ABSTRACT

Title of dissertation: REPRESENTING AND UNDERSTANDING NON-MANIFOLD OBJECTS

Annie Hui, Doctor of Philosophy, 2008

Dissertation directed by: Professor Leila De Floriani
Department of Computer Science

Solid Modeling is a well-established field. The significance of the contributions of this field is visible in the availability of abundant commercial and free modeling tools for the applications of CAD, animation, visualization etc.

There are various approaches to modeling shapes. A common problem to all of them however, is the handling of non-manifold shapes. Manifold shapes are shapes with the property of topological "smoothness" at the local neighbourhood of every point. Objects that contain one or more points that lack this smoothness are all considered non-manifold. Non-manifold objects form a huge catagory of shapes. In the field of solid modeling, solutions typically limit the application domain to manifold shapes. Where the occurrence of non-manifold shapes is inevitable, they are often processed at a high cost. The lack of understanding on the nature of non-manifold shapes is the main cause of it. There is a tremendous gap between the well-established mathematical theories in topology and the materialization of such knowledge in the discrete combinatorial domain of computer science and engineering. The motivation of this research is to bridge this gap between the two.

We present a characterization of non-manifoldness in 3D simplicial shapes. Based on this characterization, we propose data structures to address the applicational needs for the representation of 3D simplicial complexes with mixed dimensions and non-manifold connectivities, which is an area that is greatly lacking in the literature. The availability of a suitable data structure makes the structural analysis of non-manifold shapes feasible. We address the problem of non-manifold shape understanding through a structural analysis that is based on decomposition.

REPRESENTING AND UNDERSTANDING NON-MANIFOLD OBJECTS

by

Annie Hui

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2008

Advisory Committee:
Professor Leila De Floriani, Chair/Advisor
Professor David Mount
Professor Satyandra K. Gupta
Professor Amitabh Varshney
Professor Konstantina Trivisa

Dedication

To my parents

Acknowledgments

First and foremost, I would like to thank my advisor for being such a great teacher and scholar. She has been very supportive, encouraging and patient throughout all these years of my discipleship with her. I appreciate her friendship very much.

I would like to think my committee and my friends in graduate school for good advice, and for all the stimulating discussions. I am grateful for the exciting discussions with Dr. Jean-Claude Leon, Dr. Franca Giannini and Dr Laura Papaleo.

I want to thank Professor Diana O'leary for her sound advice and personal friendship. I want to express my appreciation to the support of many faithful friends in Grace Fellowship, Wallace and different parts of the world. I want to thank Neil for his encouragement. Lastly, I would like to thank my parents and grandmother for everything they have sacrificed.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AS	Automatic Segmentation (module)
be-SMART	BEyond Shape Modeling for understAnding Real world representations
CHF	Compact Half Face (data structure)
CoT	Corner Table (data structure)
CT	Cell Tuple (data structure)
DCEL / DC	Doubly-Connected Edge List
DE	Directed Edge
DLD	Double-Level Decomposition (data structure)
EIA	Extended Indexed (data structure) with Adjacencies
FE	Facet Edge (data structure)
GTA	Geometry and Topology Analyzer
HC	Handle Cell (data structure)
HE	Half Edge (data structure)
HF	Half Face (data structure)
IA	Indexed (data structure) with Adjacencies
IG	Incidence Graph
IQM	Initial Quasi-Manifold
IS	Incidence Simplicial (data structure)
L	Lath (data structure)
LE	Loop Edgeuse (data structure)
MC	Manifold Connected
MS	Manual Segmentation (module)
NMT	Non-manifold Multi Tesselation
PE	Partial Entities (data structure)
QE	Quad Edge (data structure)
RE	Radial Edge (data structure)
SA	Semantic Annotator
SIG	Simplified Incidence Graph
STEP	Standard for Exchange of Product Data, ISO 10303
SV	Star Vertex (data structure)
TD	Topological Decomposer
TS	Triangle Segment (data structure)
VF	Vertex Face (data structure)
VPC	Vertex Pair Contraction
WE	Winged Edge (data structure)

Chapter 1

Introduction

1.1 Modeling Shapes

The issue of data representation is at the core of solid modeling. Solid objects are usually described through cell and simplicial complexes. These complexes are collections of quasi-disjoint cells. The use of them offers the capabililty of decoupling between the geometry and the topology of the model. Typically, the topology at any region in the model is captured by the connectivity of the cells in that neighbourhood, while the geometry can be seen as an attributes of the cells in the complex.

There are variations on the amount of information described in an object model. At the lowest end, the model may consist of just a set of vertices and a set of polygons defined on these vertices. One example is the "soup of triangles". A soup-of-triangle data set consists of just a set of vertices which correspond to points with coordinates, and a set of triangles spanned by these vertices. The topological information captured in the data set is minimal. At the opposite end, the model may consist of a detailed description (including the topology, geometry and other attributes) of every cell in the complex.

In the rest of this work, we focus on representing simplicial complexes because they are the most common form of complexes and they play a key role in applications such as visualization, finite element analysis, geographic data processing, just to mention a few. Simplicial complexes are also a special case of cell complexes. Several topological properties that hold for cell complexes generally hold for simplicial complexes as well.

Traditionally, a shape has been described as a collection of surfaces bounding a volume of space. This approach gives rise to the so-called *boundary representation*. An alternative way is to perceive the shape as a volume of materials in the 3D space. This approach leads to the *volumetric representation*. Figures 1.1(a)-(d) gives examples of boundary and volumetric representations of a mechanical part.

Volumetric representations are used only when it is necessary to describe the "solidity" of a shape, There are various approaches to volumetric representation, such as CSG and OctTrees. We are interested in the object-based approach. The development of object-based volumetric representation is harder than that of boundary representation because the process is often not unique and involves the introduction of extra vertices; it is also difficult to visualize, or to perform shape modification on, a volumetric mesh. To date, limited number of techniques exist for working on volumetic meshes (see, for instance, Shewchuk's constrainted Delaunay tetrahedralization [74] implemented in the Tetgen tool.) The boundary representation is currently the most popular approach.

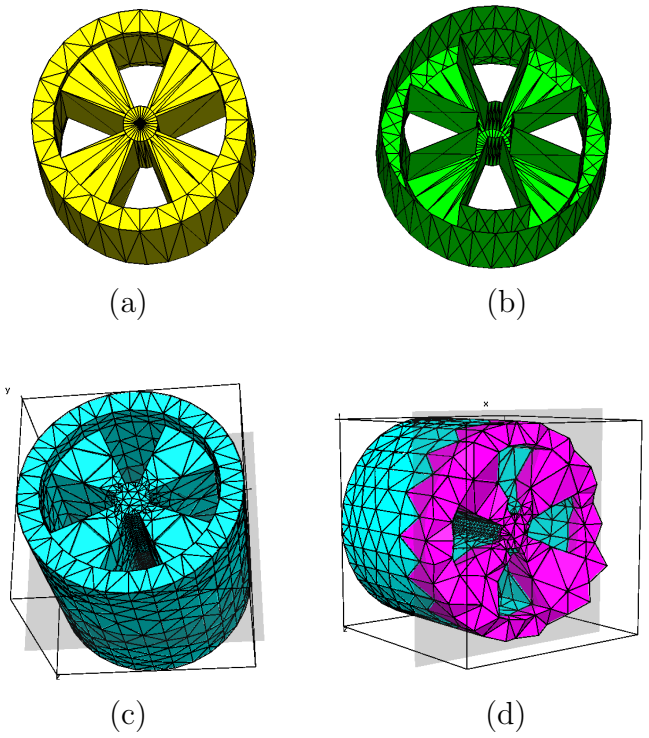


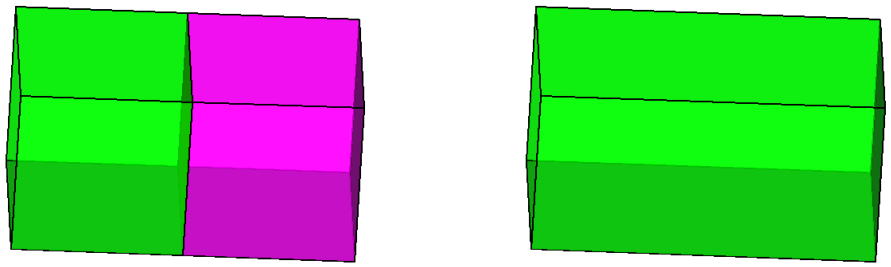
Figure 1.1: Boundary and volumetric representations of a mechanical part: (a) A mechanical part modeled by a tetrahedral mesh describing the surface of it; (b) A cross-section view of (a) showing the hollow internior of the model; (c) The same mechanical part modeled by a triangle mesh describing its volume; (d) A cross-section view of (c) highlighting the tetrahedra on the cut plane.

1.2 Pointers from the Literature

Thus, a huge amount of literature on topological data structures is based on boundary representation. Classical examples are the Winged-Edge data structure by Baumgart [4], the Half-Edge data structure by Mantyla [57], the Quad-Edge data structure by Guibas and Stolfi [41]. Data structures for manifold 2D cell and simplicial complexes exploit many nice topological properties in such complexes. A shape is (topological) smooth if the neighbourhood of every point in it is homeomorphic to a disk or half-disk. Objects with one or more point that does not fulfill this is called non-manifold. The handling of 2D manifold cell and simplicial complexes have been extensively investigated. In recent years, various data structures have been proposed for the optimization of storage costs, navigation efficiencies, capturing of additional properties and attributes, and ease of support for shape modification and compression.

A few volumetric representations have been proposed, which extend the data structures from modeling 2D manifolds to modeling the higher dimensional ones. An example is the Facet-Edge data structure [30] which is an extension of Quad-Edge data structure [41] to represent manifold complexes with volumetric cells. However, research in this area is still very limited to the manifold domain.

Application demands have driven solid modeling research in the non-manifold direction. One such demand comes from CAD applications, whereby an object consists of several connected parts. Consider the simple example of two cubes made of different materials sitting side-by-side. A boundary representation that captures



(a) (b)

Figure 1.2: (a) A model describing two cubes, made of different materials, sharing a face; (b) A model describing the overall shape of the object

the difference in parts is that of two cubes sharing one face as shown in Figure 1.2(a) and not that shown in Figure 1.2(b). Non-manifold connectivity exists in the model of Figure 1.2(a).

The needs to capture non-manifold properties in object boundary have been addressed by various researchers. The classical work on this is the Radial Edge data structure proposed by Weiler [79]. Subsequently, much effort has been channeled into improving the usability and cost effectiveness of data structures that represent shapes with 2D non-manifold boundaries. However, no work has been done on the handling of non-manifold parts in volumetric representations because of the intrinsic difficulties of the problem.

Moreover, some applications are not so interested in the precise geometric description of the surface of a shape. Instead, they are interested in the shape at some level of abstraction. One such non-trivial demand comes from the idealization prob-

lem in Reverse Engineering. An example is the modeling of supporting structures on a shape. Supporting structures such as wedges (shown in Figure 1.3(a)) are commonly found in mechanical parts and are appropriately abstracted by non-manifold connections (such as that shown in Figure 1.3(b)). The non-manifold connections between an object and its supporting structures form a framework that can be considered as a characteristic of a shape.

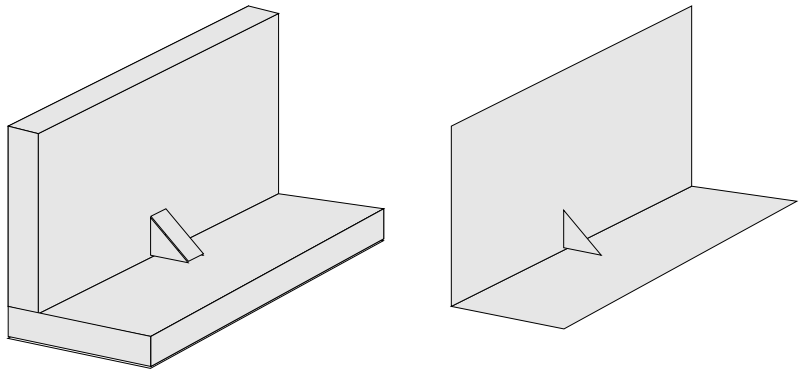


Figure 1.3: (a) A wedge that strengthens the connection between the vertical and the horizontal faces; (b) The idealized model of (a)

1.3 Our Contribution

The challenge of modeling objects with mixed dimensions and non-manifold connectivies is even greater when a model is to be represented as as mixture of volumetric parts, 2D parts and 1D parts, with non-manifold connectivites. We tackle this problem from the following aspects:

- the nature of the non-manifold parts in 3D shapes:

A characterization of them enables the design of data structures that properly represent non-manifold parts when they occur

- the costs of topological data structures that capture such non-manifold parts: The primary costs are: storage cost and navigation efficiency. These are general concerns shared to various degree by almost all applications. The secondary costs are:

  1. scalability to manifold inputs
  2. support for shape modification
  3. capturing of additional properties and attributes

  These secondary criteria are application-dependent. Some of these are hard constraints while others are irrelevant to specific applications. For example, in finite element analysis, it is necessary to assess the numerical error at each vertex, edge and face of a triangle mesh. So such an application requires a data structure that explicitly captures all such elements. An understanding of these criteria enables the optimization of costs on the topological data structures.

This work has made three contributions to solid modeling research. First, we performed a full characterization on non-manifold properties in 3D simplicial complexes. Non-manifoldness is an issue of connectivity. In order to facilitate a proper description of connectivity throughout the whole shape, we consider what non-manifold parts consist of and where they occur and show how such part can be captured.

Second, we applied our understanding of non-manifoldness to develop useful representations for 3D shapes with mixed dimensions and non-manifold connectivities. Based on the taxonomy we built on the literature, we evaluated the cost efficiencies of all data structures in each category, and identified the research frontier which is the focus of this work [15, 19, 20]. There are two motivations for the proposal of new data structures for non-manifold 3D shapes: First, only one data structure, namely the Incidence Graph, describes non-manifold simplicial meshes of dimension 3 or higher. No data structure exists that takes advantage of selected properties of simplicial complexes over cell complexes. Second, there are practical demands (1) from Finite Element Analysis applications for data structures that explicitly encode of all simplexes and (2) from Simulation applications for highly compact data structures.

