CHE: A scalable topological data structure for triangular meshes

MARCOS LAGE[1], THOMAS LEWINER[1,2], HÉLIO LOPES[1] AND LUIZ VELHO[3]

[1] Department of Mathematics — Pontifícia Universidade Católica — Rio de Janeiro — Brazil
[2] Géométrica Project — INRIA – Sophia Antipolis — France
[3] Visgraf Project — IMPA — Rio de Janeiro — Brazil
{mlage, tomlew, lopes}@mat.puc--rio.br. lvelho@visgraf.impa.br.

Abstract. This work introduces a scalable topological data structure for manifold triangular meshes called Compact Half–Edge (CHE). 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. CHE 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

Triangular meshes constitute one of the fundamental representations for geometrical 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 CHE can help in both clarity and efficiency.

Nowadays, many geometric modeling and scientific visualization systems commonly have to deal with huge models. Several data structures and algorithm have been proposed to visualize and manipulate triangular meshes. 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 triangular meshes called Compact Half–Edge (CHE). This data structure is an improvement for the *Handle–Edge* data structure [13] and extends the *Corner–Table* data structure [18]. The main advantages of CHE 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. Moreover, it is very easy to implement and use. And last, its API using class inheritance and virtual instantiation enables a unique interface of the same functions for every scale.

Paper outline. Section 2 *Combinatorial structures for geometrical objects* introduces some basic concepts of combinatorial structure. Section 3 *Previous and related works* reviews some related works. Section 4 *The CHE Data Structure* presents the CHE data structure, while section 5 *Operations on the CHE for each level* studies the complexity of the basic operations on the CHE. Section 6 *Memory Comparisons* compares CHE to some of the related works.

2 Combinatorial structures for geometrical objects

This section 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 2, 1 and 0 are, respectively, 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.

Preprint MAT. 13/05, communicated on May 15^{th}, 2005 to the Department of Mathematics, Pontifícia Universidade Católica — Rio de Janeiro, Brazil.

Stars. The *open star* of a simplex $\sigma \in K$, is the union of all simplices in K having σ as a face. The *star* of $\sigma \in K$, denoted by $\mathrm{star}\,(\sigma, K)$, is the union of all simplices of the open star together with all their faces (see Figure 1).

Manifolds. A simplicial complex M is a *combinatorial* k–*manifold* if the open 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*.

Operations on manifolds. This works aims at defining an optimized data structure for representing 2–manifolds with or without boundary. According to [8], 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 $\mathrm{star}\,(\sigma)$. 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.

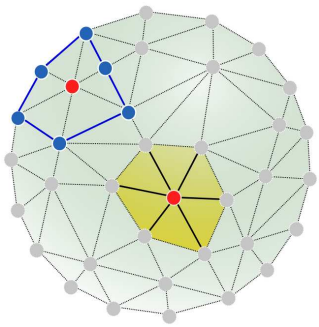

Figure 1: *The star (in yellow) and the link (in blue) of an inner vertex* v.

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 [3] introduced the concept of *loop* and Mäntylä [14] defined the *Half–Edge* data structure. Guibas and Stolfi [9] proposed a generalization to include non–orientable 2–manifolds implemented as the *Quad–Edge* data structure. Lopes [12, 6] 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. [18] 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 [5].

Cell representation. According to Brisson [4], 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* 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. [15].

Non–manifold data structures. Traditional non–manifold models usually results from Boolean operations. Weiler [19] 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 [20, 21], Yamaguchi and Kimura [22], Gursoz [10], Rossignac and O'Connor [17], Cavalcanti et al. [7], and Lee and Lee [11]. More recently, de Floriani and Hui [8] 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* [16], which is an extension of the *Handle–Edge*, to deal with general 2–dimensional cell complexes. All the above structures represent the cells explicitly.

The CHE ***proposal.*** The CHE data structure has been created to represent 2–manifolds. It extends the *Corner–Table* [18] by its scalability and at the same time can be considered a concise version of the *Handle–Edge* [13]. 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, CHE has a very different and important characteristic that is its scalability, since it can change in size according to the application.

4 The CHE Data Structure

The objective of this section is to introduce the CHE data structure for the representation of 2–dimensional mani-

folds 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 CHE data structure, with their construction from the first level. The next section will describe the main operations on each of those levels.

(a) Level 0: representing triangles by the Vertex container

The CHE uses the concept of *half–edge* (see Figure 2) to represent the association of a triangle with one of its bounding edges, or equivalently the association of this edge with its apex. Any access to the elements of a triangle is performed through its half–edges. The level 0 if the CHE represents only the triangle soup (see Figure 3) by storing the apex of each half–edge.

