Eurographics Symposium on Geometry Processing (2003)
L. Kobbelt, P. Schröder, H. Hoppe (Editors)

A scalable data structure for three-dimensional non-manifold objects

Leila De Floriani and Annie Hui

Department of Computer and Information Sciences, University of Genova, Via Dodecaneso, 35 - 16146 Genova (Italy).
Department of Computer Science, University of Maryland, College Park, MD 20742 (USA)

Abstract

In this paper, we address the problem of representing and manipulating non-manifold, mixed-dimensional objects described by three-dimensional simplicial complexes embedded in the 3D Euclidean space. We describe the design and the implementation of a new data structure, that we call the non-manifold indexed data structure with adjacencies (NMIA)*, which can represent any three-dimensional Euclidean simplicial complex compactly, since it encodes only the vertices and the top simplexes of the complex plus a restricted subset of topological relations among simplexes. The NMIA structure supports efficient traversal algorithms which retrieve topological relations in optimal time, and it scales very well to the manifold case. Here, we sketch traversal algorithms, and we compare the NMIA structure with data structures for manifold and regular 3D simplicial complexes.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling -*Curve, surface, solid and object representations*

1. Introduction

Non-manifold objects are subsets of the Euclidean space which can be regarded as combinations of wire-frame, surface, solid and cellular decompositions. Informally, a *manifold* object is a subset of the Euclidean space for which the neighborhood of each internal point is locally equivalent to an open ball. Objects that do not fulfill this property at one or more points are called *non-manifold* objects.

As pointed out by several authors [5, 16, 27, 22, 29], in a modeling system we need to represent non-manifold objects since Boolean operators are closed in the non-manifold domain, sweeping or offset operations may generate parts of different dimensionalities, non-manifold topologies are required in different product development phases, such as conceptual design, analysis and manufacturing. Furthermore, most objects encountered in the applications contain a relatively small number of non-manifold *singularities.* Thus, it is important to develop representations that scale well with the degree of "non-manifoldness" of the object.

Non-manifold objects can be effectively described through simplicial or cell complexes with a non-manifold and non-regular (i.e., with parts of different dimensionalities) domain. The contribution of this work is in designing and implementing a data structure for describing three-dimensional simplicial complexes embedded in the 3D Euclidean space, i.e., combinations of tetrahedral meshes with lower dimensional entities described by a chain of edges, or by a triangle mesh.

The trade-off when designing a data structure for a simplicial, or cell complex is among its expressive power, its storage cost and the efficiency of the algorithms for traversing the complex (which are based on primitives for retrieving incident and adjacent entities to a given one). To this aim, we have developed a data structure that satisfies the following requirements: (i) to be as compact as possible, (ii) to support retrieval of incidence and adjacency relations among the entities in the complex in optimal time, (iii) to be able to describe all kinds of non-manifold objects in 3D space partitioned into a 3D simplicial complex, and (iv) to be scalable, i.e., to exhibit just a low overhead cost due to the representation of non-manifold singularities.

Our data structure, that we call the *non-manifold indexed data structure with adjacencies (NMIA)*, encodes only the vertices and the top simplexes of the complex, plus a restricted subset of topological relations among simplexes.

For comparison purposes, we also specialize a general data structure for cell complexes, the *incidence graph* [12], to describe a certain class of simplicial complexes, and we call the resulting data structure the *simplified incidence graph*. We compare our data structure with both the simplified incidence graph and with a well-known data structure for a class of simplicial complexes, the *indexed data structure with adjacencies*.

The remainder of this paper is organized as follows. In Section 2, we summarize some background notions, which are necessary for understanding the related materials in this work. In Section 3, we review some related work. In Section 4, we describe the NMIA data structure. In Section 5, we present an implementation of this data structure and discuss its storage cost. In Section 6, we present the algorithms for retrieving topological relations for a simplicial complex described by the NMIA data structure. In Section 7, we describe the simplified incidence graph, and we present a comparison of the NMIA structure with such data structure and with the indexed data structure with adjacencies. Finally, in Section 8, we draw some conclusions and discuss future work.

2. Background

In this Section, we review some basic combinatorial notions about simplicial complexes in arbitrary dimensions, and we introduce the topological relations among the cells of a complex. We use *abstract simplicial complexes* as basic tools to capture the combinatorial structure of Euclidean simplicial complexes.

2.1. Simplicial Complexes

Let V be a finite set of points that we call *vertices*. An *abstract simplicial complex* on V is a subset Σ of the set of (non-empty) parts of V such that $\{v\} \in \Sigma$ for every point $v \in V$, and if $s \subset V$ is an element of Σ, then every subset of s is also an element of Σ [18]. Each element of Σ is called an *abstract simplex*.

The *dimension* of a simplex $s \in \Sigma$, denoted $dim(s)$, is defined by $dim(s) = |s| - 1$, where $|s|$ is the number of vertices in s. A simplex of dimension k is called a k-simplex. A complex Σ is called *d-dimensional*, or a *d-complex*, if $\max_{s\in\Sigma}(dim(s)) = d$. Each d-simplex of a d-complex Σ is called a *maximal simplex* of Σ.

The *boundary* $b(s)$ of a simplex s is defined as the set of all proper parts of s. Simplexes ξ in $b(s)$ are called *faces* of s. Similarly, the *co-boundary*, or *star*, of a simplex s is defined as $\star s = \{\xi \in \Sigma \mid s \subset \xi\}$. Simplexes ξ in $\star s$ are called *co-faces* of s. The *link* of a simplex s is the set of all faces of co-faces of s, that are not incident at s. Any simplex s such that $\star s = s$ is called a *top simplex* of Σ. In the following, we will call *restricted star* of a simplex s, $\star s - \{s\}$, and we will denote it as $st(s)$.

Two distinct simplexes are said to be *incident* if one of them is a face of the other. Two simplexes are called *k-adjacent* if they share a k-face. Two p-simplexes, with $p > 0$, are said to be *adjacent* if they are $(p-1)$-adjacent. Two vertices (i.e., 0-simplexes) are called *adjacent* if they are both incident at a common 1-simplex. Two simplexes that are neither incident nor adjacent are said to be *disjoint*.