We proposed, implemented and evaluated four new topological data structures for non-manifold simplicial 3-complexes. The *Non-Manifold Indexed data structure with Adjacencies (NMIA)* [13] is the first data structure to capture non-manifold singularities explicitly and succinctly. The NMIA has the additional feature of being highly compact and capable of supporting shape modification [14]. The *Simplified Incidence Graph (SIG)* [12, 17] and the *Incidence Simplicial (IS) data structure* [16], were developed to support the manipulation of non-manifold shapes, with the additional need for the explicit encoding of all vertices, edges, triangles and tetrahedra. The features of the SIG and the IS make such data structures suitable for the development of the multi-resolution model, *Non-manifold Multi-Tesselation*. The *Double-Level Decomposition (DLD) data structure* [47] was developed with the

same representational power as the NMIA, but along a completely different approach, namely that of decomposition.

Third, following the decomposition approach to shape modeling, we explored a new way to analysing non-manifold shapes. We proposed a decomposition-based representation that captures the high-level topology of a non-manifold shape [45, 18]. This step from shape representation to shape analysis has led to collaborations in research on shape understanding [21, 22, 66]. Part of this work on shape analysis has been built into a tool called *TopMesh*.

1.4 Overview

We give an overview to each subsequent chapter in this work. Chapter 2 reviews some mathematical backgrounds on topology in cell and simplcial complexes.

In Chapter 3, we characterize non-manifold properties, with special focus on 3D shapes. Non-manifoldness in a shape consists of parts whose neighbourhood is not "smooth". It occurs at points at which the neighbourhood is not homeomorphic to a d-dimensional ball. The neighbourhood of a cell is described combinatorially by the notion of "star". Non-manifoldness is characterized by the presence of mixed dimensions and complex connectivity in the star of a cell. These properties are discussed in details. We also demonstrate that an understanding of non-manifold properties is instrumental to several shape modeling applications. In particular, the elementary operation *vertex pair contraction*, in a shape modification, is fully described by a characterization of the change to the neighbourhoods of the vertices

to be merged.

In Chapter 4, we review the literature related to shape representation. We classify existing works primarily according to dimensions and domains, and secondarily according to the types of topological relations they encode. Dimension and domain are the first issues to consider when determining whether a data structure is feasible for an application. For a given dimension and a given domain, the next issue to consider is the types of topological information encoded within a data structure. Therefore, we classify the data structures into dimension-independent ones; ones specialized for 2D boundary models, and ones specialized for 3D volumetric models. For each dimension, we further consider whether the data structure is for manifold, regular or non-manifold domain. Then we examine each data structure according to what types of topological elements are encoded in them. Based on the types of elements they encode, we label the data structures as either implicit or explicit. The explicit type is further refined into: *incidence-based*, those which encode topological relations primarily among cells of different dimensions; *adjacency-based*, those which encode topological relations primarily among cells of the same dimension; and *edge-based*, which focused on relations of edge with respect to others. In addition to the traditional approach in which a shape is described as one whole piece. There is an alternative way of describing shapes in the literature, called decomposition. This approach is to break shapes down to simpler nearly manifold parts. We examine the representations built from this approach.

Chapter 5 address two proposals that are dimension-independent. They are called the *Simplified Incidence Graph (SIG)* and the *Incidence Simplicial (IS) data*

structure. These proposals address the demand for data structures that explicitly encode of all simplexes. The storage costs of each proposal is evaluated. Algorithms for the retrieval of topological relations are proposed for each of these data structures, with an evaluation of their efficiencies. As a test on the effectiveness of these proposals, the vertex-pair contraction operator is implemented on them. The Simplified Incidence Graph has used for the construction of the non-manifold multi-tesselation (NMT) application.

In Chapter 6, we present two data structures specialized for 3D simplicial complexes. These proposals address demand for high-optimized dimensional-specific representations. Optimization is important when an application needs to handle hundreds of millions of cells. Two proposals are made. They are the *Non-manifold Indexed data structure with Adjacencies (NMIA)* and the *Double-Level Decomposition (DLD) data structure*. Both the NMIA and the DLD data structures have very high compactness, very high scalability to manifold shapes, and support for high navigation efficiency. The NMIA offers efficient support for shape modification through a variation of vertex-pair contraction, while the DLD data structure is tailored for shape decomposition.

In Chapter 7, we explore the new frontier that is opened up through the decomposition of non-manifold 3D shape into uniformly dimensional parts connected by non-manifold joints. We proposed two levels of decomposition. At the low level, the decomposition is called *MC-decomposition*, and is based on the property of manifold-connectedness. The MC-decomposition breaks a shape into manifold-connected parts connected at non-manifold joints. This decomposition is used as

the basis for the high-level decomposition, namely the *semantics-oriented decomposition*. Components in the semantics-oriented decomposition are wire-webs, sheets and shells. This latter decomposition, along with other topological features, has been employed to construct a non-manifold shape analysis tool. The decomposition graph representing the structure of a non-manifold shape is the new research frontier for shape understanding. We report here our work collaborations, with the European Network of Excellence AIM@SHAPE, on Form Features Identification and on Shape Ontology.

In Chapter 8, we draw some concluding remarks and discuss future developments of this work and the open problems.

Chapter 2

Background Notions

In this Chapter, we review some notions on cell and simplicial complexes, that we will use throughout this dissertation (see [1] for more details).

2.1 Cell and Simplicial Complex

Intuitively, a Euclidean cell complex is a collection of basic elements, called *cells*, which cover a domain in the Euclidean space. A *k-dimensional cell* (or simply a *k-cell*) γ in the Euclidean space E^n, $1 \leq k \leq n$, is a subset of E^n homeomorphic to a closed k-dimensional ball $B^k = \{x \in R^k : ||x|| \leq 1\}$ ($||x||$ denotes the norm of vector x.) A 0-cell is a point in R^n. k is called the *order*, or *dimension*, of k-cell γ.

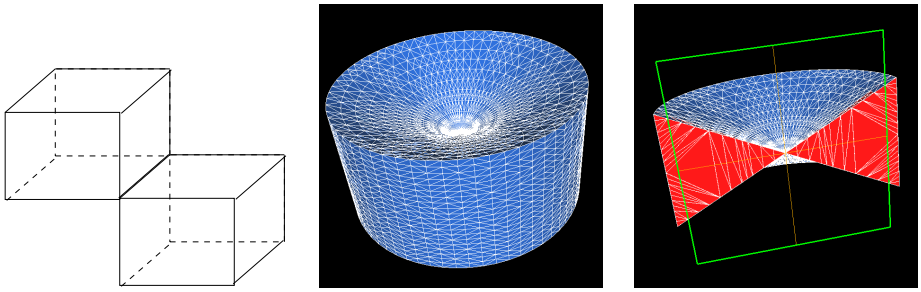
A *(Euclidean) cell complex* is a finite set Γ of cells of dimension at most d in E^n, $0 \leq d \leq n$, such that the interiors of the cells of Γ are disjoint, and if $\gamma, \gamma_1 \in \Gamma$, such that $\gamma \cap \gamma_1 \neq \emptyset$, then $\gamma \cap \gamma_1$ is the disjoint union of interiors of cells of Γ. A cell complex Γ, such that the maximum dimension of its cells is equal to d, is called a *d-dimensional complex*, or simply a *d-complex*. The *domain*, or *carrier*, of a Euclidean cell d-complex Γ embedded in E^n, with $0 \leq d \leq n$, is the subset of E^n

defined by the union, as point sets, of all the cells in Γ.

The *(combinatorial) boundary* of a cell γ in a cell complex Γ is the set of all cells in Γ, which are subsets of the boundary of cell γ (considered as a point set). Every cell in $b(\gamma)$ is called a *face* of γ. If k is the dimension of the face, it is called a *k-face*. The *co-boundary*, or *star*, of a cell γ, is the collection of all the cells in Γ containing γ in its boundary. The *link* of a cell γ is defined as the collection of the cells bounding the cells in the star of γ, which do not contain γ. A cell is called a *top cell* if it is not contained in the boundary of any other cell in Γ.

Two cells are called *k-adjacent* if they share a k-face. Two p-cells, $0 < p \leq d$, 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. An *h-path*, $0 \leq h \leq d-1$, is a sequence of $(h+1)$-cells $(\gamma_i)_{i=0}^{k}$ such that two consecutive cells γ_{i-1} and γ_i in the sequence are h-adjacent. Two cells γ and γ^* are said to be *h-connected* if there exists an h-path $(\gamma_i)_{i=0}^{k}$ such that γ is a face of γ_0 and γ^* is a face of σ_k. A complex Γ^* is called *h-connected* if and only if any two cells of Γ^* are h-connected.

A d-complex Γ, in which all top cells are d-cells, is called *regular* (or *uniformly d-dimensional*). A regular $(d-1)$-connected d-complex in which each $(d-1)$-cell is shared by one or two d-cell is called a *(combinatorial) pseudo-manifold* (possibly with boundary). A pseudo-manifold complex whose domain is a manifold is called a *manifold complex*. Figure 2.1(a) shows an example of a regular complex, which is not a pseudo-manifold, while Figure 2.1(b) and (c) show an example of a pseudo-manifold complex which does not have a manifold domain.



(a) (b) (c)

Figure 2.1: (a) A regular cell complex that is not manifold; (b) A pseudo-manifold with a non-manifold domain (a 3D pinched pie); (c) The cross-section of the pinched pie at the non-manifold vertex.

Note that sometimes in solid modeling, there is the need to describe objects with faces of disjoint boundaries. The decomposition of the boundary in such case is not a cell complex, but it can be transformed into a cell 2-complex by adding suitable dummy edges [57].

2.2 Simplexes and Simplicial Complex

implicial complexes can be seen as a subclass of cell complexes. Their cells, called *simplexes*, are defined as the convex combination of points in the Euclidean space. A Euclidean *simplex* σ of dimension k is the convex hull of $k+1$ linearly independent points in the n-dimensional Euclidean space E^n, $0 \leq k \leq n$. We simply call a *Euclidean simplex* of dimension k a *k-simplex*. k is called the *dimension* of σ. Any Euclidean p-simplex σ', with $0 \leq p < k$, generated by a set $V_{\sigma'} \subseteq V_\sigma$ of cardinality $p+1 \leq d$, is called a *p-face* of σ. Whenever no ambiguity arises, the dimension of σ' will be omitted, and σ' is simply called a *face* of σ. Any face σ' of σ such that $\sigma' \neq \sigma$ is called a *proper face* of σ.

A finite collection Σ of Euclidean simplexes forms a *Euclidean simplicial complex* if and only if (i), for each simplex $\sigma \in \Sigma$, all faces of σ belong to Σ, and (ii), for each pair of simplexes σ and σ', either $\sigma \cap \sigma' = \emptyset$ or $\sigma \cap \sigma'$ is a face of both σ and σ'. If d is the maximum of the dimensions of the simplexes in Σ, we call Σ a *d-dimensional simplicial complex*, or a *simplicial d-complex*. The domain (or carrier) of a Euclidean simplicial complex is defined in the same way as for a cell complex. Since a simplicial complex is a cell complex, all the properties of cell complexes are inherited by simplicial complexes.

2.3 Topological Relations

The connectivity information among the entities in a cell or in a simplicial complex are expressed through *topological relations*. These latter provide an effective framework for defining, analyzing and comparing the wide spectrum of existing data structures. Data structures for cell and simplicial complexes can be described formally in terms of the topological entities and relations they encode. We define topological relations for the case of a cell complex (since a simplicial complex can be seen as a special case of a cell complex).

We consider a cell d-complex Γ and a cell $\gamma \in \Gamma$, with $0 \leq p \leq d$. We can define *topological relations* as follows:

- *Boundary relation* $R_{p,q}(\gamma)$, with $0 \leq q \leq p-1$, consists of the set of q-cells which are faces of γ.

- *Co-boundary relation* $R_{p,q}(\gamma)$, with $p+1 \leq q \leq d$, consists of the set of q-cells

incident in γ.

- For $p>0$, *adjacency relation* $R_{p,p}(\gamma)$ consists of the set of p-cells in Γ that are $(p-1)$-adjacent to γ.

- Relation $R_{0,0}(\gamma)$, where γ is a vertex, consists of the set of vertices that are adjacent to γ through a 1-cell (an edge).

Figure 2.2 illustrates topological relations: $R_{2,1}(f)$ is the set of edges bounding 2-cell f (see Figure 2.2(a)), relation $R_{0,1}(v)$ is the set of edges incident in vertex v (see Figure 2.2(b)), relation $R_{2,2}(f)$ consists of the set of 2-cells which share one edge (1-cell) with 2-cell f (see Figure 2.2(c)). Note that both boundary and co-boundary relations are called *incidence relations*.

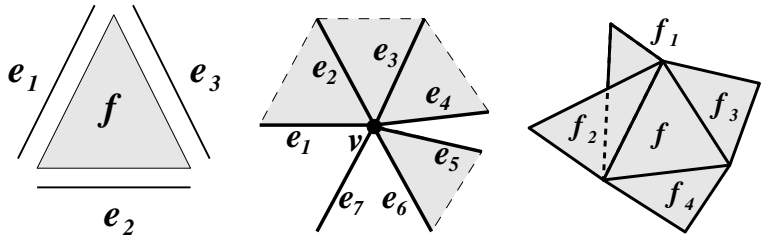


Figure 2.2: Example of topological relations (a) boundary relation $R_{2,1}(f) = \{e_1, e_2, e_3\}$ for face f, (b) co-boundary relation $R_{0,1}(v) = \{e_1, \cdots, e_7\}$ for vertex v, and (c) adjacency relation $R_{2,2}(f) = \{f_1, \cdots, f_4\}$ for face f

We call *constant* any relation which involves a constant number of entities. Relations which involve a variable number of entities are called *variable*. Co-boundary and adjacency relations are variable relations in general. Boundary relations are constant in simplicial complexes. Thus, we consider an algorithm for retrieving a

topological relation R to be *optimal* if it retrieves a given relation R in time linear in the number of entities involved in R. Depending on the amount of information encoded, data structures for cell and simplicial complexes may support the retrieval of topological relations according to various degree of efficiency. If the retrieval of a relation requires examining the star of all the cells adjacent to or on the boundary of the query cell, we say that the data structure offers a *sub-optimal* support for the retrieval of that relation. If the retrieval a relation requires examining all cells of a specific dimension, then the data structure does not support an efficient retrieval of that relation.

Chapter 3

Understanding and Handling Non-manifold 3D Shapes

In this Chapter, we present a mathematical characterization of non-manifold properties in simplicial complexes, specifically for the 3D case, and discuss a generic approach to capturing non-manifold properties. Note that these properties are characterizable likewise on cell complexes.