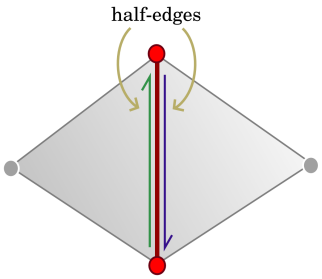

Figure 2: *Two half–edges associated to the same edge.*

In the CHE data structure, the half–edges, the vertices and the triangles are indexed by non–negative integers. Each triangle is represented by 3 consecutive half–edges that define its orientation. For example, half–edges 0, 1 and 2 correspond to the first triangle, the half–edges 3, 4 and 5 correspond to the second triangle and so on...

Vertex geometry. The representation of the mesh geometry is done by the use of a container, named 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 G[v].

Half–Edges' rules. A half–edge with index HE_{id} he is associated with the triangle of index $\lfloor \mathsf{he}/3 \rfloor$. Therefore, the indexes of the three half–edges that belong to the triangle with index T_{id} t are $3\mathsf{t}$, $3\mathsf{t}+1$, and $3\mathsf{t}+2$. The next and previous half–edges of a given half–edge HE_{id} he on triangle $\lfloor \mathsf{he}/3 \rfloor$ can be obtained by the use of the following rules (Figure 4):

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

The Vertex container. The association of each half-edgeHE_{id} he to its apex is stored in a container of integers, named the Vertex container and denoted by V[]. The integer $\mathsf{v} = \mathsf{V}[\mathsf{he}]$ is the index of the apex of half-edge he (see Figure 5). The size of V[] is $3\mathbf{n}_2$, and each entry of V[] varies from 0 to $\mathbf{n}_0 - 1$.

(b) Level 1: representing the adjacencies among triangles through the Opposite container

Level 1 of the CHE adds to the level 0 (Figure 3) information on the neighbor of each triangle (Figure 6). Since M is a 2–manifold, each half–edge is incident to one or two triangles. In order to explicitly represent the adjacency relation of two triangles, the CHE uses another container of integers, named the Opposite container, denoted by O[]. The edge–adjacency between neighboring triangles is represented by associating to each half–edge HE_{id} he its opposite half–edge O[he] (Figure 7), which has the same vertices but opposite orientation. If the half–edge he is on the boundary, then it doesn't have an opposite, which is encoded by $\mathsf{O}[\mathsf{he}] = -1$. Thus, the value of O[he] allows to directly checking whether a half–edge he is on the boundary or not. The size of O[] is $3\mathbf{n}_2$, and each entry of O[] varies from -1 to $\mathbf{n}_2 - 1$. Algorithm 1 shows how to efficiently construct the O[] container from the V[] container. This algorithm uses maps, which is a simple associative container.

(c) Level 2: representing the cells explicitly

The Edge map. Incidence relations on edges are essential in many applications such as simplification and subdivision algorithms. An edge is identified by its half–edge of lower index. The edges can be explicitly represented by a map EH, mapping an edge identifier to the index of one of its incident half–edges, and eventually to its attributes such as color, collapse cost... The map EH has $\mathbf{n}_1$ entries, which can be allocated in $O(\mathbf{n}_1 log(\mathbf{n}_1))$.

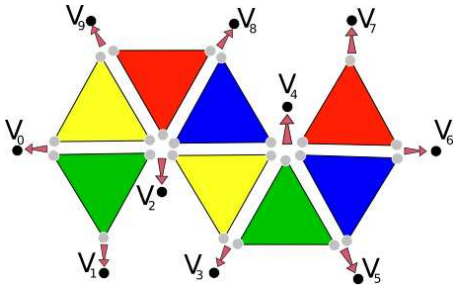

Figure 3: CHE *level 0.*

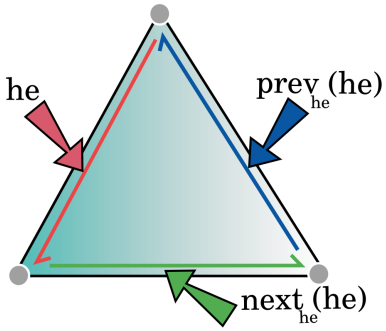

Figure 4: *Next and previous half–edges.*

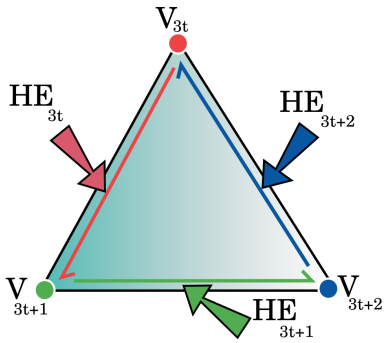

Figure 5: *Insides of half-edges and vertices.*

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–edge incident to vertex v. In the case the vertex is on the boundary, the stored half–edge should be a boundary one. Such container has size $O(\mathbf{n}_0)$ and can be constructed in time $O(\mathbf{n}_2)$.