An *h-path* is a sequence of simplexes $(s_i)_{i=0}^{k}$ such that two successive simplexes s_{i-1}, s_i are h-adjacent. Two simplexes s and s' are *h-connected*, if and only if there exists an h-path $(s_i)_{i=0}^{k}$ such that s is a face of s_0 and s' is a face of s_k. A subset Σ' of a complex Σ is called *h-connected* if and only if every pair of its vertices are h-connected. Any maximal h-connected sub-complex of a complex Σ is called an *h-connected component* of Σ. The term *connected* is used as a shortcut for 0-connected.

A d-complex Σ where all top simplexes are maximal (i.e., of dimension d) is called *regular*, or *uniformly d-dimensional*. A $(d-1)$-simplex s in a d-complex Σ is a *manifold $(d-1)$-simplex* if and only if there are at most two d-simplexes incident at s. Otherwise, s is called a *non-manifold* $(d-1)$-simplex. A regular $(d-1)$-connected d-complex in which all $(d-1)$-simplexes are manifold is called a *(combinatorial) pseudo-manifold* (possibly with boundary). A pseudo-manifold satisfying the additional property that all its vertices have a link combinatorially equivalent either to the $(d-1)$-dimensional sphere or to the $(d-1)$-dimensional ball is called a *(combinatorial) manifold*.

A Euclidean simplex of dimension d is the convex hull of $d+1$ linearly independent points in the n-dimensional Euclidean space E^n, with $d \leq n$. We simply call a *Euclidean d-simplex* a *d-simplex* when the context is understood: a 0-simplex is a *vertex*; a 1-simplex an *edge*; a 2-simplex a *triangle*; a 3-simplex a *tetrahedron*. Any Euclidean k-simplex t generated by a set $V_t \subseteq V_s$ of cardinality $k+1 \leq d$ is called a *k-face* of s.

A finite collection Σ of Euclidean simplexes is a *Euclidean simplicial complex* when both (i) for each simplex $s \in \Sigma$, all faces of s belong to Σ, and (ii) for each pair of simplexes s and s', either $s \cap s' = \emptyset$ or $s \cap s'$ is a face of both s and s'.

The *domain*, or *carrier*, of a d-dimensional Euclidean simplicial complex Σ embedded in E^n, with $d \leq n$, is the subset of E^n defined by the union, as point sets, of all the simplexes in Σ. The combinatorial structure of a Euclidean simplicial complex is an abstract simplicial complex. The domain of a Euclidean simplicial complex which is described by a combinatorial manifold is a manifold in E^n. Whenever no ambiguity arises, we will use the term *simplex* to denote an abstract, or a Euclidean, simplex. We will also use the term *complex* to denote an abstract, or a Euclidean, simplicial complex.

2.2. Topological Relations

Let Σ be a d-complex. Let $s \in \Sigma$ be a p-simplex, with $0 \leq p \leq d$. For each integer value q, $0 \leq q \leq d$, we define the topological relation $R_{pq}(s)$ as a retrieval function that returns the q-simplexes of Σ that are not disjoint from s. In particular:

- For $p < q$, $R_{pq}(s)$ consists of the set of simplexes of order q in the star of s.
- For $p > q$, $R_{pq}(s)$ consists of the set of simplexes of order q in the set of faces of s.
- For $p > 0$, $R_{pp}(s)$ is the set of p-simplexes in Σ that are $(p-1)$-adjacent to s.
- $R_{00}(v)$, where v is a vertex, consists of the set of vertices w such that $\{v, w\}$ is a 1-simplex of Σ.

Relation R_{pq} is called a *boundary relation* if $p > q$, a *co-boundary relation* if $p < q$ and an *adjacency relation* if $p = q$. Boundary and co-boundary relations together are called *incidence relations*.

3. Related Work

Most of the existing work in the literature focuses on the two-dimensional boundary representation of three-dimensional objects, and on the manifold domain. In the manifold domain, several data structures have been proposed for representing the decomposition of the boundary of a three-dimensional manifold into a simplicial complex [1, 11, 15, 17, 21, 23, 28]. The approach proposed in [15] has been generalized to manifold complexes in three and higher dimensions [11, 2], while the half-edge data structure has been extended to the three dimensional case in [21]. The compact corner table data structure [28] has been generalized to arbitrary non-manifold meshes.

Most work in the context of *non-manifold* modeling has been done in two dimensions for representing the boundary of non-manifold, non-regular objects. The first proposal for a topological data structure for boundary representation of non-manifold objects is the *radial-edge structure* [29]. In [16], Gursoz et al. describe a vertex-based data structure, called the *tri-cyclic cusp* structure, which extends the radial-edge structure by maintaining also inclusion relations between the local neighborhoods of a vertex. A similar structure has been introduced by Yagamuchi and Kimura [30]. A more compact data structure, called the *partial entity structure* [19] has been more recently proposed: it has been shown to require half of the space of the radial-edge structure. The storage costs of all such data structures do not scale with the number of non-manifold singularities, since they have been developed under the assumption that objects contain several non-manifold joints. The radial-edge data structure has been specialized in [24] to the case of two-dimensional simplicial meshes.

In [8], a compact data structure for non-manifold and non-regular two-dimensional simplicial complexes has been presented, which encodes both connectivity and adjacency information with a small memory overhead, and scales very well to the manifold case. A compact scalable edge-based data structure for non-manifold two-dimensional simplicial complexes has been proposed by Campagna et al. [4], which also scales well to manifold meshes, but it is restricted to regular complexes. A few proposals exist for modeling shapes in arbitrary dimensions through cell complexes. Selective Geometric Complexes (SGCs) [27] can describe objects through cell complexes whose cells can be either open, or not simply connected. In SGCs, cells and their mutual adjacencies are encoded in an incidence graph [12], which is a complete, but a verbose data structure. N-G-maps [20] are an implicit representation for a sub-class of pseudo-manifolds, called quasi-manifolds, but they are also quite space-consuming. The winged representation [25] can describe d-dimensional pseudo-manifold simplicial complexes, i.e., just regular ones. It generalizes to arbitrary dimensions the so-called incidence data structure with adjacency commonly used for triangle and tetrahedral meshes [3].

An alternative approach to the design of non-manifold data structures consists of decomposing a non-manifold object into simpler and more manageable parts [7, 10, 13, 14]. The proposals in [10, 13, 14] are restricted to modeling two-dimensional regular complexes, which describe the boundary of a solid object. In [7], a sound decomposition for d-dimensional non-manifold objects described through simplicial complexes is defined, which is unique and produces a description of an abstract d-complex (not necessarily embeddable in the Euclidean space) as a combination of nearly manifold components. A data dimension-independent data structure for such decomposition is defined in [9], which describes the components and their connectivity in a two-level representation.