Non-manifold situations are frequently encountered by shape modification applications. One pair of elementary operator in shape modification is vertex pair contraction and vertex split, upon which high-level shape modification operations are built. Through an understanding of the properties of non-manifolds, we discuss how they can be handled in such an operator.

3.1 Mathematical Characterization of Non-manifold Singularities

We characterize the non-manifold singularities in the combinatorial representation of a non-manifold shape by defining non-manifold cells in its discretization

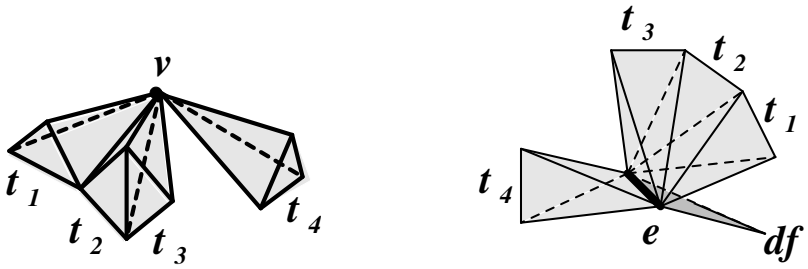
as a cell complex.

A vertex (0-cell) v in a cell (simplicial) d-complex Γ (with $d \geq 1$) is a *manifold* vertex if and only if the link of v in Γ is homeomorphic to a triangulation of the $(d-1)$-sphere S^{d-1}, or of the $(d-1)$-disk B^{d-1}. A vertex is called *non-manifold* otherwise (see the example in Figure 3.1(a).)

An edge (1-cell) e in a d-complex Γ (with $d \geq 2$) is a *manifold edge* if and only if the link of e in Γ is homeomorphic to a triangulation of the $(d-2)$-sphere S^{d-2}, or of the $(d-2)$-disk B^{d-2}. An edge is called *a non-manifold edge* otherwise (see the example in Figure 3.1(b).)

In general, a k-cell γ is a d-complex Γ (with $d \geq k+1$) is a *manifold* k-cell if and only if the link of γ in Γ is homeomorphic to a triangulation of the $(d-k)$-sphere S^{d-k} or of the $(d-2)$-disk B^{d-k}. It is called *non-manifold* otherwise.

In a manifold d-complex, all the top cells are of dimension d. A non-manifold complex may consist of top cells of mixed dimensions. In the case of 3D simplicial complexes with mixed dimensions, we call 1-dimensional top simplexes *wire-edges*, while we call the 2-dimensional top simplexes *dangling-faces*. A 2D simplicial complex is not uniformly dimensional if it contains wire-edges.



(a) (b)

Figure 3.1: (a) A non-manifold vertex v; (b) A non-manifold edge e

3.2 Understanding Non-manifold Properties in 3D Combinatorial Shapes

Traditionally, shapes are modeled as collections of uniformly-dimensional manifold shapes. Any non-manifold parts that arise in shape processing are considered as singularities. The current research is on how to handle such singularities effectively. Assuming that a shape is mostly manifold, it is sufficient to capture the subsets of the shape at which non-manifold properties exist. In the case of a 3D simplicial complex embedded in 3D Euclidean space, such subsets pertain to:

1. the lower-dimensional parts composed of wire-edges and dangling-faces;

2. the neighborhoods of the non-manifold edges; and

3. the neighborhoods of the non-manifold vertices.

In a 3D simplicial 3-complex embedded in 3D space, there are no non-manifold faces because each triangle is shared by at most two tetrahedra. The neighborhood of non-manifold simplexes is described by their stars and their links. The star of a

non-manifold edge e consists at least two disjoint groups of tetrahedra, or dangling-faces. Wire edges are not present in the star of e. All simplexes in the star of e can be radially ordered around the edge. Likewise, the link of e consists of chains of edges and isolated vertices and can be radially ordered on a plane. Each chain of edges in the link of e corresponds to a fan of tetrahedra in the star of e, while each isolated vertex corresponds to a dangling-face. Figure 3.2(a) illustrates the star of a non-manifold edge, which consists of fans of tetrahedra and dangling-faces. Figure 3.2(b) shows the link of the non-manifold edge shown in Figure 3.2(a).

The star of a non-manifold vertex v is more complex. It may consist of either connected or disjoint groups of tetrahedra, dangling-faces and wire-edges. The link of v is a general 2-dimensional simplicial complex which may be geometrically projected onto a sphere in the 3D space. Each k-simplex (for $k=0,1,2$) in the link of v corresponds to a $(k+1)$-simplex in the star of v. Every non-manifold vertex in the link of v corresponds to a non-manifold edge incident at vertex v. Figure 3.3(a) gives an example of a non-manifold vertex whose neighborhood consists of seven disjoint components, of which four are wire-edges, one consists of just dangling-faces, one consists of just tetrahedra, and one has mixed dimensions. Figure 3.3(b) shows the link of the non-manifold vertex shown in Figure 3.3(a). It is the complexity of the connectivity at non-manifold vertices which makes the case of non-manifold modeling challenging. Since the link of v is a 2-complex for which Euler's Formula $V - E + F = 2$ holds (where V, E and F are respectively the numbers of vertices, edges and faces in the link of v), the relationship $E_s - F_s + T_s = 2$ holds for the elements in the star of a vertex, where E_s, F_s and T_s is are respectively the number

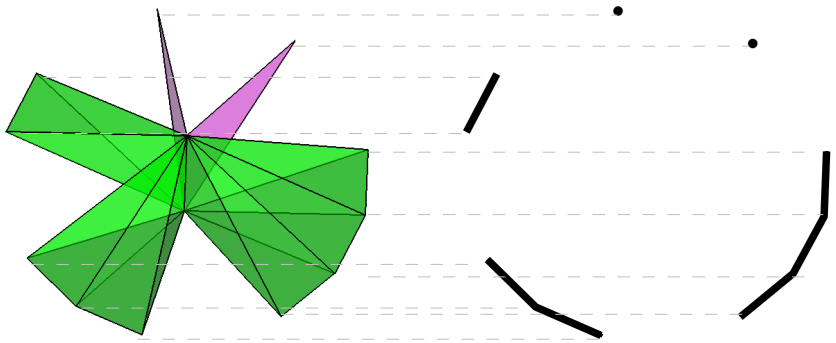


Figure 3.2: (a) Example of simplicial 3-complexes showing the star of a non-manifold edge. The tetrahedra are in green while the dangling-faces are in purple. The link of the non-manifold edge is shown in (b).

of edges, faces and tetrahedra in the star of v.

The non-manifold cases in simplicial 2-complexes embedded in 3D Euclidean space can be considered as a subset of those in the 3-complexes because of the absence of tetrahedra. The lower-dimensional parts are composed of wire-edges only. The star of a non-manifold edge e in a 2-complex consists of more than two triangles, and thus the link of e is a set of isolated vertices. Figure 3.2(a) shows an example of a non-manifold edge in a simplicial 2-complex. Its link is shown in Figure 3.2(b). The star of a non-manifold vertex v in a simplicial 2-complex consists of triangles and wire-edges. Each component in the star of v is either 1-dimensional (that is a wire-edge), or 2-dimensional (that is a 1-connected set of triangles). The link of v is a 1-complex, in which an isolated vertex corresponds to a wire-edge in the star of v while a 0-connected set of edges corresponds to a 1-connected set of

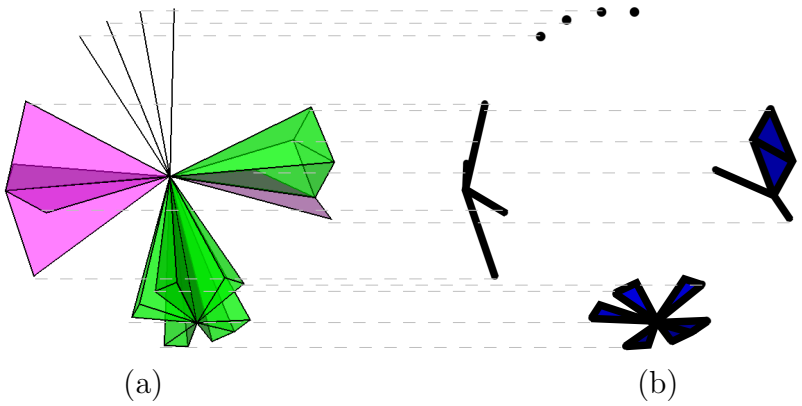


Figure 3.3: (a) Example of simplicial 3-complexes showing the star of a non-manifold vertex. The tetrahedra are in green while the dangling-faces are in purple. The link of the vertex is shown in (b).

triangles in the star of v. An example of a non-manifold vertex v is shown in Figure 3.3(a) and its link is shown in Figure 3.3(b). An interesting point to note is that the non-manifold singularities for simplicial 3-complexes embedded in 3D space are the same as those for the 2D complexes in 3D space. It means that the problem of representing the former is no harder than that of the latter.

Table 3.1 summarizes the non-manifold properties, in simplicial complexes of dimensions 2 and 3, in terms of the types of non-manifold simplexes that may be present in the complexes, and the locations at which non-manifold connectivities may occur. Table 3.2 and Table 3.3 summarizes the detailed characteristics of the non-manifold singularities in simplicial 3- and 2-complexes discussed in this Subsection.

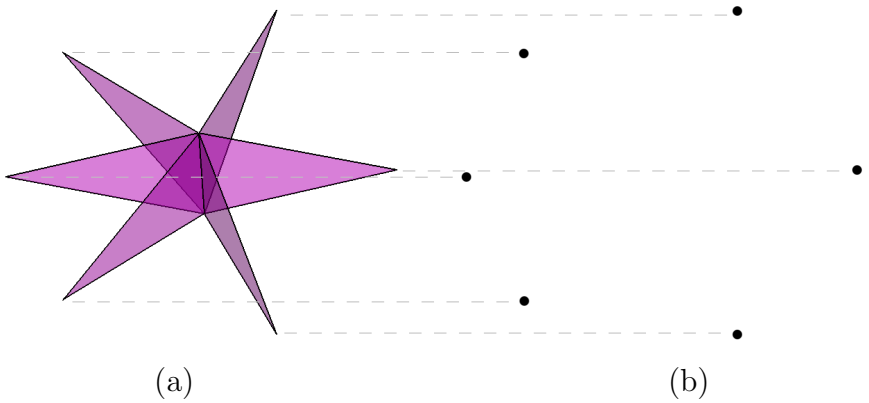


Figure 3.4: (a) Example of simplicial 2-complexes showing the star of a non-manifold edge. The link of the non-manifold edge is shown in (b).

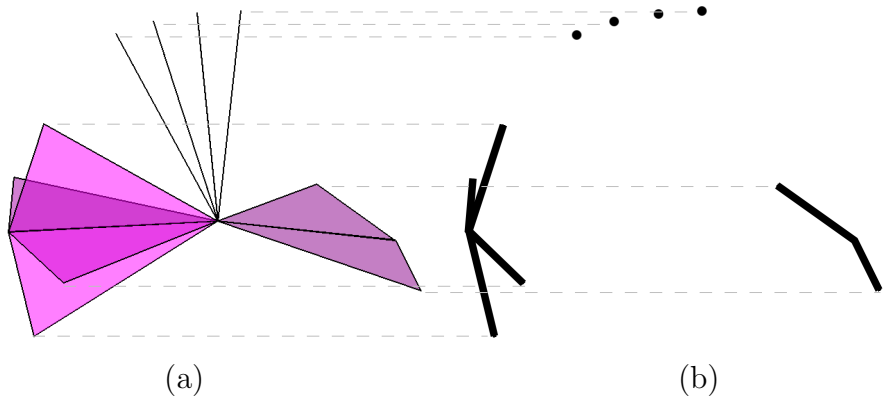


Figure 3.5: (a) Example of simplicial 2-complexes showing the star of a non-manifold vertex. The link of the non-manifold vertex is shown in (b).

Shape	Simplicial 3-complex	Simplicial 2-complex
Lower dimensional parts	Dangling faces and wire-edges	Wire edges
Non-manifold connectivity	At edges and vertices	

Table 3.1: Summary of non-manifold characteristics in simplicial 3- and 2-complexes embedded in E^3

Properties	In simplicial 3-complex	In simplicial 2-complex
Composition of star $st(e)$	At least 2 fans of tets, or Mixed tets and dangling-faces, or At least 3 dangling-faces	At least 3 triangles
Orderability of star $st(e)$	Linearly sortable around e	
Composition of link $lk(e)$	Chains of edges and Isolated vertices	Isolated vertices
Projectibility of link $lk(e)$	Geometrically projectable onto a circle on a plane	

Table 3.2: Summary of the properties of the neighborhood of a non-manifold edge e in simplicial 3- and 2-complexes embedded in E^3

3.3 Handling Non-manifold Singularities in the Updates of Simplicial Complexes

A question related to the representation of a non-manifold shape is on how a local modification on the shape will affect the topology in correspondence of the modification. Shape modification is one major cause to the occurrence of non-manifold singularities in a shape. While not at the center of our research, this question is of primary interest to applications which modify a shape in the modeling process, such as [52].

In this Section, we consider one elementary update operation, namely *vertex-pair contraction (VPC)*, and a special case of it, known as *edge collapse*, performed on a 3D complexes. In the vertex-pair contraction operation, two vertices v_1 and

Properties	In simplicial 3-complex	In simplicial 2-complex
Composition of star $st(v)$	Mixed tets, dangling-faces and wire-edges	Mixed dangling-faces and wire-edges
Orderability of star $st(v)$	Not sortable	
Composition of link $st(v)$	Sets of connected triangles and edges, and isolated vertices	Sets of connected edges and isolated vertices
Projectibility of link $lk(v)$	Onto a sphere in 3D space	
Connectivity at link $lk(v)$	Any non-manifold vertices in $lk(v)$ correspond to non-manifold edges in $st(v)$	

Table 3.3: Summary of the properties of the neighborhood of a non-manifold vertex v in simplicial 3- and 2-complexes embedded in E^3

v_2 in the existing complex are merged into a new vertex in the complex. Variations on vertex-pair contraction include the *assymetric VPC* in which vertex v_2 is merged into v_1 without creating a new vertex. This latter operation is often selected since it does not create new vertices. *Edge-collapse* which requires that the existing vertices v_1 and v_2 share an edge e. This requirement ensures that the operation does not introduce new handles into the shape. Edge-collapse on 3D complex is of special interest to some applications because it allows topological modification to be made in a controlled fashion. An application built using VPC is the progressive simplicial complexes.

