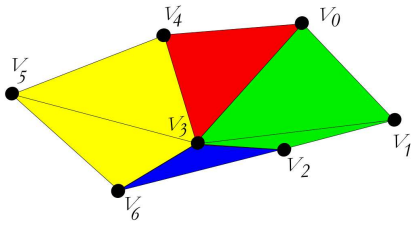

Figure 10. CHF level 1.

Level 1 of the CHF adds to the level 0 (figure 5) information on the neighbor of each tetrahedron (figure 10). Since M is a 3–manifold, each half–face is incident to one or two tetrahedrons. In order to explicitly represent the adjacency relation of two tetrahedrons, the CHF uses another container of integers, named the Opposite container, denoted by O[]. The face–adjacency between neighboring tetrahedrons is represented by associating to each half–face HF_{id} hf its opposite half–face O[hf] (figure 11), which has the same vertices but opposite orientation. If the half–face hf is on the boundary, then it doesn't have an opposite, which is encoded by $\mathsf{O}[\mathsf{hf}] = -1$. Thus, the value of O[hf] allows to directly checking whether a half–face hf is on the boundary or not. The size of O[] is $4\mathbf{n}_3$, and each entry of O[] varies from -1 to $\mathbf{n}_3 - 1$. Algorithm 1 shows how to efficiently construct the O[] container from the V[] container. This algorithm uses maps, which is a simple associative container.

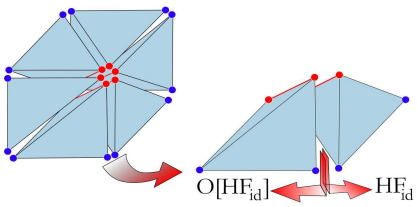

Figure 11. Opposite of a half-face.

Algorithm 1 Opposite container construction

```
1:  O[i] ← −1                                   // Inits the container
2:  map{V_id × V_id × V_id → HF_id} adjacency
3:  for HF_id hf ∈ {0 ... 4n_3 − 1} do
4:                                   // Gets the vertices tuple of hf
      v_0 ← V[next_hf (hf)] ; v_1 ← V[mid_hf (hf)] ;
      v_2 ← V[pref_hf (hf)] ; v^3 ← sort (v_0, v_1, v_2)
5:    if adjacency.find (v^3) then
6:      O[hf] ← adjacency [v^3]
7:      O[O[hf]] ← hf              // Found opposite half–face
8:      adjacency.erase (v^3)
9:    else                         // Temporarily stores half–face
10:     adjacency [v^3] ← hf
11:   end if
12: end for
```

4.2. Level 2: representing the cells explicitly

The Face map. An interior face will be identified by the lowest of its two half–face $F_{id} := HF_{id}$, since the other half–face can be obtained by the use of container O[]. A boundary half–face will be identified by its unique half–face. The faces can then be explicitly represented by a map FH mapping each face, identified by one of its half–face, to its attribute. Depending on the application, this attribute can be the normal vector, material color, differential quantity... The vertices of the faces can be easily accessed through the half–face identifying it. The map FH has exactly $\mathbf{n}_2$ entries, which can be allocated in $O(\mathbf{n}_2 log(\mathbf{n}_2))$ time and space using a classical red–black tree structure.

The Edge map. Incidence relations on edges are essential in many applications such as simplification and subdivision algorithms. An edge is identified by an ordered pair of integers $E_{id} :=< \mathsf{v}_1, \mathsf{v}_2 >$, where $\mathsf{v}_1 < \mathsf{v}_2$ are the indexes of the edge vertices. The edges can be explicitly represented by a map EH, mapping an edge identifier to the index of one of its incident half–faces, and eventually to its attributes such as color, collapse cost... 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 gives directly the classification of an edge as interior or boundary. The map EH has $\mathbf{n}_1$ entries, which can be allocated in $O(\mathbf{n}_1 log(\mathbf{n}_1))$.

The extra Vertex container. To compute simple geometry operators such as derivation, it is necessary to obtain the vertex star efficiently. Therefore, it is useful to store an extra container of integers VH that for each vertex v stores an index of a half–face incident to vertex v. In the case the vertex is on the boundary, the stored half–face should be a boundary one. Such container has size $O(\mathbf{n}_0)$ and can be constructed in time $O(\mathbf{n}_3)$.

4.3. Level 3: representing the boundary through a Compact Half–Edge

The boundary of a combinatorial 3–manifold is a set of combinatorial 2–manifolds without boundary. Lopes and Tavares in [18] pointed out the importance of having efficient boundary cells manipulation for building and unbuilding 3-manifolds. Moreover, advancing front triangulations is a very significant application that needs an explicit representation of the boundary. To do so, the incidences and adjacencies of cells on the boundary should be explicitly represented. In the CHF such representation is done by the use of the Compact Half–Edge (CHE) data structure, which is a version of the CHF for surfaces.