4. Design of the Data Structure

In this Section, we introduce the design choices performed and the elements (entities and topological relations) of a data structure for describing three-dimensional simplicial complexes embedded in the three-dimensional Euclidean space.

4.1. Design Issues

In our data structure, we want to be able to describe 3D simplicial complexes containing also one- and two-dimensional top simplexes, that we call *wire-edges* and *dangling faces*, respectively, in order to represent parts of different dimensionalities in the object. Moreover, we need to describe situations in which the restricted star of an edge, or of a vertex. consists of more than one connected component, in order to represent edges and vertices in the object for which the manifold condition does not hold. Note that, since we are dealing with are 3-complexes embedded in E^3, any 2-simplex, which is not a top simplex, must be on the boundary of either one or two simplexes.

The above singularities can be summarized as the presence of:

1. several connected components in the restricted star of some vertex (called a *nm-vertex*) (see Figure 1);
2. 1-dimensional top simplexes (*wire-edges*) (see Figure 2);
3. several connected components in the restricted star of some edge (called a *nm-edge*) (see Figure 3);
4. 2-dimensional top simplexes (*dangling faces*) (see Figure 4).

In all the figures, tetrahedra are in solid grey of two intensities; dangling faces are shaded; edges and vertices of interest are highlighted in bold.

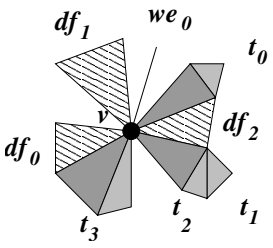


Figure 1: *$st(v)$ has four connected components*

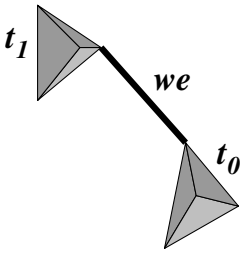


Figure 2: *a wire-edge, we, connecting two tetrahedra*

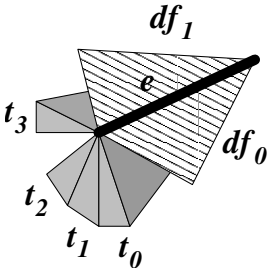


Figure 3: *$st(e)$ has four connected components*

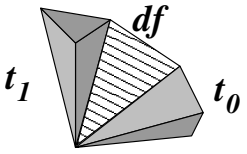


Figure 4: *a dangling face, df, connecting two tetrahedra*

Thus, the data structure must describe correctly the previous four cases. Cases 1 and 3 involve connectivity issues, and are discussed below.

4.2. Singularities at a non-manifold vertex

To take care of the connectivity information at an nm-vertex, we observe that the restricted star of a nm-vertex v consists of several connected components. We call each such component a *vertex-based cluster*. If two simplexes s_i, s_j belong to the same vertex-based cluster in $st(v)$, then there exists a 1-connected path passing through only those simplexes which form the cluster. These clusters cannot be ordered around v.

As no ordering is possible, the basic strategy of moving around a non-manifold vertex is to perform a breadth-first search and visit all simplexes that are related through relations R_{33}, R_{23}, or R_{32} in the same cluster (to be described in Section 6).

4.3. Singularities at a non-manifold edge

The restricted start of an nm-edge consists of several connected components. We call each such component an *edge-based cluster*. Edge-based clusters can be ordered around the edge, for instance, in counter-clockwise direction. Also, if simplexes s_i, s_j belong to the same connected component in $st(e)$, then there exists a 2-connected path from s_i to s_j that traverses through only those simplexes that are incident at e. This implies that a cluster consists either of a single dangling face, or of a collection of tetrahedra fanning out from the nm-edge. This is an interesting property for navigation: when we want to find all the elements within an edge-based cluster, we simply move from one tetrahedron to the next one by using relation R_{33}. When we finish examining a cluster, we need to find the next cluster. To this aim, for each simplex s_i of dimension 2 or higher, we keep track of its left and right neighbors around each of its edges when necessary. This condition is described as follows:

- If a simplex s_i is the only element of a cluster, then both its left and right neighbors (which may be identical) would belong to some other clusters around e. These two simplexes are the left and right neighbors of s_i with respect to edge e.
- If the simplex on the left of s_i belongs to the same cluster as s_i, then we say s_i has no left neighbor with respect to edge e.
- A symmetric case holds for the right neighbor.

For example, consider the object in Figure 3. In Figure 5, we show on the right a cross-section of the star of edge e, The table in Figure 5 shows the edge-based clusters associated with the nm-edge e by considering the clusters in a counter-clockwise order.

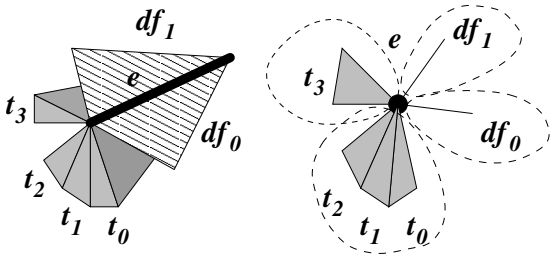
4.4. Entities and Topological Relations

4.4.1. Entities

Given a simplicial complex Σ, the *non-manifold indexed data structure with adjacencies (NMIA)* encodes all vertices (0-simplexes), wire-edges (top 1-simplexes), dangling faces (top 2-simplexes), and tetrahedra (3-simplexes) of Σ. Other 1- and 2-simplexes are not explicitly represented.

To describe edge- and vertex-based clusters, we introduce three relations, that represent the incidence of the edge-based clusters on an edge, and of the vertex-based clusters on a vertex:

- Relation $R_{0,clusters}(v)$ is a retrieval function which associates, with vertex v, one representative k-simplex for each vertex-based cluster incident at v. Let us consider the object in Figure 1 as an example. The object is reproduced in

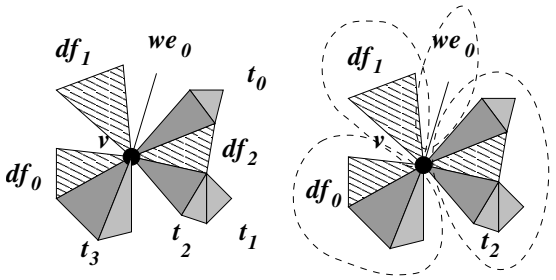


Entity	left neighbor at e	right neighbor at e
df_0	t_0	df_1
df_1	df_0	t_3
t_3	df_1	t_2
t_2	t_3	empty
t_1	empty	empty
t_0	empty	df_0

Figure 5: *Clusters around a nm-edge e and their neighboring relations.*

Figure 6. There are four clusters incident at vertex v. We take one representative from each of these clusters. One possible way is shown on the bottom of Figure 6.