3.3.1 Effect of Vertex-Pair Contraction on a Simplicial Complex

We formally define here the effect of vertex-pair contraction on a d-dimensional simplicial complex. Then, we consider the case when this operation is performed

on 3D complexes. To this aim, we introduce some notations. For a vertex w, we denote as $lk(w)$ its link and with $st(w)$ the set of simplexes in its star. Let v_1 and v_2 be two vertices in a simplicial d-complex Σ. A vertex-pair contraction applied to pair (v_1, v_2) consists of contracting vertices v_1 and v_2 to a new vertex v. Thus, all simplexes that are in $st(v_1)$ or in $st(v_2)$ become incident at v. This can be described through a map F which maps simplexes in $st(v_1) \cup st(v_2)$ onto $st(v)$, in such a way that for each simplex $\sigma \in st(v_1) \cup st(v_2)$, $F(\sigma) = \sigma - \{v_1, v_2\} \cup \{v\}$. Note that if a p-simplex also belongs to $st(v_1) \cap st(v_2)$, map F transforms σ into a $(p-1)$-simplex. Figure 3.6 shows the effect of a vertex-pair contraction. Map F is a surjective function. Its effect can be characterized by *four* possible cases. Let us consider a p-simplex σ $(p \leq d)$ in the star of the new vertex v:

Case 1: There exists *one* p-simplex σ_1 in the star of v_1 such that $\sigma = F(\sigma_1)$, and σ_1 has an empty intersection with every simplex in the star of v_2 (see Figure 3.7). In this case, σ is obtained from σ_1 just by replacing v_1 with v.

Case 2: There exists *one* p-simplex σ_2 in the star of v_2 which has an empty intersection with every simplex in the star of v_1, such that $\sigma = F(\sigma_2)$. This case is completely symmetric with respect to case 1.

Case 3: There exist *two* p-simplexes σ_1 and σ_2, belonging to the star of v_1 and of v_2, but not to the intersection of the two stars, such that $\sigma = F(\sigma_1)$ and $\sigma = F(\sigma_2)$ (see Figure 3.8).

Case 4: There exist *three* simplexes, a $(p+1)$-simplex σ' belonging to the intersection

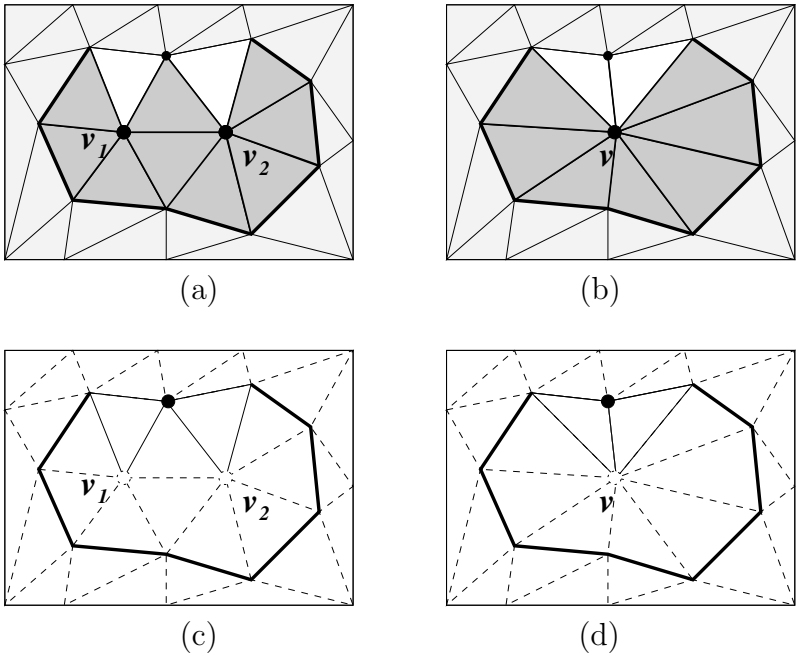


Figure 3.6: In a vertex-pair contraction, vertices v_1 and v_2 become one new vertex v: (a) shows $st(v_1) \cup st(v_2)$ in dark gray; (b) shows $st(v)$ in dark gray which replaces $st(v_1) \cup st(v_2)$ in the complex; (c) shows $lk(v_1) \cup lk(v_2)$ in thick black lines and a vertex. It remains the same as $lk(v)$ which is shown in (d).

of the stars of v_1 and v_2 and two p-simplexes σ_1, and σ_2, belonging to the stars of v_1 and v_2, respectively, such that $\sigma = F(\sigma')$, $\sigma = F(\sigma_1)$ and $\sigma = F(\sigma_2)$. In this case, σ results from contracting σ' incident at edge $e = \{v_1, v_2\}$, and from transforming σ_1 and σ_2 into σ through map F (see Figure 3.9).

Consider the vertex-pair contraction on a simplicial complex of dimension 3. Figures 3.10(a)-(j) summarizes all the cases that occur in such a complex.

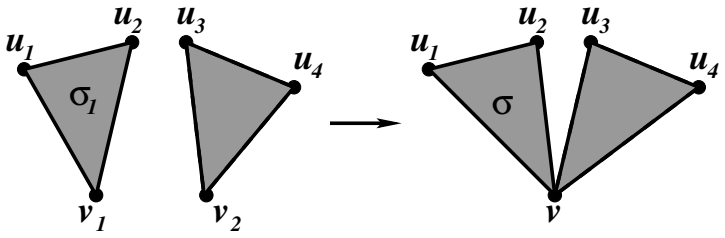


Figure 3.7: Case 1: simplex σ is obtained from exactly one simplex, σ_1, because σ_1 does not intersect with $st(v_2)$.

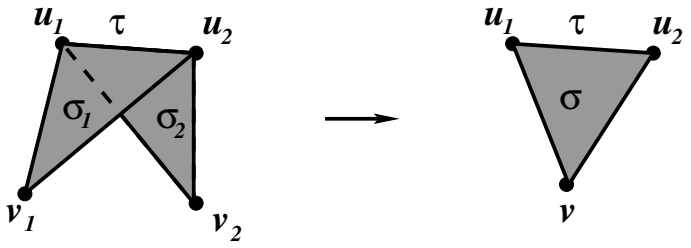


Figure 3.8: Case 3: simplex σ is obtained from two simplexes, σ_1 and σ_2 because σ_1 and σ_2 intersect at τ. $\sigma = F(\sigma_1) = F(\sigma_2) = \tau - \{v_1, v_2\} \cup \{v\}$.

3.3.2 Vertex Expansion on Simplicial Complex

In shape modeling applications, often there is a need to reverse the action performed by a simplification operation. For example, in the application of multi-resolution modeling, a model is first reduced to its coarsest level (which is called

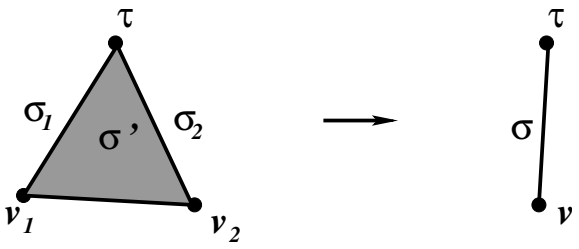


Figure 3.9: Case 4: simplex σ is obtained from three simplexes, σ', σ_1 and σ_2, the intersection of all three of which is at τ. $\sigma = F(\sigma_1) = F(\sigma_2) = \tau - \{v_1, v_2\} \cup \{v\}$.

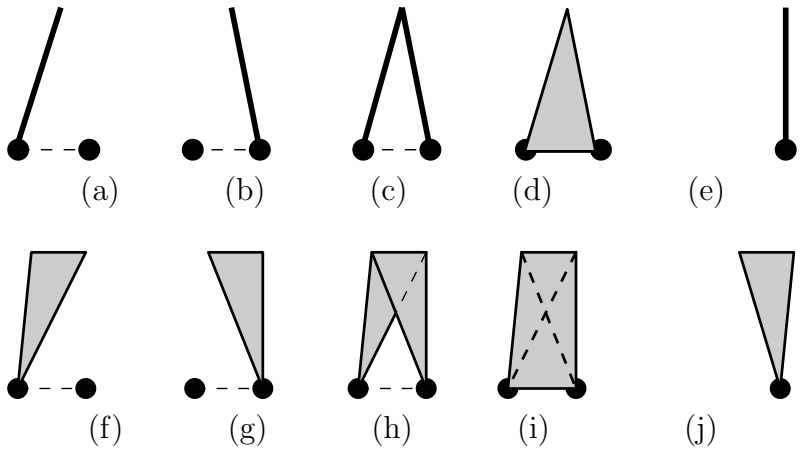


Figure 3.10: All the cases that occur in the vertex-pair contraction operation on a simplicial 3-complex: (a)-(d) Cases 1 to 4 for $p = 1$; (e) Mapping of (a)-(d) by F; (f)-(i) Cases 1 to 4 for $p = 2$; (j) Mapping of (f)-(i) by F; the dotted lines connecting the vertices that are to be merged denote the possible presence of an edge between the two vertices.

the base mesh) through a series of simplification operations. Then regions of the model is reconstructed on demand to a higher level of details. In such applications, there is not only the need to perform simplification, but also the need to reverse its effect. The reverse operation of a vertex-pair contraction, *vertex expansion*, can be performed by reversing the actions taken in the vertex-pair contraction algorithm and it is briefly described here. Vertex expansion is defined as follows: vertex v is expanded into two new vertices v_1 and v_2, that may or may not be connected by an edge (see Figure 3.11). For each p-simplex σ in the star of v, σ may become:

- a new p-simplex incident only at v_1 (in the example of the triangle in Figure 3.12(a), Figure 3.12(b) shows the result of this case of the expansion of v), or
- a new p-simplex incident only at v_2 (see Figure 3.12(c)), or

- two new p-simplexes, one each at v_1 and v_2 (see Figure 3.12(d)), or
- a new $(p+1)$-simplex incident at v_1 and v_2 (see Figure 3.12(e))

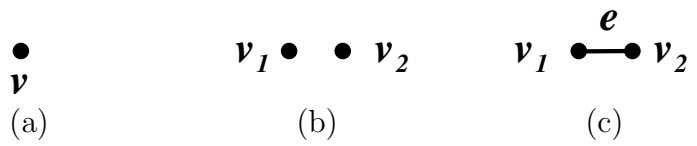


Figure 3.11: Vertex expansion: expansion of v may result in two independent vertices or two vertices connected by an edge

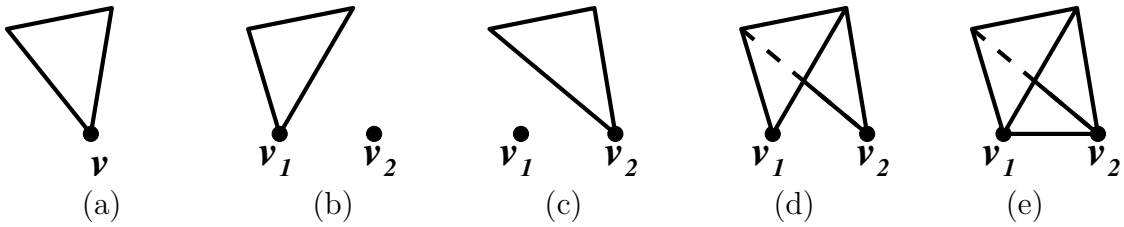


Figure 3.12: Vertex expansion: (a) Before expansion, a triangle is incident at vertex v; (b)-(d): After expansion; (b) case 1: the new triangle is incident at v_1; (c) case 2: the new triangle is incident at v_2; (d) case 3: two new triangles are created, one each at v_1 and v_2; (e) case 4: a tetrahedron is created incident at v_1 and v_2

3.3.3 Effect of Edge Collapse on a Simplicial 3-Complex[1]

In this Section, we study edge collapse, a constrained version of vertex-pair contraction, on a simplicial 3-complex. This is the most commonly used update

[1]Originally published in [14]. Reproduced with notice of ACM copyright: permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

operation on tetrahedral meshes, and is the basis for techniques like [9, 39, 75]. Conditions for performing edge collapse on a manifold shape without changing its topology has been studied by [29]. We are interested in the topological changes caused by edge collpse. We analyze edge collapse in terms of how the connectivity of top simplexes change in the neighbhourhood of the collapsing edge. Understanding this enables the edge collapse operation to be performed on highly compact data structures for 3D simplicial complexes.

The reverse operation of edge collapse is *vertex split*. Vertex split is a variant of vertex expansion (see Section 3.3.2), with the constraint that the vertex v is expanded into two vertices v_1 and v_2 sharing an edge e.

Let $e = (v_1, v_2)$ be the edge to be collapsed in a 3D simplicial complex Σ. Let Σ' be the complex resulting from Σ by collapsing edge e into a vertex v.

An *edge collapse* applied to an edge $e = (v_1, v_2)$ in Σ consists of replacing edge e with the new vertex v in Σ', collapsing all dangling-faces and tetrahedra in the restricted star of e, $st(e)$, to wire-edges and faces incident at v, respectively, and in updating all the top simplexes in $st(v_1) \cup st(v_2)$, i.e., all the top simplexes incident at either v_1 or v_2. We call $st(v_1) \cup st(v_2)$ the neighbourhood of the collapsing edge e.

Given a top k-simplex, σ, $k = 1, 2, 3$, in Σ, such that $\sigma \in st(e) \cup st(v_1) \cup st(v_2)$, either σ is incident at edge e, or σ is incident at v_1 or v_2 but not at both. In the former case, σ must be either a dangling-face or a tetrahedron. In the latter case, there are two possible situations. Let us assume that σ is incident at v_1, then either σ intersects some other top simplex $\overline{\sigma}$ incident at the other vertex, v_2, or σ has an

empty intersection with all other top simplexes incident at v_2.