(d) Level 3: representing the boundary through a Half–Edge List

The boundary of a combinatorial 2–manifold is a set of combinatorial 1–manifolds without boundary. [13] pointed out the importance of having efficient boundary cells manipulation for building and unbuilding manifolds. Moreover, advancing front triangulations is a very significant application that needs an explicit representation of the boundary. In the CHE such representation is done by storing for each boundary curve $\mathcal{C}$ an identifier (half–edge) of an edge of $\mathcal{C}$. The boundary curve can then be traversed by the star operations on the triangulation.

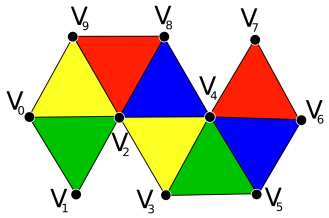

Figure 6: CHE *level 1.*

Algorithm 1 Opposite container construction

```
1:  O[i] ← −1                                   // Inits the container
2:  map{V_id × V_id → HE_id} adjacency
3:  for HE_id he ∈ {0 ... 3n_2 − 1} do
4:                                   // Gets the vertices pair of he
      v_0 ← V[prev_he (he)] ; v_1 ← V[next_he (he)] ;
      v^2 ← sort (v_0, v_1)
5:    if adjacency.find (v^2) then
6:      O[he] ← adjacency [v^2]
7:      O[O[he]] ← he              // Found opposite half–edge
8:      adjacency.erase (v^2)
9:    else                         // Temporarily stores half–edge
10:     adjacency [v^2] ← he
11:   end if
12: end for
```

5 Operations on the CHE for each level

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

(a) R_{0*}: the vertex star

The vertex star is essential in particular to volume modeling. The CHE answers relations R_{0*} in time $O(\mathbf{n}_2)$ at level 0, since the function has to transverse all the V[] container. At level 1, the V[] container is traversed until one half–edge 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 $O(\mathbf{n}_2)$, 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–edge to traverse the vertex star.

(b) R_{1*}: the edge star

Obtaining the edge star is simpler than the vertex star. Since an edge is identified by one of its half–edges he, we can get its vertices directly by $\mathsf{v}_0 = \mathsf{V}[\mathrm{next}_{\mathsf{he}}(\mathsf{he})]$ and

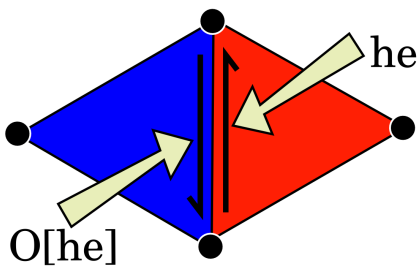

Figure 7: *Opposite of a half-edge.*

$\mathsf{v}_1 = \mathsf{V}[\mathrm{prev}_{\mathsf{he}}(\mathsf{he})]$, answering R_{10} in constant time at any level. At level 0, to answer R_{11} and R_{12} the methods have to transverse all the V[] container to find the opposite half–edge if any, therefore the time complexity is $O(\mathbf{n}_2)$ for both relations. At level 1 and above, the complexity is reduced to $O(1)$, since the edge is identified by one of its incident half–edges and the O[] container gives the other one directly.

(c) R_{2*}: triangle incidence and adjacency

All the incidences of a triangle are answered in constant time at all levels of the CHE, except at level 0 where the query for adjacent triangles, R_{22}, is answered in $O(\mathbf{n}_2)$. 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–Edge	$204\mathbf{n}_0$
Corner–Table	$52\mathbf{n}_0$
Directed–Edges *Small*	$48\mathbf{n}_0$
Directed–Edges *Medium*	$72\mathbf{n}_0$
Directed–Edges *Full*	$120\mathbf{n}_0$
CHE *Nvel* 0	$24\mathbf{n}_0$
CHE *Nvel* 1	$48\mathbf{n}_0$
CHE *Nvel* 2	$112\mathbf{n}_0$
CHE *Nvel* 3	$112\mathbf{n}_0$

Table 1: *Memory complexity for topology, compared with classical data structures.*

All data structures described in section 3 *Previous and related works* 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 CHE with different structures for 2–manifolds (see table 1). 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 2–manifold representation, called CHE. The CHE 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 CHE implementation clear and general.

The authors plan to extend the CHE in order to consider non-manifolds. In that case, the scalable level structure of CHE is used again by adding one level to deal with 2-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 CHE by multimaps.

Acknowledgments

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] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

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

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

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

[17] 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.

[18] 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.

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

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

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

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