- Relation $R_{2,clusters}(f)$ is a retrieval function which associates, with each edge e_i of a dangling face f, the edge-based clusters incident at e_i, in a counter-clockwise ordered sequence.
- Relation $R_{3,clusters}(s)$ is a retrieval function which associates, with each edge e_i of a 3-simplex s, the edge-based clusters incident at e_i, in a counter-clockwise ordered sequence.



cluster:	1	2	3	4
representative chosen:	we_0	df_1	df_0	t_2

Figure 6: *Vertex-based clusters at the vertex shown in bold, and choice of representatives for each cluster shown in the table*

Thus, for each tetrahedron t, we encode the following relations:

- Relation $R_{30}(t)$, which associates, with each tetrahedron t, its four vertices ordered according to the orientation chosen for t.
- Relation $R_{33}(t)$, which associates, with each tetrahedron t, the four tetrahedra adjacent to t through a 2-simplex, (the i-th tetrahedron in $R_{33}(t)$ is the one that does not contain the i-th vertex of t).
- Relation $R_{3,clusters}(t)$, as defined above, for each of the six edges of tetrahedron t (considered in an order compatible with the orientation of t).

For each dangling face f, we encode the following relations:

- Relation $R_{20}(f)$, which associates, with each dangling face f, its three vertices (ordered according to the orientation chosen for f).
- Relation $R_{2,clusters}(f)$, as defined above, for each of the three edges of dangling face f (considered in an order compatible with the orientation of f).

For each wire-edge e, we encode relation $R_{10}(e)$, which associates, with edge e, its two extreme vertices. Note that there cannot exist edge-based clusters incident at a wire-edge e. Finally, for each vertex v, we encode relation $R_{0,clusters}(v)$, as defined above. Note that only one representative for each vertex-based cluster is maintained.

5. Implementation of the NMIA Data Structure

In this Section, we describe our implementation of the NMIA data structure, and we evaluate its storage cost. Although the space needed to index one entity is just $log_2 m$ bits, where m denotes the total number of such entities in the data structure, in our implementation, for simplicity, we use one integer (represented on 32 bits) to index an entity. We consider references and indexes as integers also in the comparison presented in Section 7.

Our present design aims at supporting both efficient traversal and mesh modifications. Short dynamic arrays are used to store relations to make local modifications on connectivity simpler.

For each entity (vertex, wire-edge, dangling face and tetrahedron), we store a one-bit flag for temporarily marking a simplex as having been visited. These flags have to be reset after each query. We encode the $R_{30}(t)$, $R_{20}(f)$ and $R_{10}(e)$ relations as arrays of indexes to the four, three and two vertices of a tetrahedron t, of a dangling face f, and of a wire-edge e, respectively.

For each tetrahedron t, relation $R_{33}(t)$ is encoded as a 2-bit flag f_{33}, a (possibly empty) array containing the 2-adjacent neighbors of t and a reference to this latter array. A null reference means that t has no 2-adjacent neighbors. Otherwise, $f_{33}+1$ is equal to the number of 2-adjacent tetrahedra.

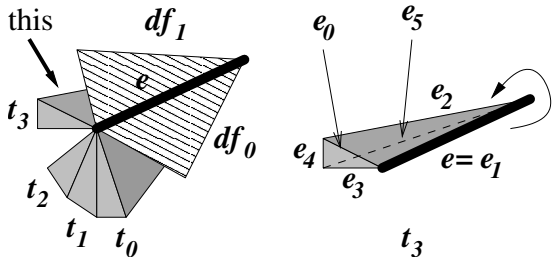
Relation $R_{3,clusters}(t)$ stores the edge-based cluster infor-

mation for each of the six edges of t. For each edge e of t, it encodes:

- a 4-bit flag f_{3c} to indicate the types of left and right neighbors that t has at e. The meanings of the flag are reported in Table 1. Note that, if f_{3c} is equal to 1, 2, 3, or 6, then t has one neighboring cluster at e; if f_{3c} is equal to 4, 5, 7, or 8, then t has two neighboring clusters at e. Otherwise, it has no neighbor. Since t has six edges, six flags and up to twelve neighbors have to be stored, and thus a total of 24 bits for each tetrahedron.
- an integer array of cluster entities containing the indexes either to a tetrahedron or to a dangling face. The size of this array is exactly equal to the number of cluster entities to be stored.
- a reference to the cluster entity array.

flag	left neighbor type	right neighbor type
0	empty	empty
1	empty	dangling face
2	empty	tetrahedron
3	dangling face	empty
4	dangling face	dangling face
5	dangling face	tetrahedron
6	tetrahedron	empty
7	tetrahedron	dangling face
8	tetrahedron	tetrahedron

Table 1: *Meanings of the values of the f_{3c} flag*



Edge:	e_0	e_1	e_2	e_3	e_4	e_5
Flag value:	0	5	0	0	0	0

Values of array: 1 2

Figure 7: *Encoding of the neighboring clusters of tetrahedron t_3 with respect to its six edges*

As an example, let us consider the complex in Figure 7. Consider tetrahedron t_3. We label with e_0,.., e_5 the six edges of t_3, as shown in Figure 7 on the right. The neighboring cluster information at the six edges are encoded with the flags shown in the table in Figure 7. Edge e_1 has flag value 5 because the left neighbor of t_3 at edge e_1 is a dangling face (df_1), and the right neighbor is a tetrahedron (t_2). At all other edges, t_3 appears as a cluster by itself, i.e., it has no left nor right neighbors at other edges. Thus, all flag values are 0. From the values of the six flags, we can tell that there are only two entities related to t_3 in $R_{3,cluster}(t_3)$, whose indexes are stored in the cluster entity array. We do not need to store their types since such information are already encoded by the flags.

For each dangling face f, relation $R_{2,clusters}(f)$ is encoded in a completely similar way as $R_{3,clusters}$ for tetrahedra. Since a dangling face has three edges, $R_{2,clusters}$ is scaled accordingly.