Therefore, the following three cases may occur:

1. $\sigma \in st(e)$: in this case, σ can either be a top 2-simplex or a 3-simplex. Within this case, we further examine the neighbourhood of σ within the scope of $st(v_1) \cup st(v_2) - st(e)$. There are three sub-cases:

   (a) for every top simplex σ_1 in $st(v_1) - st(v_2)$, $\sigma \cap \sigma_1 = \{v_1\}$ and for every top simplex σ_2 in $st(v_2) - st(v_1)$, $\sigma \cap \sigma_2 = \{v_2\}$; that is σ connects to simplexes in $st(v_1) \cup st(v_2) - st(e)$ only through either v_1 or v_2;

   (b) there exists a top simplex σ_1 in $st(v_1) - st(v_2)$ that shares a q-face τ with σ (where $q > 0$), but for every top simplex σ_2 in $st(v_2) - st(v_1)$, $\sigma \cap \sigma_2 = \{v_2\}$; that is σ is more than 0-connected to some simplexes in $st(v_1) - st(e)$, but only 0-connected to simplexes in $st(v_2) - st(e)$. This also include the symmetric case where v_1 and v_2 are reversed.

   (c) there exists a top simplex $\sigma_1 \in st(v_1) - st(v_2)$ that shares a q-face τ_1 with σ, and a top simplex $\sigma_2 \in st(v_2) - st(v_1)$ that shares a r-face τ_2 with σ, such that σ_1 shares a m-face with σ_2, $(m < q, r)$.

   Figures 3.13(a)-(f) give six examples of simplexes that are incident at edge $e = (v_1, v_2)$ to be collapsed as in case 1. In Figures 3.13(a) to 3.13(c), the top simplex in $st(e)$ is tetrahedron t_1. In Figures 3.13(d) to 3.13(f), the top simplex in $st(e)$ is dangling-face df_1.

2. $\sigma \in st(v_1)$, $\sigma \notin st(v_2)$ and there exists $\overline{\sigma} \in st(v_2)$ such that $\sigma \cap \overline{\sigma} = \tau$ and

$\tau \neq \emptyset$: in this case, σ is incident at v_1 but not in v_2, and there exists a top simplex $\overline{\sigma}$ incident at v_2 which intersects σ. Therefore, there exists a pair of non-top simplexes $(\gamma, \overline{\gamma})$ such that $\gamma = \{\tau, v_1\}$ is a face of σ and $\overline{\gamma} = \{\tau, v_2\}$ is a face of $\overline{\sigma}$. This case also includes the symmetric situation in which $\sigma \in st(v_2)$ and $\sigma \notin st(v_1)$ and there exists $\overline{\sigma} \in st(v_1)$ such that $\sigma \cap \overline{\sigma} \neq \emptyset$.

In Figures 3.14(a)-(f), six examples are given of top simplexes that are incident at only one extreme vertex of the edge to be collapsed, and that intersect some top simplexes that are incident at the other extreme vertex. As in Figure 3.13, e is the edge to be collapsed. In Figures 3.14(a) to 3.14(c), the non-empty intersection of those top simplexes incident at v_1 and those incident at v_2 is edge (u_1, u_2). In the other three examples in Figures 3.14(d) to (f), the non-empty intersection is vertex u.

3. $\sigma \in st(v_1)$, $\sigma \notin st(v_2)$ and $\sigma \cap \overline{\sigma} = \emptyset$, for every $\overline{\sigma} \in st(v_2)$: in this case, σ is incident at v_1 and not in v_2 and it does not have any intersection with simplexes incident at v_2. This case also includes the symmetric situation in which $\sigma \in st(v_2)$ and $\sigma \notin st(v_1)$ and $\forall \overline{\sigma} \in st(v_1)$, $\sigma \cap \overline{\sigma} = \emptyset$. Figure 3.15 shows an example where a top simplex incident at v_1 does not intersect any simplex incident at v_2.

Recall from Section 3.3.1 the map $F(\sigma) = \sigma - \{v_1, v_2\} \cup \{v\}$ which define the merge of two vertices. In edge collapse, map F is applied with the constraint that

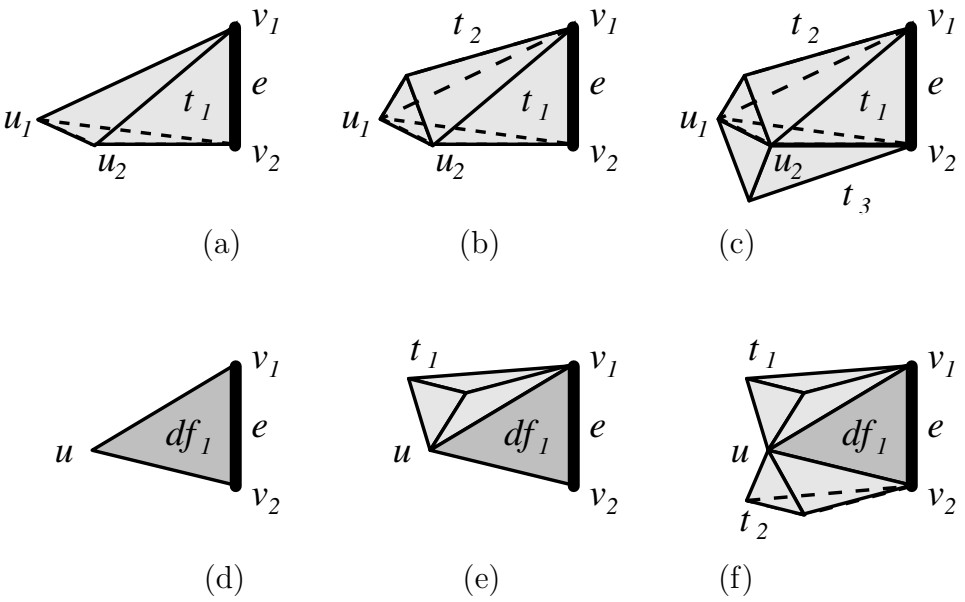


Figure 3.13: Six examples of top simplexes that are incident at the edge $e = (v_1, v_2)$ to be collapsed: in (a), (b) and (c), the top simplex incident at e is tetrahedron t_1. In (d), (e) and (f), the dangling-face df_1 is incident at e. (Case 1)

v_1 and v_2 share a common edge e. The effect of edge collapse on the connectivity of the top simplexes at the neighbourhood of the collapsing edge is fully characterized as follows:

- in case 1: map F reduces the top p-simplex σ in Σ to a $(p-1)$-simplex σ' in the reduced complex Σ'. For each of the sub-cases within this case, the result of applying F is as follows:
  - for sub-case 1: σ' is a top $(p-1)$-simplex;
  - for sub-case 2: Consider the q-face τ shared between σ and σ_1 in Σ. If $q = p-1$, then σ' in the reduced complex is a face of $\sigma'_1 = F(\sigma_1)$. Otherwise, σ' is a top $(p-1)$-simplex;
  - for sub-case 3: Consider the q-face τ_1 shared between σ and σ_1 and the

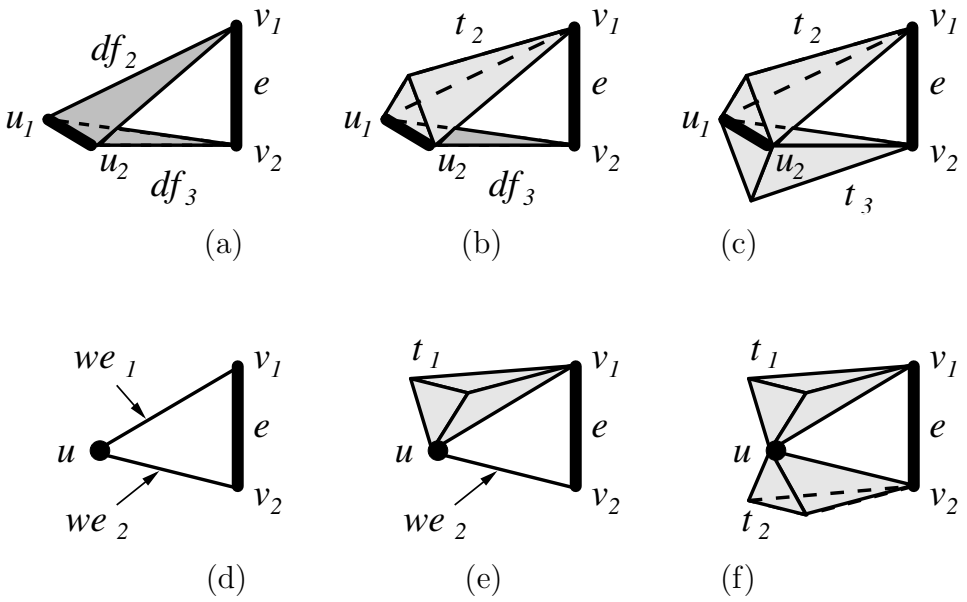


Figure 3.14: Six examples of top simplexes that are incident at one vertex of the edge e to be collapsed, and that have non-empty intersection with some other top simplexes that are incident at the other vertex. In (a), dangling-face df_2 is incident at v_1 and dangling-face df_3 is incident at v_2. Their intersection is edge (u_1, u_2). (b) and (c) are similar to (a), except that the simplex incident at either vertex may also be a tetrahedron. In (d), wire-edges we_1 and we_2 are incident at v_1 and v_2, respectively, and their intersection is vertex u. Similarly, in (e) and (f), the intersection is at vertex u. (Case 2)

r-face τ_2 shared between σ and σ_2 in Σ. If $q = p - 1$, then σ' in the reduced complex is a face of $\sigma'_1 = F(\sigma_1)$. Similarly, if $r = p - 1$, then σ' is a face of $\sigma'_2 = F(\sigma_2)$. If $q, r < p - 1$, then σ is a top $(p-1)$-simplex. The simplexes σ'_1 and σ'_2 are r-connected in Σ'.

Note that if two tetrahedra t_1 and t_1 in $st(e)$ are face-adjacent in Σ, then F transforms t_1 and t_2 into two edge-adjacent faces in Σ'.

- in case 2: map F transforms σ and $\overline{\sigma}$ into new simplexes σ' and $\overline{\sigma'}$ sharing the same vertex v; F also transform the p-faces γ and $\overline{\gamma}$ into a new p-face γ'

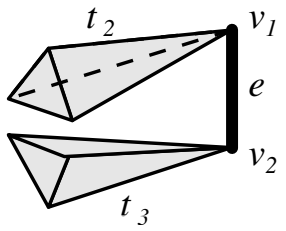


Figure 3.15: An example in which a top simplex, that is incident at one vertex of the edge e to be collapsed, has an empty intersection with all other top simplexes that are incident at the other vertex. Tetrahedra t_2 and t_3 are incident at v_1 and v_2, respectively, and they do not intersect each other. (Case 3)

common to σ' and $\overline{\sigma'}$

- in case 3: map F transforms σ into a new simplex σ' incident at v

Figures 3.16(a)-(f) and Figures 3.17(a)-(f) show the effect of edge collapse on each of the examples in discussed in Figures 3.13, 3.14. In each example, the drawing on top is a reproduction of that in Figures 3.13 and 3.14. The drawing at the bottom shows what happens after edge e is collapsed. In Figures 3.16(a) to 3.16(c), tetrahedron t_1 is incident at e. After edge collapse, t_1 may be become a dangling-face, as in Figure 3.16(a), a boundary face, as in Figure 3.16(b), or an internal face, as in Figure 3.16(c). In Figures 3.16(d) to 3.16(f), df_1 is incident at e. After edge collapse, df_1 may become a wire-edge, as in Figure 3.16(d), or a boundary edge, as in Figures 3.16(e) and 3.16(f). Figure 3.17 is similar. The result of edge collapse on the example of Figure 3.15 is shown in Figure 3.18.

3.4 Summary

In this Chapter we have characterized the non-manifold properties pertaining to non-manifold simplicial shapes in the Euclidean 3D space. An understanding on such characteristics essential to the design of efficient topological data structures for 3D shapes. For the application of such data structures, we have also formulated an elementary mesh modification operator, namely vertex-pair contraction, and characterized edge collapse (a constrained form of vertex-pair contraction) when it is applied to simplicial 3D shapes. The materials in this Chapter serve as the foundation for data structure designs and shape modification applications.

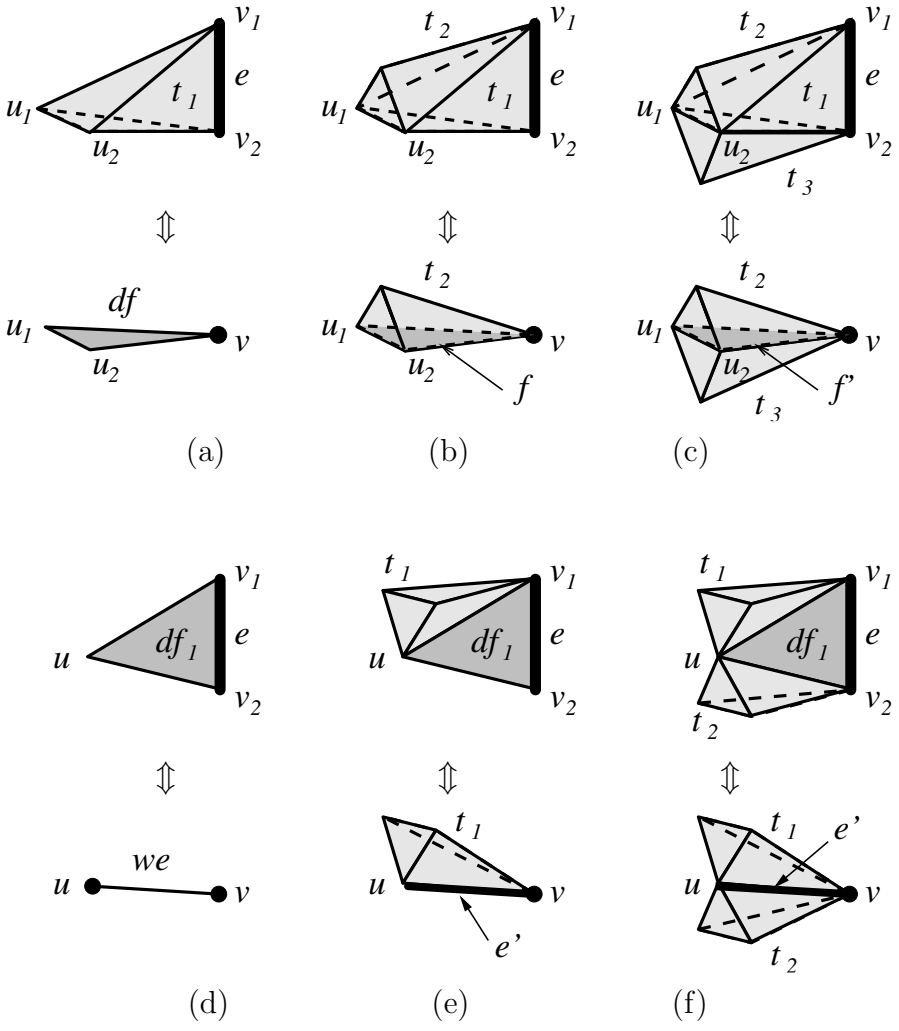


Figure 3.16: Effect of edge collapse on the six examples shown in Figure 3.13. Edge $e = (v_1, v_2)$ is collapsed into vertex v. In (a), (b) and (c), tetrahedron t_1 becomes a new dangling-face, df, an external face, f, of tetrahedron t_2, or an internal face, f', shared by tetrahedra t_2 and t_3, respectively. In (d), dangling-face df_1 becomes a wire-edge, we. In (e) and (f), df_1 becomes a boundary edge, e', of simplexes incident at both u and the new vertex v.