The CHE [13] is also a scalable data-structure. Similarly to CHF, the first level of CHE has only the Vertex container. The second level includes the Opposite container for the half–edges. The third level represents the cells explicitly. And finally, the fourth level introduces an explicit representation for the boundary curves.

Here, the CHF implements the second level of CHE, i.e., it uses two containers of integers, the bV[] container and the bO[] container. The vertices of CHE and CHF have the same index, and the CHE does not need to store the vertex geometry for level 0. Each boundary triangle is represented by 3 consecutive oriented–edges that define its frontier. In the CHE, the oriented–edges 0, 1, and 2 correspond to the first triangle, the oriented–edges 3, 4, and 5 correspond to the second triangle and so on... The bV[] container stores the indexes of the vertices of the boundary triangles. The bO[] container stores in each entry the index of the opposite oriented–edge on the boundary surface. Those two containers are obtained in linear time traversing V[] and O[].

5. Operations on the CHF for each level

This section discusses the computational performance of some R_{pq} relations at all levels of the CHF.

5.1. R_{0*}: the vertex star

The vertex star is essential in particular to volume modeling. The CHF answers relations R_{0*} in time $O(\mathbf{n}_3)$ at level 0, since the function has to transverse all the V[] container. At level 1, the V[] container is traversed until one half–face incident to the input vertex is found, after that the vertex star is obtained in time $O(deg(\mathsf{v}))$ by the use of the O[] and the rules described above. Thus, the worst case at level 1 has complexity $O(\mathbf{n}_3)$, but it is in average $deg(\mathsf{v})$ times faster than for level 0. Finally, at level 2 and 3 the complexity of finding the star of a vertex v is reduced to $O(deg(\mathsf{v}))$, since VH directly stores the starting half–face to traverse the vertex star.

5.2. R_{1*}: the edge star

Algorithm 2 R_{13} $(E_{id} < \mathsf{v}_1, \mathsf{v}_2 >)$, level 0

```
1: container< V_id > R_13
2: for HF_id hf ∈ {0 ... 4n_3 − 1} do    // all half–faces
3:    if v_1 == V[hf] and
            (v_2 = V[next_hf (hf)] or v_2 = V[mid_hf (hf)]
                 or v_2 = V[pref_hf (hf)]) then
4:       R_13.insert(hf ≫ 2) ; continue  // add tetrahedron
5:    end if
6: end for
```

The edge star is also very important to several kind of algorithms, such as volume simplification schemes using edge collapse. Here, the edge is considered as an ordered pair of indexes of its vertices. Thus, R_{10} is directly answered. At level 0, the time complexity for the CHF to answer the relations R_{12} and R_{13} is $O(\mathbf{n}_3)$, since the method has to transverse all the V[] container (see algorithm 2). At level 1, again the V[] container is traversed to find a half–face incident to the input edge, after that by the use of the O[] container and the rules for $mate$ and $radial$, the cycle of half–faces around the edge is obtained in time $O(deg(\mathsf{e}))$, where $deg(\mathsf{e})$ is the number of faces incident to e. The worst case for level 1 has complexity time $O(\mathbf{n}_3)$ but it is in average $deg(\mathsf{e})$ times faster than for level 0. Finally, at level 2 and 3 the complexity is reduced to $O(deg(\mathsf{e}))$, since *EH* directly identifies the first half–face (see algorithm 3).

Algorithm 3 R_{13} $(E_{id} < \mathsf{v}_1, \mathsf{v}_2, (\mathsf{hf}_0, \mathsf{he}_0) >)$, level 3

```
1: container< V_id > R_13
2: T_id t_0 ← ⌊hf_0/4⌋                          // first tetrahedron
3: HF_id hf ← hf_0 ; HE_id he ← he_0 ; T_id t ← t_0
4: repeat
5:    R_13.insert(t);                           // insert to the result
6:    (hf, he) ← radial_he (hf, he)             // radial mate
7:    t ← ⌊hf/4⌋                                // next tetrahedron
8: until hf ≠ −1 and t ≠ t_0
```