Finally, relation $R_{0,clusters}(v)$ for a vertex v is encoded by using an array of 2-bit flags and an integer array of indexes, each being an index to one representative (a wire-edge, a dangling face, or a tetrahedron) of a cluster at v. The flag indicates whether the representative is a wire-edge, a dangling face, or a tetrahedron. As an example, let us consider the complex depicted in Figure 6. The particular choice we have made is encoded in the $R_{0,clusters}(v)$ relation as shown in Table 2.

representative chosen:	we_0	df_1	df_0	t_2
flag:	0	1	1	2
index of entity:	0	1	0	2

Table 2: *Encoding of $R_{0,clusters}(v)$ for the non-manifold vertex v of Figure 6*

We have designed and implemented an algorithm for building the NMIA data structure by starting from the collection of all the top simplexes in the complex. The input consists of all vertex coordinates plus the list of the top simplexes, where each top simplex is expressed through the indexes of its vertices. The algorithm reconstructs the adjacency information and detects the non-manifold situations around vertices and edges. The current version is intended to build the data structure statically to work off-line. In this case, a preprocessing step is taken to make the connectivity information at each simplex available.

5.1. Storage costs

Let n_t, d, w, n_v denote the number of tetrahedra, dangling faces, wire-edges, and vertices in a simplicial complex, respectively. Let b be the number of boundary faces in the complex (i.e., those triangles that belong to exactly one tetrahedron). Let c_e and c_v be the total number of clusters over all the nm-edges and nm-vertices, respectively.

For the purpose of comparison with other data structures, we also define the following quantities. Let n_f be the total

number of faces (including dangling faces). Let n_e be the total number of edges (including wire-edges).

Let k_t be the total number of neighboring edge-based clusters stored over all tetrahedra, and k_d be the total number of neighboring edge-based clusters stored over all the dangling faces. Thus, we have that $2c_e = k_t + k_d$, because every edge-based cluster appears twice, once as a left and once as a right neighbor.

We need $3n_v$ doubles for vertex coordinates, $n_t + d + w + n_v$ bits for the one-bit navigation flag. In the following, we report the space requirements for each encoded relation:

- R_{30}: $4n_t$ integers
- R_{20}: $3d$ integers
- R_{10}: $2w$ integers
- R_{33}: $2n_t$ bits for flags + $(4n_t - b)$ integers for the R_{33} array + n_t integers for references to the R_{33} array
- $R_{3,clusters}$: $24n_t$ bits for flags (of which each tetrahedron has 6) + k_t integers for the cluster arrays + n_t references to the cluster arrays
- $R_{2,clusters}$: $12d$ bits for flags (of which each dangling face has 3) + k_d integers for the cluster arrays + d references to the cluster arrays
- $R_{0,clusters}$: $2c_v$ bits for flags + c_v integers for the cluster arrays storing cluster representatives + n_v integers keeping tracking of the length of each cluster array + n_v references to the array.

Now, we evaluate and compare the space requirements of the NMIA structure in the general case, in the case of pseudo-manifold complexes and in the case of manifold complexes. We assume that reference and indexes are stored as 32 bits integers.

- General non-manifolds:

  geometric information: $3n_v$ doubles
  topological entities: $(n_t + d + w + n_v)$ bits
  topological relations: $(10n_t - b + 4d + 2w + 2n_v + 2c_e + c_v)$ integers + $(26n_t + 12d + 2c_v)$ bits.

- Pseudo-manifolds:
  Pseudo-manifold complexes do not have dangling faces and wire-edges (thus, $d = w = 0$), but they may have non-manifold edges and non-manifold vertices. The space requirements thus become:

  geometric information: $3n_v$ doubles
  topological entities: $(n_t + n_v)$ bits
  topological relations: $(10n_t - b + 2n_v + 2c_e + c_v)$ integers + $(26n_t + 2c_v)$ bits

- Manifolds:
  Manifold complexes do not have dangling faces, wire-edges and non-manifold edges. Each vertex has exactly one cluster. So $d = w = c_e = 0$ and $c_v = n_v$. The storage requirements thus become:

  geometric information: $3n_v$ doubles
  topological entities: $(n_t + n_v)$ bits
  topological relations: $(10n_t - b + 3n_v)$ integers + $(26n_t + 2n_v)$ bits

In practical applications, it has been shown that $n_t \approx 6n_v$ [6]. Thus, the space requirements for a manifold are approximately equal to $73n_v$ integers + $3n_v$ doubles.

6. Retrieving Topological Relations

In this Section, we discuss how topological relations can be retrieved in optimal, or almost optimal, time from the NMIA data structure. This means that all non-stored relations can be retrieved in time linear in the number of entities involved in the specific relation, that is, for instance, the edges incident at a given vertex v (the R_{01} relation) can be extracted in time linear in the number of edges incident at v. Such retrieval algorithms are the basis for developing efficient traversal algorithms through the complex described by the data structure.

Retrieving those relations which are explicitly stored in the data structure, i.e., R_{30} and R_{33}, for tetrahedra, R_{20}, for dangling faces, R_{10}, for wire-edges, takes constant time.

Retrieving boundary relations $R_{32}(t)$ and $R_{31}(t)$ provides as results the faces of tetrahedron t, specified as triplets of vertices, and the edges of tetrahedron t, specified as pairs of vertices, respectively, and can be performed in constant time.

To retrieve boundary relation $R_{21}(f)$, we need to specify face f in case f is not a dangling face. Thus, if f is a boundary face of a tetrahedron t, then we specify f as *face(t, i)*, that is the i-th face of t, where $i = 0, .., 3$. The 1-simplexes involved in $R_{21}(f)$ are again specified as pairs of vertices. Thus, relation $R_{21}(f)$ is retrieved in constant time.

Relation $R_{23}(f)$ can be extracted only for boundary faces of tetrahedra. Face f is specified as *face(t, i)*. The retrieval algorithm makes use of relation $R_{33}(t)$ to find the other tetrahedron sharing f, if it exists. This takes constant time.

Retrieving relations $R_{12}(e)$ and $R_{13}(e)$ involves navigating around an edge e. The corresponding retrieval algorithms are implemented in a very similar fashion. The edge being queried can be a boundary edge of a dangling face or of a tetrahedron, but not a wire-edge. An edge of a dangling face f is addressed as *edge(f, i)* for $i = 0, ..., 2$ and an edge of a tetrahedron t is *edge(t, j)* for $j = 0, ..., 5$. We use relations $R_{3,clusters}$ or $R_{2,clusters}$ depending on whether e is an edge of a tetrahedron, or of a dangling face. If a cluster is a collection of tetrahedra that fan out from edge e, we use the R_{33} relation to move from one tetrahedron to the next in the same cluster, and R_{30} to make sure that both of them are incident at e.