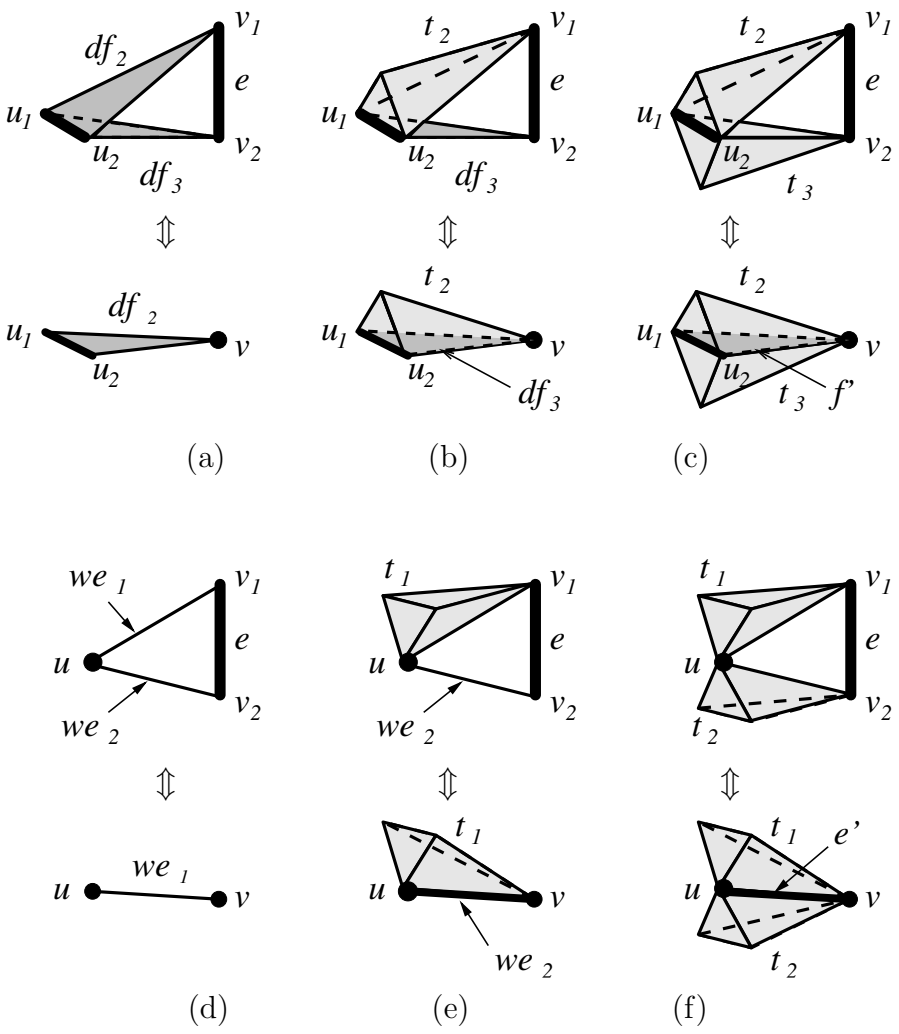


Figure 3.17: Effect of edge collapse on the six examples shown in Figure 3.14. Edge $e = (v_1, v_2)$ is collapsed into a new vertex v. In (a), dangling-faces df_2 and df_3, which originally intersect at edge (u_1, u_2), become one face df_2. In (b), df_3 becomes an external face of tetrahedron t_2. In (c), the two tetrahedra t_2 and t_3 become face-adjacent at f'. In (d), wire-edges we_1 and we_2 become a single wire-edge, we_1. In (e), we_2 becomes a boundary edge of simplexes incident at both u and the new vertex v. In (f), the two set of simplexes that are incident at v_1 and v_2, respectively, become edge-adjacent at edge e' after edge collapse.

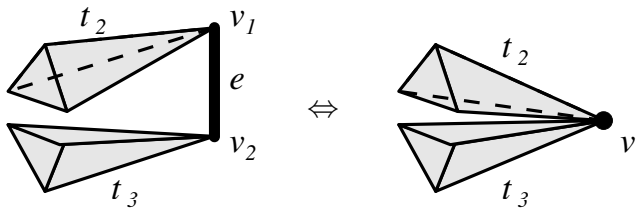


Figure 3.18: Effect of edge collapse on the example shown in Figure 3.15. When edge e is collapsed, the two tetrahedra become 0-adjacent.

Chapter 4

State of the Art[1]

4.1 Criteria of classification and evaluation of topological data structures

We can first classify the data structures for cell and simplicial complexes in terms of:

1. *the domain* of the complexes they represent: manifold, pseudo-manifold, regular, etc.

2. *the dimension*: *dimension-independent* data structures can describe cell and simplicial complexes in any dimension, while *dimension-specific* data structures are for 2D and 3D cell and simplicial complexes embedded in the three-dimensional Euclidean space.

3. *the topological information encoded*: in a cell (or simplicial) complex, the basic topological elements are the cells (simplexes). A data structure may encode all the cells of a complex, or only a subset of it.

[1]Originally published in [19] Copyright ©2007 Eurgraphics.

4. *the way topological information is encoded*: some data structures encode the cells and their topological relations *explicitly*. In such data structures, the cells are *entities* and the relations are associated with the entities. *Implicit* data structures encode the relations among cells indirectly, through tuples of cells in the same relation.

Explicit data structures can be further classified into *incidence-based*, and *adjacency-based* representations. Incidence-based data structures encode all cells in a complex and a suitable subset of their incidence relations. Adjacency-based data structures generally encode only top cells (i.e., cells which are not on the boundary of other cells) and vertices, and adjacency relations among them plus possibly a suitable subset of co-boundary relations. We distinguish a further category for data structures for simplicial and cell 2-complexes, which consists of *edge-based* data structures in the 2D case.

Data structures that are designed for cell complexes can be used for simplicial complexes. In some cases, specializations of such data structures have been developed by taking advantage of the properties of simplicial complexes.

In the remainder of this Chapter, we organize the description of the various representations on the basis of the dimension of the complex they represent. We present a description of each data structure in terms of the entities and topological relations it encodes, and we evaluate it based on its expressive power, on its space requirements, and on the efficiency in supporting navigation inside the complex (i.e., in retrieving topological relations not explicitly encoded). In explicit data structures,

topological queries are based on cells and simplexes. In contrast, in implicit data structures, navigation is typically performed with the tuples as the atomic units. The efficiency of navigation is measured in terms the number of such atomic units retrieved which correspond to the cells or simplexes in the relations queried.

The space requirements are expressed throughout this Chapter in terms only of number of items of topological information encoded, since we assume that all the data structures encode the same geometrical information. This also gives an evaluation which is independent of the specific implementation, which is not always described in the literature in sufficient details. We compare the various data structures inside each category based on the above features and, for representations for non-manifold shapes, also based on their *scalability* to the manifold case. One important issue in evaluating a data structure for non-manifold shapes is its *scalability* to the manifold case, which is evaluated as the overhead of the storage cost of data structure when applied to a manifold shape with respect to that of a data structure of the same type but specifically designed for manifold shapes. This is relevant since in a typical modeling scenario we need to have a representation capable to deal with non-manifold shapes, but most of the shapes will be in any case manifold. We emphasize data structures for simplicial complexes since these are the most common mesh-based models in a variety of applications.

4.1.1 Dimension-independent Data Structures

In this Section, we discuss dimension-independent representations for cell complexes first, and then for simplicial complexes. We review first two dimension-independent implicit representations, namely the *Cell-Tuple* [6] and the *N-G-map* [53] representations, and an explicit incidence-based representation, the *Incidence Graph (IG)* [32]. The two implicit representations are for manifold shapes, while the latter is for non-manifold shapes as well. We then discuss data structures specific for simplicial complexes, namely the *Indexed data structure with Adjacencies (IA)*[65] (which is a d-dimensional extension of the representation discussed in [64]). The IA data structure is an adjacency-based representation and is for pseudo-manifolds.

4.1.1.1 Cell-Tuple and N-G-map

A *Cell-Tuple* [6] is a representation for Euclidean cell complexes with a manifold domain, while the *n-G-map* [53] has been developed for abstract cell complexes belonging to the class of quasi-manifolds, which is a superclass of combinatorial manifolds defined in [53]. In essence, however, the cell-tuples and the n-G-maps are equivalent. Here, we describe, for brevity, only the Cell-Tuple data structure.

Given a Euclidean d-dimensional cell complex, a *cell tuple* is a $(d+1)$-tuple t of $d+1$ cells, $t = (c_0, c_1, \cdots c_d)$, such that c_i is an i-cell on the boundary of cells c_{i+1} to c_d. A function s_i for $i = 0..d$, called a *switch function*, is defined on the cell-tuples such that $t' = s_i(t)$ if the cell tuple t' is identical to t in every element except the i-th one. The s_i functions partition the set of cell-tuples into equivalent classes of

size 2 each. The s_i functions have the following two properties:

- for $i = 0, ..., d$, s_i is an involution, that is, given a cell tuple t, $s_i(s_i(t)) = t$;
- for $i = 0, ..., d-2$ and $i+2 \leq j \leq d$, $s_i s_j$, where $s_i s_j(t) = s_j(s_i(t))$, is an involution, that is, $s_i s_j(s_i s_j(t)) = t$.

Figure 4.1(a) gives a simple example of a cell complex defined on a surface without boundary. The cell complex is composed of two internal 2-cells A and B, and the 2-cell C. Figure 4.1(b) shows all the tuples in small squares, and all the s_i $(i = 0, 1, 2)$ functions. Two tuples are related by function s_0 if they are connected through a dotted line, by s_1 if connected by a thin solid line, or by s_2 if connected by a dashed line.

The Cell-Tuple data structure encodes all cell-tuples in a complex, and the switch functions s_i for $i = 0..d$. It is an implicit data structure because the cells and their mutual topological relations are implicitly represented by the cell-tuples.

The topological relations encoded by the cell tuple data structure can be formalized as follows:

- boundary $R_{p,q}$ for each p-cell, $p > 0$, for each $0 \leq q < p$.
- co-boundary $R_{p,q}$ for each p-cell, $p < d$, for each $0 \leq q < p$.

The space requirements of the Cell-Tuple data structure can be evaluated for a simplicial d-complex with n_d d-simplexes as follows [26]. The number of tuples is at most $n_d(d+1)!$. The switch functions s_i are encoded as (s_i, t, t') where $s_i(t) = t'$,

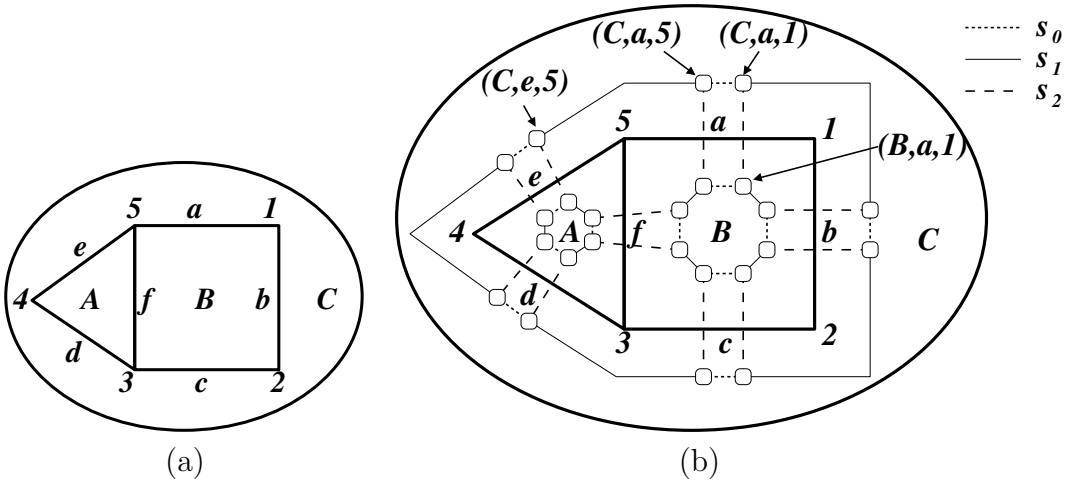


Figure 4.1: (a) A simple cell complex on a surface homeomorphic to a sphere. The complex is composed of triangle A, square B and the external 2-cell C on the surface; (b) all the tuples and all the switch s_i ($i = 0, 1, 2$) functions encoded by the cell-tuple

which consists of $n_d(d+1)(d+1)!$ pieces of information. To efficiently support topological navigation, it is necessary to store links from each p-simplex σ to each of the cell-tuples that contain of σ. This needs $n_d(d+1)!$ extra links and thus it results in a verbose representation.

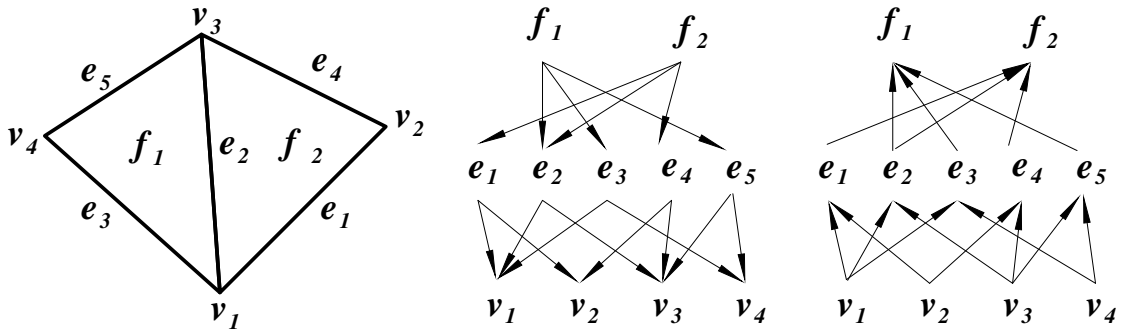
It can be shown that all topological relations can be retrieved in optimal time from the Cell-Tuple data structure. As an example, consider the retrieval of relation $R_{0,2}(5)$ for vertex 5 in Figure 4.1, which consists of all the 2-cells that are incident at vertex 5. The retrieval starts with any of the tuples that include vertex 5, such as $(C, a, 5)$. By alternately applying functions s_2 and s_1 to each new tuple visited, the cyclic sequence $(C, a, 5)$, $(B, a, 5)$, $(B, f, 5)$, $(A, f, 5)$, $(A, e, 5)$, $(C, e, 5)$ is obtained, which produces the set of 2-cells $\{C, B, C\}$ that are incident at vertex 5.