5.3. R_{2*}: the face star

Obtaining the face star is simpler than the edge and vertex star. Since a face is identified by one of its half–face hf, we can get its vertices directly by $\mathsf{v}_0 = \mathsf{V}[\mathrm{next}_{\mathsf{hf}}(\mathsf{hf})]$, $\mathsf{v}_1 = \mathsf{V}[\mathrm{mid}_{\mathsf{hf}}(\mathsf{hf})]$, $\mathsf{v}_2 = \mathsf{V}[\mathrm{pref}_{\mathsf{hf}}(\mathsf{hf})]$, answering R_{20} in constant time at any level. The edges $\mathsf{v}_0\mathsf{v}_1$, $\mathsf{v}_1\mathsf{v}_2$, $\mathsf{v}_2\mathsf{v}_0$, can also be retrieved, answering R_{21} in constant time at any level. At level 0, to answer R_{22} and R_{23} the methods have to transverse all the V[] container to find the opposite half–face if any, therefore the time complexity is $O(\mathbf{n}_3)$ for both relations. At level 1 and above, the complexity is reduced to $O(1)$, since the face is identified by one of its incident half–faces and the O[] container gives the other one directly.

5.4. R_{3*}: tetrahedron incidence and adjacency

All the incidences of a tetrahedron are answered in constant time at all levels of the CHF, except at level 0 where the query for adjacent tetrahedrons, R_{33}, is answered in $O(\mathbf{n}_3)$. At level 1, 2 and 3 by the use of the O[] container, the query is obtained in $O(1)$ time.

6. Memory Comparisons

	memory consumption
Handle–Face	$135\mathbf{n}_3 + 10\mathbf{n}_2 + 12\mathbf{n}_1 + 10\mathbf{n}_0$
CHF_0	$4\mathbf{n}_3$
CHF_1	$8\mathbf{n}_3$
CHF_2	$8\mathbf{n}_3 + \mathbf{n}_2 \log_2 \mathbf{n}_2 + \mathbf{n}_1 \log_2 \mathbf{n}_1 + \mathbf{n}_0$
CHF_3	CHF_2 + $6\,\mathrm{card}\left(\partial M^2\right)$

Table 2. Memory complexity for topology.

All data structures described in section 3 provide different trade–offs between memory usage and time complexity of basic operations such as identifying a simplex, accessing its vertices, computing its stars. When handling huge amount of data, a programmer has always to balance the memory consumption with the time complexity. We compared CHF with the unique structure specific to 3–manifolds that has an explicit representation for cells and for the boundary: the *Handle–Face* [18] (see table 2). In a real context, the structure can be adapted in three complementary ways. First, the programmer can choose to use a specific level for the whole program. Second, a set of rules can be defined to decide dynamically which level offers the best memory consumption / execution time trade–off. Last, the whole structure can be first reserved, dynamically filled each time a query is performed and completed when filled more than a given ratio.

7. Conclusions and Future Works

The main contribution of this paper is an introduction of an efficient data structure for 3–manifold representation, called CHF. The CHF is straightforward to use, concise and easy to implement. Moreover, it can adapt its size according to its necessity by the use of inheritance. The use of containers makes the CHF implementation clear and general.

In order to illustrate its use, figure 1 shows some simple visualization examples using the CHF. The first picture is an example of a wire-frame visualization of a scalar field

defined on a molecule. The second illustrates some isosurfaces of a scalar field defined on the Stanford Bunny. The third one shows in blue the boundary vertices and in red the interior ones. Finally, the last picture shows in different colors the several boundary components of the volume mesh.

The authors plan to extend the CHF in order to consider non-manifolds. In that case, the scalable level structure of CHF is used again by adding one level to deal with 3-complexes with singular vertices, and another one to deal with singular edges. This extension is straightforward in both cases, replacing the map containers used in CHF by multi-maps.

Acknowledgments

The authors would like to thank Pierre Alliez (INRIA) for the volumetric models. Marcos Lage is supported by CAPES. Hélio Lopes is partially supported by CNPq and FAPERJ (contract E261170.693/2004).

References

[1] J. W. Alexander. The combinatorial theory of complexes. *Annals of Mathematics*, 31:219–320, 1930.

[2] B. G. Baumgart. A Polyhedron Representation for Computer Vision. *AFIPS National Computer Conference*, 44:589–596, 1975.

[3] Y. Bertrand and J.-F. Dufourd. Algebraic Specification of a 3D–Modeler Based on Hypergraphs. *CVGIP Graphical Models and Image Processing*, 56:29–60, 1994.

[4] I. C. Braid, R. C. Hillyard, and I. A. Stroud. Stepwise construction of polyhedra in geometric modeling. In K. W. Brodlie, editor, *Mathematical Methods in Computer Graphics and Design*, pages 123–141. Academic Press, 1980.

[5] E. Brisson. Representing Geometric Structures in d Dimensions: Topology and Order. *Discrete and Computational Geometry*, 9:387–426, 1993.