We illustrate through the example in Figure 8 how to retrieve both $R_{12}(e)$ and $R_{13}(e)$ relations, i.e., how to navigate around an edge e in counter-clockwise direction. To this aim, we perform the following steps:

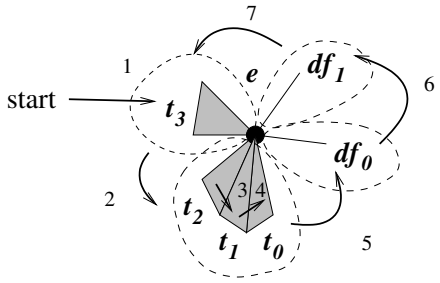


Figure 8: *Navigating through the entities incident at a non-manifold edge e (here we show a view perpendicular to edge e, as shown in Figure 6*

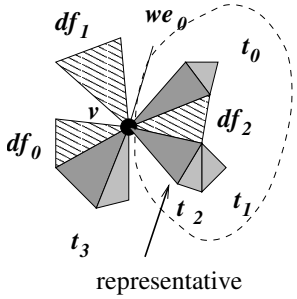


Figure 9: *The cluster selected for navigation*

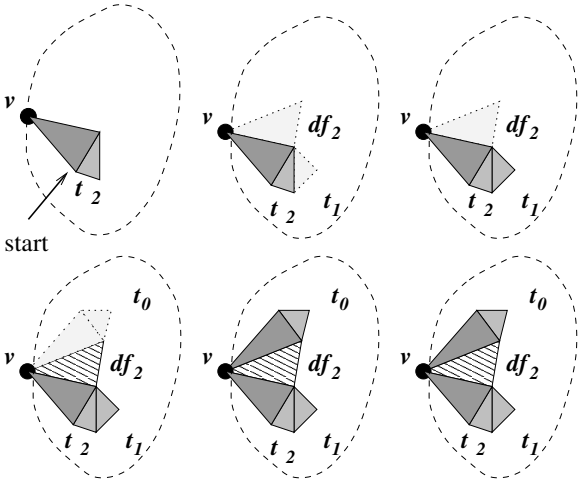


Figure 10: *Navigation of the circled cluster*

1. We start with e being an edge of t_3.
2. Using $R_{3,clusters}(t_3)$, we retrieve the left and right neighbors of t_3 with respect to e. These are df_1 and t_2, respectively. Then, we move to t_2.
3. From $R_{3,clusters}(t_2)$ we see that t_2 has no right neighbor. Thus, relation $R_{33}(t_2)$ allows us to extract all the tetrahedra which are face-adjacent to t_2. By using $R_{30}(t_2)$, we can select t_1 as the only tetrahedron that is also incident at e. Then, we move to t_1.
4. From $R_{3,clusters}(t_1)$ we see that t_1 has no right neighbor. Thus, again we use the $R_{33}(t_1)$ to retrieve the tetrahedra which are face-adjacent to t_1 and $R_{30}(t_1)$ to select those incident at e. This gives t_0 and t_2, and thus we move to t_0.
5. From $R_{3,clusters}(t_0)$ we see that the right neighbor of t_0 is df_0. Thus, we move to df_0.
6. From $R_{2,clusters}(df_0)$ we see that the right neighbor of df_0 is df_1. Thus, we move to df_1.
7. From $R_{2,clusters}(df_1)$ we see that the right neighbor of df_1 is t_3. Thus, we are done, since we started with t_3.

It can be easily seen that retrieving relation $R_{12}(e)$ takes a time linear in the number of faces incident at e, i.e., it can be performed in optimal time. Retrieving relation $R_{13}(e)$ takes $O(|R_{12}(e)|)$ time, where $|R_{12}(e)|$ denotes the number of faces incident at e, because of the possible presence of dangling faces.

Relation $R_{22}(f)$ is basically retrieved in the same way as relation $R_{12}(e)$, one for each edge of face f. Therefore, it takes $O(|R_{22}(f)|)$ time, which is optimal.

Retrieving relations $R_{00}(v), R_{01}(v), R_{02}(v)$, and $R_{03}(v)$ requires a breadth-first search over all the clusters incident at vertex v. Within each cluster, entities that are incident at v are traversed by using R_{33}, R_{30}, $R_{3,clusters}$, and $R_{2,clusters}$ relations.

Again we illustrate, through an example, how to retrieve all tetrahedra, dangling faces and wire-edges incident at a given non-manifold vertex v (see Figures 9 and 10). In the same way, we can retrieve all $R_{0i}(v)$ relations, where $i = 0,1,2,3$. Note that we use a queue to guide the breadth-first traversal of the star of v and that we mark a simplex as visited when we insert it into the queue. We perform the following steps:

1. We start with only t_2 in the queue, which we have obtained from $R_{0,clusters}(v)$.
2. Using $R_{33}(t_2)$, we find that t_1 is incident at v. Using $R_{3,clusters}(t_2)$, we find that df_2 is also incident at v. Thus, t_1 and df_2 are inserted into the queue.
3. Now, we dequeue t_1 and examine $R_{33}(t_1)$ and $R_{3,clusters}(t_1)$. This gives t_2 and df_2, that are already marked as visited.
4. Then, we dequeue df_2. and we examine relation $R_{2,clusters}(df_2)$. This gives t_0, t_1 and t_2, among which only t_0 is not visited. Thus, we insert t_0 into the queue.
5. We dequeue t_0 and we examine the $R_{33}(t_0)$ and $R_{3,clusters}(df_2)$ relations. This gives df_2, which has already been visited.
6. The traversal is finished, since the queue is empty. Reset the flags of all the simplexes retrieved.

Retrieving relations $R_{00}(v)$ and $R_{01}(v)$ takes optimal time. As in the case of R_{13}, both $R_{02}(v)$ and $R_{03}(v)$ relations can be retrieved in $O(|R_{00}(v)|)$ time due to the possible presence of tetrahedra, for $R_{02}(v)$, or of dangling faces, for $R_{03}(v)$.

Finally, by using R_{01} and R_{10} relations, we can retrieve the $R_{11}(e)$ relation, i.e., all the edges incident at the extreme vertices of edge e, in optimal time.