4.1.1.2 Incidence Graph (IG)

The *Incidence Graph (IG)* [32] is an incidence-based explicit data structure for cell complexes. The incidence relations among cells that differ by one dimension are explicitly encoded. Formally, the IG encodes all the cells of any given cell d-complex Γ, and for each p-cell γ, its immediate boundary, and immediate co-boundary relations, namely:

- for each p-cell γ, where $0 < p \leq d$, boundary relations $R_{p,p-1}(\gamma)$,
- for each p-cell γ, where $0 \leq p < d$, co-boundary relations $R_{p,p+1}(\gamma)$

Figures 4.2(a)-(c) give an example that illustrates the relations encoded in the IG in the form of a directed graph.



(a) (b) (c)

Figure 4.2: (a) A simple simplicial complex formed by two triangles; (b) all the boundary relations encoded by the IG; (c) all the co-boundary relations encoded by the IG

The design of the IG supports a simple recursive strategy to retrieve topological boundary and co-boundary relations. Boundary relation $R_{p,q}(\gamma)$ $(p > q)$ for a given

p-cell γ is obtained by retrieving the encoded boundary $R_{i,i-1}$ relations of all the i-faces for $i = p, \cdots, q-1$ of γ. Co-boundary relation $R_{p,r}(\gamma)$ $(p < r)$ is obtained by retrieving the encoded co-boundary $R_{i,i+1}$ relations of all the i-cells for $i = p, \cdots, r-1$ in the star of γ. The retrieval of such relations can be done in time linear in the number of cells involved, which is thus optimal.

We can evaluate the exact space requirements of the IG when it encodes a simplicial complex because each simplex has a constant number of faces. The boundary and co-boundary relations encoded by the IG for a simplicial complex amounts to

$$2 \sum_{0<p\leq d} n_p(p+1)$$

pieces of information, because each p-simplex has exactly $(p+1)$ faces of dimension $(p-1)$.

The space requirements for an Incidence Graph can be evaluated as follows. Let n_q denote the number of cells of dimension q, with $0 \leq q \leq d$ in a cell complex Γ. The storage cost of the IG is bounded by

$$\sum_{0<p\leq d} n_p n_{p-1},$$

where $n_p n_{p-1}$ is the bound on the number of boundary relations in the complex $R_{p,p-1}$. The co-boundary relations encoded in the IG are symmetric to the boundary relations.

In the case of manifold cell 2-complexes, every edge is shared by two faces and is incident at two vertices. For every link from an edge to either its vertex or its face, there is a reverse link from that entity back to the edge. Thus, we can deduce the storage cost of the IG for manifold 2-complexes will be $8n_1$.

4.1.1.3 Indexed Data Structure with Adjacencies

The *Indexed data structure* is a representation for simplicial d-complexes which encodes, for each top k-simplex σ, relation $R_{k,0}(\sigma)$, i.e., the indexes to its $(k+1)$ vertices. Only boundary relations of type $R_{k,j}(\sigma)$, $j < k$, for any top k-simplex σ, can be extracted in optimal time from such representation. Note that the j-simplexes on the boundary of σ are described through $j+1$ vertex indexes.

The *Indexed data structure with Adjacencies (IA)*, also called *winged representation* [64, 65], extends the indexed data structure into a topological data structure, which also encodes adjacency information among the simplexes. This restricts its representation domain to pseudo-manifolds. The IA data structure encodes, for each d-simplex σ in a simplicial complex Σ:

- relation $R_{d,0}(\sigma)$, i.e., the indexes of its $(d+1)$ vertices;
- relation $R_{d,d}(\sigma)$, i.e., the indexes of the $(d+1)$ d-simplexes sharing a $(d-1)$-face with σ.

Only boundary relations, as in the indexed data structure, plus relation $R_{d,d}$ can be retrieved in optimal time from the IA data structure. Vertex-based coboundary relations can retrieved in optimal time from an extension of the IA data structure, which we call the Extended Indexed data structure with Adjacencies (EIA). This is achieved by encoding, for each vertex v, a partial version of relation $R_{0,d}(v)$, that we denote $R^*_{0,d}(v)$, i.e., one d-simplex for each connected component of the link of v.

Figure 4.3 gives an example of the retrieval of the complete $R_{0,3}(v)$ relation

for vertex v. We start from the encoded partial $R^*_{0,3}(v) = \{t_1, t_4\}$ relation. From v, t_1 is accessible through $R^*_{0,3}(v)$. t_2 is accessible through the $R_{3,3}(t_1)$ relation of t_1. Similarly, the tetrahedra t_3 and t_5 in the star of v are retrievable first by extracting t_4 from $R^*_{0,3}(v)$ and then by retrieving $R_{3,3}(t_4)$.

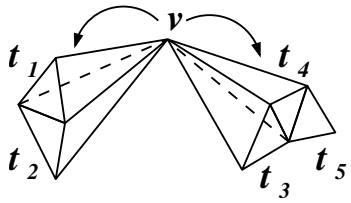


Figure 4.3: Example of the retrieval of the $R_{0,3}(v)$ relation from the encoded partial $R^*_{0,3}(v) = \{t_1, t_4\}$ relation in the EIA data structure

This extension allows extracting all simplexes in the star of a vertex in time linear in the number of such simplexes, i.e., all $R_{0,k}(v)$ relations, where $0 \leq k \leq d$, can be retrieved in time linear in the number of d-simplexes in the star of v. The retrieval of $R_{0,k}(v)$ for $d \leq 3$ is still optimal, while this is not true if $d > 3$ since there is no linear relation among the number of k-simplexes incident in vertex v and the number of vertices in $R_{0,0}(v)$. Retrieval of all $R_{q,k}(\sigma)$ relations for $0 < q < k < d-1$ requires traversing the star of each of the vertices of σ and thus it takes time linear in the number of d-simplexes incident at the vertices of σ. Thus, such algorithms are still local, but sub-optimal.

The storage cost of the EIA data structure is equal to $2n_d(d+1)+n_0$ items for a simplicial complex with n_d d-simplexes and n_0 vertices, which is only n_0 items more with respect to storing just the $R_{d,0}$ and $R_{d,d}$ relations. For a manifold simplicial

2-complex, this leads to $6n_2 + n_0$, which is approximately $13n_0$ as a consequence of Euler's formula.

4.1.1.4 Comparisons

We compare the dimension-independent data structures described above in terms of their expressive power, of their characteristics, their storage costs and their efficiency in supporting topological navigation.

Table 4.1 summarizes the comparison among the various dimension-independent data structures in terms of their domain, of the complexes they can describe, and of the representation method.

Data Structure	Domain	Complexes	Method
Cell-Tuple	Manifold	Cell	Implicit
IG	Non-manifold	Cell	Incidence-based
EIA	Manifold	Simplicial	Adjacency-based

Table 4.1: Data structures for d-dimensional complexes

We summarize the space requirements of the above data structures, for manifold and for arbitrary simplicial complexes. These costs are expressed throughout this Chapter only in terms of items of topological information encoded. The storage costs of the cell-tuple, of the IG and of the EIA data structure for a manifold d-dimensional simplicial complex are as follows (they are expressed in terms of the number of q-simplexes, denoted as n_q):

- Cell-Tuple: $n_d(d+1)(d+1)! + n_d(d+1)!$
- EIA: $2(d+1)n_d + n_0$
- IG: $2\sum_{0<p\leq d} n_p(p+1)$

For arbitrary d-dimensional simplicial complexes, we can only report the storage costs of the IG, since the other two are for restricted classes of complexes. In the following, k_q denotes the total number of connected components at all the links of the q-simplexes of the complex.

Table 4.2 summarizes the navigation costs by evaluating the optimality of algorithms for retrieving topological relations on the various representations.

Data Structure	Boundary relations	Co-boundary relations	Adjacency relations
Cell-Tuple	Optimal	Optimal	Optimal
IG	Optimal	Optimal	Optimal
EIA	Optimal	$R_{0,d}$: optimal Others: sub-optimal	$R_{d,d}$: optimal Others: sub-optimal

Table 4.2: Navigation efficiency of data structures for d-dimensional complexes

4.1.2 Data Structures for 2D Complexes

In this Section, we discuss representations for cell and simplicial 2-complexes embedded in the 3D Euclidean space. We classify them according to the taxonomy introduced in Section 4.1, and organize their description in two subsections accord-

ing to the domain of the complexes (manifold or non-manifold). We perform the comparison based on their space requirements and on their efficiency in retrieving topological relations. We evaluate and compare the non-manifold representations also based on their scalability to the manifold case.

4.1.2.1 Representations for Manifold 2-Complexes

We discuss here representations for cell and simplicial complexes for manifold shapes. A thorough analysis and comparison of data structures for manifold cell 2-complexes can be found in [73]. Here, we briefly review the *Winged-Edge* [4], the *Doubly-Connected Edge List (DCEL)* [61], the *Half-Edge* [57] the *Quad-Edge* [41], the *Lath-based* [48] data structures, and the Star-Vertex [49] data structure for manifold cell complexes, and the Corner Table [71] data structures for manifold simplicial complexes (usually called *triangle meshes*). The Winged-Edge, DCEL, and the Half-Edge data structures are all edge-based representations, since they represent the edge as the primary entity and the relations around it. The quad-edge and the lath-based data structures are implicit representations. The Star-Vertex data structure is an adjacency-based representation for cell complexes. The Corner Table is an adjacency-based representation specific for simplical complexes.

We will analyze and compare these data structures, also with respect to two-dimensional instances of the dimension-independent data structures, in terms of their space requirements and their efficiency in retrieving topological relations. We will focus our comparison on the case of triangle meshes by considering the special-

izations of representations for cell complexes to the simplicial case.

Explicit Edge-based Data Structures for Cell 2-Complexes The *Winged-Edge (WE) data structure* [5] is historically the first one proposed for cell 2-complexes. It encodes: (i) for each edge e, its two vertices, the two 2-cells (usually called *faces* in the context of boundary representations for solid objects) incident at e, and the four edges that are both adjacent to e and are on the boundary of the two faces incident at e (see Figure 4.4(a)); (ii) for each face f, a reference to one edge on the boundary of f; (iii) for each vertex v, a reference to one edge incident at v. It supports the retrieval of all topological relations in optimal time. Also the cells in the star of a vertex or on the boundary of a face can be traversed in both clockwise or counterclockwise directions. Given a cell 2-complex with n_2 faces, n_1 edges and n_0 vertices, the Winged-Edge data structure stores $n_2+8n_1+n_0$ pieces of topological information.

The *Doubly-Connected Edge List (DCEL) data structure* [61] is a simplified version of the WE representation, though it has been developed independently. For each edge e, instead of encoding all four edges on the boundary of the two faces incident at e (see Figure 4.4(b)), it stores only two edges, one for each of the two faces incident in e. The DCEL supports the traversal of all topological relations, but only in counterclockwise direction in the star of a vertex, and in clockwise direction around the boundary of a face. The DCEL data structure encodes $n_2+6n_1+n_0$ pieces of topological information.

The *Half-Edge (HE) data structure* [57] encodes two copies of each edge, each

of which is called a *half-edge*. A half-edge has a direction with respect to the face which it bounds. For each half-edge, the following information is encoded: its start vertex, the face associated with it, the previous and the next edges on the same face, the companion half-edge (see Figure 4.4(c)). Relations encoded at vertices and faces are the same as those encoded in the Winged-Edge and DCEL data structure. The Half-Edge data structure supports the retrieval of all topological relations in optimal time, and also the cells in the star of a vertex or on the boundary of a face can be traversed in both clockwise or counterclockwise directions. There are $n_2+10n_1+n_0$ pieces of topological information encoded in the HE data structure. Moreover, an implementation of the HE data structure which has the same storage cost as the WE data structure is described in [73].

The edge-based relations encoded in each of the edge-based data structures described are illustrated in Figures 4.4(a)-(c). All the edge-based data structures presented in this section encode, for each edge, relations $R_{1,0}$ and $R_{1,2}$, different partial $R^*_{1,1}$ relations, since only two or four edges are encoded, a partial $R^*_{0,1}$ relation for each vertex, which consists of one edge in the star of the vertex, and a partial $R^*_{2,1}$ relation for each face, which consists of one edge on the boundary of the vertex. All these data structures support the retrieval of all topological relations in optimal time.

The Quad-Edge and Lath-based Data Structures The *Quad-Edge data structure* [41] and the *Lath-based data structures* [48] are implicit data structures for cell 2-

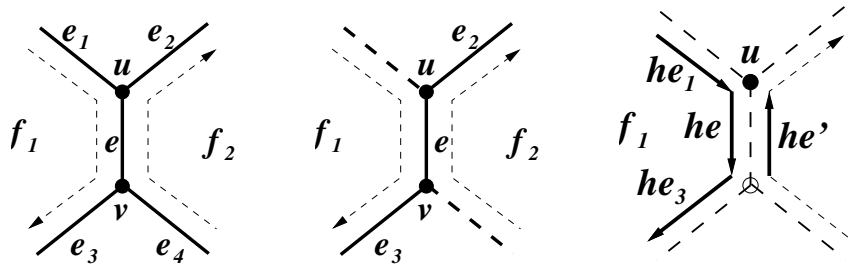


Figure 4.4: Edge-based relations represented in the edge-based data structures: (a) In the Winged-Edge data structure, e has a reference to $e_1, e_2, e_3, e_4, u, v, f_1$ and f_2. (b) In the DCEL, e has a reference to e_2, e_3, u, v, f_1 and f_2. (c) In the Half-Edge data structure, each edge e is represented as two half-edges he and he. Half-edge he has a reference to he', he_1, he_3, u and f_1

complexes with a manifold domain. In such complexes, edges in the star of each vertex can be ordered radially on a plane around the vertex, and the edges on the boundary of a face can be ordered clockwise or counterclockwise around the face. Thus, each edge belongs to four loops: the two at its extreme vertices, and the two at the faces sharing it. Such representations exploit this property.