[6] S. Campagna, L. Kobbelt, and H.-P. Seidel. Directed edges: A scalable representation for triangle meshes. *Journal of Graphics Tools*, 3(4):1–11, 1998.

[7] A. Castelo, H. Lopes, and G. Tavares. Handlebody Representation for Surfaces and Morse Operators. In *Curves and Surfaces in Computer Vision Graphics III*, pages 270–283, 1992.

[8] P. Roma Cavalcanti, P. C. P. Carvalho, and L. F. Martha. Non–Manifold modelling: an approach based on spatial subdivision. *Computer–Aided Design*, 29(3):209–220, 1997.

[9] D. P. Dobkin and M. J. Laszlo. Primitives for the manipulation of three–dimensional subdivisions. *Algorithmica*, 4:3–32, 1989.

[10] L. de Floriani and A. Hui. A scalable data structure for three–dimensional non–manifold objects. In *Symposium on Geometry processing*, pages 72–82. ACM, 2003.

[11] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *Transactions on Graphics*, 4:74–123, 1985.

[12] E. L. Gursoz, Y. Choi, and F. B. Prinz. Vertex–Based Representation of Non–Manifold Boundaries. In J. U. Turner, M. J. Wozny, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 107–130. Elsevier, 1990.

[13] M. Lage, T. Lewiner, H. Lopes, and L. Velho. CHE: A scalable topological data structure for triangular meshes. Technical report, PUC — Rio de Janeiro, 2005.

[14] S. Lee and K. Lee. Partial Entity Structure: A Compact Non–Manifold Boundary Representation Based on Partial Topological Entities. In *Solid Modeling and Applications*, pages 159–170. ACM, 2001.

[15] P. Lienhardt. N–dimensional Generalized Combinatorial Maps and Cellular Quasi–Manifolds. *Journal of Computational Geometry & Applications*, 4:275–324, 1994.

[16] H. Lopes, G. Nonato, S. Pesco, and G. Tavares. Dealing with topological singularities in volumetric reconstruction. In P.-J. Laurrent, P. Sablonière, and L. Schumaker, editors, *Curve and Surface Design*, pages 229–238, Saint Malo, 2000. Vanderbilt University Press.

[17] H. Lopes. *Algorithm to build and unbuild 2 and 3 dimensional manifolds*. PhD thesis, Department of Mathematics, PUC–Rio, 1996.

[18] H. Lopes and G. Tavares. Structural operators for modeling 3–manifolds. In C. Hoffman and W. Bronsvort, editors, *Solid Modeling and Applications*, pages 10–18. ACM, 1997.

[19] M. Mäntylä. *An Introduction to Solid Modeling*. Computer Science Press, Rockville, 1988.

[20] G. Nonato, R. Minghim, M. C. F. de Oliveira, and G. Tavares. A Novel Approach for Delaunay 3D Reconstruction with a comparative analysis in the Light of Applications. *Computer Graphics Forum*, 20(2):161–174, 2001.

[21] G. Nonato, A. Castelo, R. Minghim, and H. Hideraldo. Topological tetrahedron characterization with application in volume reconstruction. *Journal of Shape Modeling*, 11(2), 2005.

[22] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension–independent modeling with simplicial complexes. *Transactions on Graphics*, 12(1):56–102, 1993.

[23] S. Pesco, H. Lopes, and G. Tavares. A Stratification Approach for Modeling 2–cell complexes. *Computers & Graphics*, 28(2):235–247, 2004.

[24] J. Rossignac, A. Safonova, and A. Szymczak. 3D Compression Made Simple: Edgebreaker on a Corner–Table. In *Shape Modeling International*, pages 278–283. IEEE, 2001.

[25] J. Rossignac and M. A. O'Connor. SGC : A Dimension Independent Model for Pointsets with Internal Structures and Incomplete Boundaries. In J. U. Turner, M. J. Wozny, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 145–180. Elsevier, 1990.

[26] K. J. Weiler. *Topological Structures for Geometric Modeling*. PhD thesis, Rensselaer Polytechnic Institute, New York, USA, 1986.

[27] S. T. Wu. A new combinatorial model for boundary representation. *Computers & Graphics*, 13(4):477–486, 1989.

[28] S. T. Wu. Non–manifold data models: implementation issue. In *MICAD*, Computer Graphcis and Computer Aided Technologies, pages 37–56, 1992.

[29] Y. Yamaguchi and F. Kimura. Non–Manifold Topology Based on Coupling Entities. *Computers & Graphics*, 15(1):42–50, 1995.