Note that in our design, we specify a face or an edge, which is not a top simplex, implicitly in terms of one the top simplexes (such as tetrahedra, dangling faces) containing it. We use top simplexes as entities because the retrieval primitives are usually used in the context of algorithms which traverse the complex by moving from one top simplex to another.

In case a k-simplex σ is specified explicitly through the indexes of its $(k+1)$ vertices, and we need to retrieve the index of the simplex, if σ is a top simplex, relation $R_{00}(v_i)$ must be retrieved for each vertex v_i of σ. In the manifold case the same strategy is commonly used, since a manifold simplicial complex is usually encoded with the indexed data structure with adjacencies.

7. Comparisons

In this Section, we present another data structure, that we call the *simplified incidence graph*, which is a simplified version of the incidence graph [12] for simplicial pseudo-manifolds, we review the *indexed data structure with adjacencies* for simplicial pseudo-manifolds, and we compare the NMIA data structure with the former two.

7.1. Simplified Incidence Graph

The incidence graph [12] is a data structure for d-dimensional cell complexes, in which every i-cell is stored as an entity, and boundary topological relations $R_{i,i-1}$ for $i = 1,...,d$ as well as co-boundary relations $R_{i,i+1}$ for $i = 0,...,d-1$ are encoded. For simplicial complexes, boundary relations are constant.

In the case of three-dimensional simplicial pseudo-manifolds, co-boundary relations $R_{12}(e)$ and $R_{01}(v)$ can be retrieved in optimal time (i.e., linear in the number of simplexes in $R_{12}(e)$ and $R_{01}(v)$, respectively), by storing, for $R_{12}(e)$, one representative 2-simplex for each edge-based cluster incident at edge e, and, for $R_{01}(v)$, one representative edge for each vertex-based cluster incident at v. In the manifold case, each edge and each vertex has only one cluster. We call the incidence graph in which we store just one representative for $R_{01}(v)$ and $R_{12}(e)$ relations the *simplified incidence graph*.

The simplified incidence graph stores all 0-, 1-, 2- and 3-simplexes. It encodes the coordinates of each vertex ($3n_v$ doubles for the 3D coordinates) and the following relations:

- relation $R_{32}(t)$: the four faces on the boundary of tetrahedron t ($4n_t$ integers).
- relation $R_{21}(f)$: the three edges on the boundary of face f ($3n_f$ integers).
- relation $R_{10}(e)$: the two vertices on the boundary of edge e ($2n_e$ integers).
- relation $R_{23}(f)$: the one, or two tetrahedra that are incident at f ($4n_t$ integers).
- relation $R^*_{12}(e)$: only one representative edge from each edge-based cluster that is incident at e ($c_e + n_e - k$ integers, where k is the number of nm-edges).
- relation $R^*_{01}(v)$: only one representative edge from each vertex-based cluster that is incident at vertex v (c_v integers).

We can evaluate the storage requirements of the data structure in the case of pseudo-manifolds and manifolds:

- Pseudo-manifolds:

  geometric information: $3n_v$ doubles
  topological relations: $(8n_t + 3n_f + 3n_e + c_e + c_v - k)$ integers

- Manifolds: $c_e = k = 0$ and $c_v = n_v$

  geometric information: $3n_v$ doubles
  topological relations: $8n_t + 3n_f + 3n_e + n_v$ integers

  If we assume that $n_t \approx 6n_v$, the space requirements are about $131n_v$ integers + $3n_v$ doubles.

It can be shown that all topological relations can be extracted in optimal time from the simplified incidence graph.

7.2. Indexed Data Structure with Adjacencies

The indexed data structure with adjacencies is a compact data structure for simplicial pseudo-manifolds in arbitrary dimensions, which encodes only 0- and d-simplexes and their connectivity. We describe here the three-dimensional instance of this data structure.

The indexed data structure encodes the vertex coordinates ($3n_v$ doubles), all 0- and 3-simplexes plus the following relations:

- relation $R_{30}(t)$: the four vertices on the boundary of t ($4n_t$ integers).
- relation $R_{33}(t)$: the four tetrahedra that are adjacent to tetrahedron t ($4n_t - b$ integers).

For both pseudo-manifold and manifold complexes, its space requirements are:

geometric information: $3n_v$ doubles
topological relations: $8n_t - b$ integers

If we assume in the manifold case that $n_t \approx 6n_v$, then the storage cost is approximately equal to $48n_v$ integers + $3n_v$ doubles.

Also, it can be easily seen that only boundary relations can be retrieved in optimal time from the indexed data structure with adjacencies. This data structure cannot support navigation through vertices and edges in optimal time.

7.3. Comparison

For a manifold or a reasonably regular object such that c_e and c_v are small, we can see that the space used by NMIA data structure is about *half* of that used by the simplified incidence graph.

The space requirements are about 1.5 times with respect to those of the indexed data structure with adjacencies, which, however, cannot support navigation through vertices and edges, since only the encoded relations plus boundary relations for tetrahedra can be retrieved efficiently.

8. Concluding Remarks

We have described the design and the implementation of a new data structure, that we called the *non-manifold indexed data structure with adjacencies (NMIA)*, which can represent any three-dimensional abstract simplicial complex in which any 2-simplex is on the boundary of at most two 3-simplexes. The NMIA structure is compact, since it encodes only the vertices and the top simplexes of the complex plus a restricted subset of topological relations among simplexes. It supports efficient navigation algorithms to retrieve topological relations, and it scales very well to the pseudo-manifold and manifold cases.

For comparison purposes, we have described a simplified version of the incidence graph for simplicial complexes, the *simplified incidence graph*, which, however, can only represent pseudo-manifolds. While it can be shown that this latter data structure also supports navigation in optimal time, its storage cost is almost twice that of the NMIA structure.

In order to accommodate non-manifold singularities and fast navigation, the storage cost of the NMIA structure is about 1.5 times that of the simple *indexed data structure with adjacencies*, which supports efficient retrieval only of the R_{3i} relations, with $i = 0, 1, 2, 3$.

In our future work, we plan to investigate extensions of the NMIA data structure to higher dimensions, to develop algorithms for updating the NMIA structure for simplification purposes. Our aim is to design and develop a multi-resolution volumetric model for non-manifold, and non-regular 3D objects, which extends the model proposed in [8] to the three dimensional case. In this context, we plan to use the NMIA data structure to represent the adaptive meshes extracted from the multi-resolution model through selective refinement queries.

9. Acknowledgments

This work has been partially supported by the project MACROGeo (funded by the Italian Ministry of Education, University and Research under contract number RBAU01MZJ5).

References

1. B. G. Baumgart. Winged edge polyhedron representation. Technical Report CS-TR-72-320, Stanford University, Department of Computer Science, October 1972.

2. E. Brisson. Representing geometric structures in *d* dimensions: Topology and order. In *Proceedings 5th ACM Symposium on Computational Geometry*, pages 218–227. ACM Press, June 1989.

3. E. Bruzzone, and L. De Floriani. Two data structures for building tetrahedralizations. *The Visual Computer*, 6(5): 266–283, 1990.

4. S. Campagna, L. Kobbelt, and H.-P. Seidel. Directed edges - a scalable representation for triangle meshes. *Journal of Graphics Tools*, 4(3), 1999.

5. W. Charlesworth and D. C. Anderson. Applications of non-manifold topology. In *Proceedings Computers in Engineering Conference and Engineering Database Symposium*, pages 103–112. ASME, 1995.

6. P. Cignoni, L. De Floriani, P. Magillo, E. Puppo, and R. Scopigno. Selective refinement queries for volume visualization of unstructured tetrahedral meshes *IEEE Transactions on Visualization and Computer Graphics*,to appear, 2003.

7. L. De Floriani, M. M. Mesmoudi, F. Morando, and E. Puppo. Non-manifold decomposition in arbitrary dimensions. In A. Braquelaire, J.-O. Lachaud, and A. Vialard, editors, *Discrete Geometry for Computer Imagery, Lecture Notes in Computer Science*, volume 2301, pages 69–80. Springer-Verlag, 2002. Extended version to appear in *Graphical Models*, 2003

8. L. De Floriani, P. Magillo, E. Puppo, and D. Sobrero. A multi-resolution topological representation for non-manifold meshes. In *Proceedings 7th ACM Symposium on Solid Modeling and Applications (SM02)*, 2002. Saarbrucken, Germany, June 17-21. Extended version to appear in *Computer-Aided Design*, 2003.

9. L. De Floriani, F. Morando and E. Puppo. Representation of Non-manifold Objects Through Decomposition into Nearly Manifold Parts. In *Proceedings 8th ACM Symposium on Solid Modeling and Applications (SM03)*, 2003. Seattle, USA, June 16-20, 2003.

10. H. Desaulnier and N. Stewart. An extension of manifold boundary representation to r-sets. *ACM Trans. on Graphics*, 11(1):40–60, 1992.

11. D. Dobkin and M. Laszlo. Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica*, 5(4):3–32, 1989.

12. H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

13. B. Falcidieno and O. Ratto. Two-manifold cell decomposition of r-sets. In A. Kilgour and L. Kjelldahl, editors, *Proceedings EUROGRAPHICS '92*, volume 11, pages 391–404, September 1992.

14. A. Gueziec, G. Taubin, F. Lazarus, and William Horn. Converting sets of polygons to manifold surfaces by cutting and stitching. In Scott Grisson, Janet McAndless, Omar Ahmad, Christopher Stapleton, Adele Newton, Celia Pearce, Ryan Ulyate, and Rick Parent, editors, *Conference abstracts and applications: SIGGRAPH 98, July 14–21, 1998, Orlando, FL*, Computer Graphics, pages 245–245, New York, NY 10036, USA, 1998. ACM Press.

15. L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams. *ACM Trans. on Graphics*, 4(2):74–123, April 1985.

16. E. L. Gursoz, Y. Choi, and F. B. Prinz. Vertex-based representation of non-manifold boundaries. In M. J. Wozny, J. U. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 107–130. Elsevier Science Publishers B.V., North Holland, 1990.

17. K. I. Joy, J. Legakis, R. MacCracken. Data Structures for Multiresolution Representation of Unstructured Meshes In G.Farin, H. Hagen, B. Hamann, editors, *Hierarchical Approximation and Geometric Methods for Scientific Visualization*, Springer Verlag, Heidelberg, 2002.

18. V. A. Kovalevsky. Finite topology as applied to image analysis. *Computer Vision, Graphics, and Image Processing*, 46(2):141–161, May 1989.

19. S.H. Lee and K. Lee. Partial Entity structure: a fast and compact non-manifold boundary representation based on partial topological entities. In *Proceedings of the Sixth ACM Symposium on Solid Modeling and Applications*, pp.159-170. ACM, June 2001. Ann Arbor, Michigan.

20. P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *Int. Journal of Comp. Geom. and Appl.*, 4(3):275–324, 1994.

21. H. Lopes and G. Tavares. Structural operators for modeling 3-manifolds. In *Proceedings of the Fourth ACM Symposium on Solid Modeling and Applications*, pages 10–18. ACM, May 1997. Atlanta, Georgia, May 14-16.

22. Y. Luo and G. Lukács. A boundary representation of form features and non-manifold solid objects. In *Proc. First ACM Symposium on Solid Modeling and Applications*, Austin, TX, June 1991.

23. M. Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, 1983.

24. S. McMains and C. S. J. Hellerstein, Out-of-core building of a topological data structure from a polygon soup In *Proceedings Sixth ACM Symposium on Solid Modeling and Applications*, Ann Arbor, Michigan, 2001, pages 171–182.

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

26. J. Rossignac and D. Cardoze. Matchmaker: Manifold BReps for non-manifold r-sets. In Willem F. Bronsvoort and David C. Anderson, editors, *Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications*, pages 31–41. ACM, June 1999.

27. J.R. Rossignac and M.A. O'Connor. SGC: A dimension-independent model for point sets 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 Science Publishers B.V. (North–Holland), Amsterdam, 1990.

28. J. Rossignac, A. Safonova, A. Szymczak. 3D compression Made Simple: Edgebreaker on a Corner Table. In *Proceedings Shape Modeling International 2001*, Genova, Italy, May 2001

29. K. Weiler. The Radial Edge data structure: A topological representation for non-manifold geometric boundary modeling. In J.L. Encarnacao, M.J. Wozny, H.W. McLaughlin, editors, *Geometric Modeling for CAD Applications*, pages 3–36. Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1988.

30. Y. Yamaguchi and F. Kimura. Non-manifold topology based on coupling entities. *IEEE Computer Graphics and Applications*, 15(1):42–50, January 1995.