In the Quad-Edge data structure, each *quad-edge* is associated with its two extreme vertices, its two adjacent faces and the next edges in its four loops. Basically, the Quad-Edge data structure encodes the same information as the Winged-Edge data structure. In a quad-edge that corresponds to edge e, the four adjacent edges of e are organized as part of the two loops around two faces, and two loops around two vertices. In the Winged-Edge data structure, the same four edges belong to the two loops of the two faces. Same relations at vertices and faces are encoded as in the Winged-Edge data structure. As the edge-based data structures presented in Subsection 4.1.2.1, the Quad-Edge data structure encodes partial relations $R^*_{2,1}$

for each face, partial relation $R^*_{0,1}$ for each vertex, complete relations $R_{1,0}$, $R_{1,2}$ and a partial version of relation $R_{1,1}$ for each edge. As in the other edge-based representations, all topological relations can be retrieved in optimal time. The storage cost of the Quad-Edge data structure is $n_2+8n_1+n_0$.

The *Lath-based* data structures are a collection of data structures that use vertices and *laths* as the basic elements. Each lath is uniquely identified with exactly one vertex, one edge and one face of a complex. A lath is conceptually similar to a cell-tuple. A Lath-based data structure requires no separate records for edges and faces. There are three variations in the encoding of a lath, giving rise to three data structures: the *Split-Edge*, the *Half-Edge-Lath* and the *Corner* data structures.

In the *Split-Edge* data structure, each lath se corresponds to one side of an edge and encodes a link to its start vertex u, a link to the lath se' of the other side of the same edge, and a link to the lath se_3 of the next edge in the clockwise direction on the same face (see Figure 4.5(a)).

In the *Half-Edge-Lath* data structure, each lath he (illustrated in Figure 4.5(b)) is associated with half of an edge. It encodes a link to its vertex u, a link to the lath he' of the other half of the same edge, and a link to the lath he'_1 of the next edge in the clockwise direction around the same vertex. Joy et al. called this version of the lath-based data structure the *Half-Edge* data structure, but the edge is halved differently from that of the Half-Edge data structure described above [57], and we call it the Half-Edge-Lath to distinguish it from the latter.

In the *Corner* data structure, a lath is associated with one corner of a vertex. Each lath u' encodes: a link to the vertex u, a link to the lath v' of the next vertex

v in the clockwise direction on the same face, and a link to the lath u''' of the next face in the clockwise direction around the same vertex.

All Lath-based data structures support the retrieval of all topological relations through laths in optimal time. However, because of the implicitness of the faces, edges and vertices, access from these cells to their associated laths is not time-efficient. All Lath-based data structures have the same storage costs, which is equal to $6n_1$, because each lath stores three pieces of topological information and the total number of laths is $2n_1$ in each case elaborated as follows. For the Split-Edge and Half-Edge-Lath structures, every edge corresponds to exactly two split-edge laths, exactly two half-edge laths, respectively. As far as the Corner data structure is concerned, the number of corners at each vertex is equal to the degree of the vertex (the number of edges incident at that vertex), while each edge is incident at exactly two vertices.

The Star-Vertex Data Structure The *Star-Vertex (SV) data structure* [49] is an adjacency-based data structure for manifold planar cell 2-complexes. The basic entity here is the vertex. For each vertex v, the Star-Vertex data structure encodes all the vertices in the link of v in counterclockwise order (see Figure 4.6(a)). For each vertex v' in the link of v, the data structure encodes a reference to the position of vertex v'' in the link of v', such that v, v', v'' are three consecutive vertices in clockwise order on the same face. Consider the example of face f in Figure 4.6(b). v_5, v_1 and v_2 are three consecutive vertices ordered in clockwise direction around f.

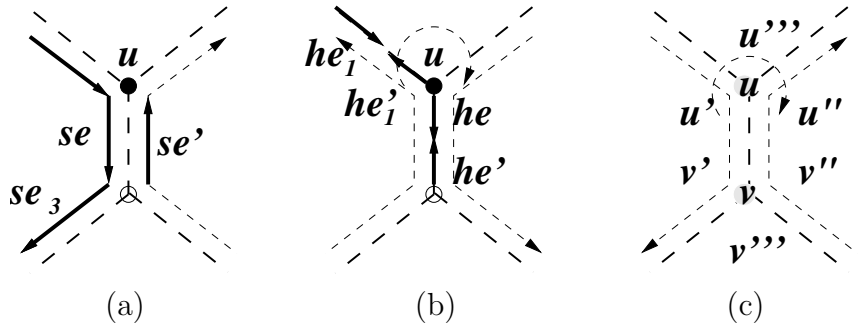
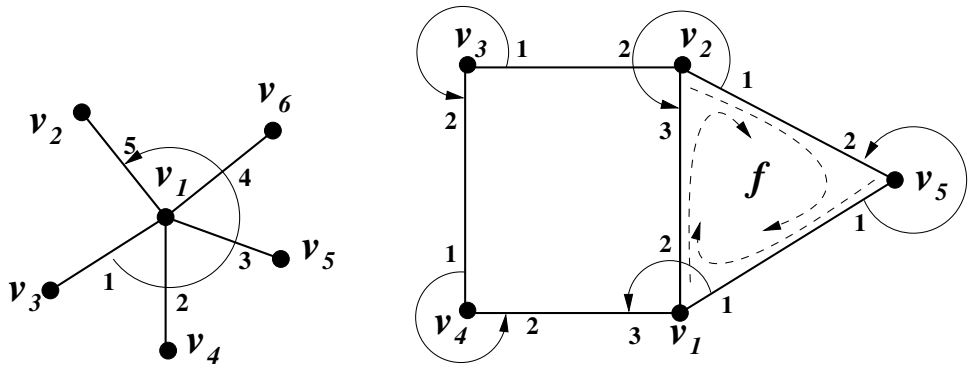


Figure 4.5: Relations encoded in a lath: (a) A Split-Edge Lath associates split-edge se with its opposite split-edge se', with its succeeding split-edge se_3 ordered clockwise on the same face, and with its start vertex u; (b) Half-Edge Lath associates half-edge he with its other half-edge he', with its start vertex u and with its succeeding half-edge ordered clockwise around u; (c) Corner Lath associates a corner u' of vertex u, with u itself, with the succeeding corner v' ordered clockwise on the same face, and with the succeeding corner u''' ordered clockwise on vertex u.

Vertex v_2 is the second vertex in the link of v_1. Therefore, the position 2 is encoded along with v_1 in as entry $(v_1, 2)$ at vertex v_5. The full encoding of the example is shown in the table shown in Figure 4.6(c). The vertices on boundary of the whole shape are order in counter clockwise.

In terms of topological relations, the Star-Vertex data structure encodes relation $R_{0,0}$ explicitly. Relation $R_{2,0}(f)$ is partially encoded with face f being implicitly described through one of its vertices. The Star-Vertex data structure only supports the retrieval of $R_{2,0}$ relation and $R_{0,0}$ relation in optimal time. Co-boundary relations cannot be retrieved locally.

The number of pieces of information encoded by the Star-Vertex data structure is twice the sum of the number of neighbors at all vertices, $\sum_v deg(v)$, because for



(a) (b)

v_1	$(v_5, 2)$, $(v_2, 1)$, $(v_4, 1)$
v_2	$(v_5, 1)$, $(v_3, 2)$, $(v_1, 3)$
v_3	$(v_2, 3)$, $(v_4, 2)$
v_4	$(v_3, 1)$, $(v_1, 1)$
v_5	$(v_1, 2)$, $(v_2, 2)$

(c)

Figure 4.6: (a) The link of vertex v_1 consists of $\{v_2, v_3, \cdots, v_6\}$ and is encoded by the Star-Vertex data structure at v_1 in the counter-clockwise order as labeled; (b) For each vertex in the link of v, a reference to the next vertex on the same face is encoded; (c) The information encoded for the example of (b), in which the left column is each vertex, and in the right column is the link of v. For each vertex in the link, the reference to the next vertex on the same face is encoded.

each neighbor, two pieces of information (i.e., the neighbor vertex, and the next vertex on face) are encoded. The term $\sum_v deg(v)$ is equal to twice the number of edges. Thus the storage cost is $4n_1$. In the case of a simplicial 2-complex with n_2 triangles, based on Euler's formula, this is approximately equal to $6n_2$. The Star-Vertex data structure encodes $6n_2$ pieces of information for a manifold simplicial 2-complex.

The Corner Table (CoT) Data Structure The *Corner Table (CoT) data structure* [71] is an adjacency-based data structure for manifold simplicial 2-complexes. A

corner is a unique index that is assigned to a triangle-vertex pair. It encodes the following information:

- For each triangle t, its three corners at its three incident vertices: v_1, v_2, v_3. (As illustrated in Figure 4.7(a), corner c describes the association between triangle t and vertex v_1);
- For each corner c of triangle t, let e be the edge of t that is opposite to c. Then the opposite corner of the triangle that shares e is associated to c. (In Figure 4.7(b), c' is the opposite corner of c.)

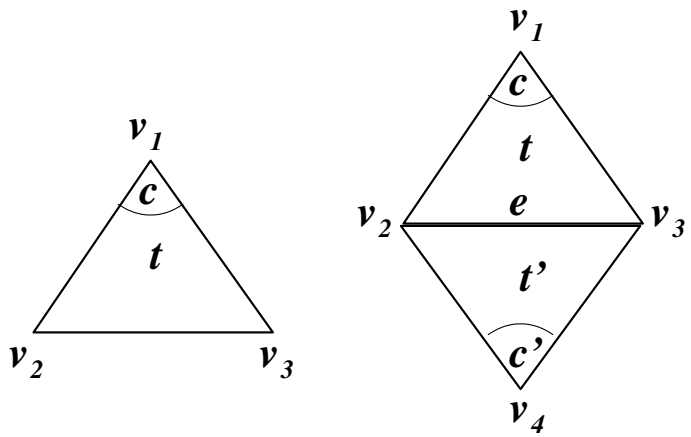


Figure 4.7: An illustration of the notion of corners in Corner Table: (a) Corner c associates triangle t with its incident vertex v_1; (b) The opposite corner of c is c'

A corner table is fully encoded in two arrays T and O. Corners are represented as the indices of both T and O. Array T stores the vertices of each triangle, ordered in counter-clockwise direction. Given a corner c as a index between 0 and $3n_2$ (n_2 being the number of triangles), the specific triangle associated by c is computable

as $t = c \ mod 3$ and the specific vertex associated by c is given by the array entry $T[c]$. The opposite corner of c is given by $O[c]$. The compactness of the corner table arises from using the order of array elements and the array indices to capture the association between a triangle and its vertices.

Formally, the Corner-Table data structure encodes the complete $R_{2,2}$ and $R_{2,0}$ relations. The $R_{0,2}$ relation is partially encoded through the corners. Direct access from an explicit vertex to its corners is not supported. All topological relations can be retrieved as corners from the Corner-Table in optimal time. The total amount of encoded information is equal to $6n_2$, of which $3n_2$ accounts for the vertices of the triangles, and $3n_2$ accounts for the opposite corner of each corner.

Comparisons We compare the data structures for manifold 2-complexes in terms of their characteristics, their space requirements and efficiency in supporting the retrieval of topological relations. Table 4.3 summarizes the characteristics of the various data structures in terms of the complexes they represent and of their representation method.

The storage cost of each data structure is evaluated based on the topological information encoded. We also consider the two-dimensional instances of the dimension-independent data structures, except for the Cell-Tuple data structure, which is the dimension-independent generalization of the Quad-Edge data structure. The storage costs of these data structures for a cell 2-complex with n_2 faces, n_1 edges and n_0 vertices, are listed below. From Euler's formula, we have that

Data Structure	Domain	Complexes	Method
Winged-Edge	Manifold	Cell	Edge-based
DCEL	Manifold	Cell	Edge-based
Half-Edge	Manifold	Cell	Edge-based
Quad-Edge	Manifold	Cell	Implicit
Lath	Manifold	Cell	Implicit
Star-Vertex	Manifold	Cell	Adjacency-based
Corner Table	Manifold	Simplicial	Adjacency-based

Table 4.3: Characteristics of the data structures for manifold 2-complexes

$n_2 \leq 2n_0$ and $n_1 \leq 3n_0$. Thus, for the sake of comparison, we can express all the storage costs in terms of the number of vertices.

- Winged-Edge: $n_2 + 8n_1 + n_0 \approx 27n_0$
- DCEL: $n_2 + 6n_1 + n_0 \approx 21n_0$
- Half-Edge: $n_2 + 10n_1 + n_0 \approx 33n_0$
- Quad-Edge: $n_2 + 8n_1 + n_0 \approx 27n_0$
- Laths: $6n_1 \approx 18n_0$
- SV: $4n_1 \approx 12n_0$
- IG: $8n_1 \approx 24n_0$

The storage costs of these data structures are evaluated for six manifold cell complexes and reported in Table 4.4. The Lath data structures are the most compact data structures for manifold cell 2-complexes, followed by the Incidence Graph. The

Data set	n_0	n_1	n_2	deg(V)	deg(F)
Football 1	1232	2340	1110	3.80	4.22
Football 2	930	1500	572	3.23	5.24
Crumb	312	564	254	3.62	4.44
Multidode	80	150	72	3.75	4.17
Torus	10.2k	20.5k	10.2k	4.00	4.00
Cone	641	1310	671	4.09	3.90

(a)

Data set	CT	IG	WE	DC	HE	QE	L	SV
Football 1	26.6k	18.7k	21.1k	16.4k	25.7k	21.1k	14.0k	9.36k
Football 2	13.7k	12.0k	13.5k	10.5k	16.5k	13.5k	9.00k	6.00k
Crumb	6.1k	4.5k	5.1k	4.0k	6.2k	5.1k	3.4k	2.26k
Multidode	1.7k	1.2k	1.4k	1.1k	1.7k	1.4k	0.9k	0.60k
Torus	246k	164k	184k	143k	225k	184k	123k	82.0k
Cone	16.1k	10.5k	11.8k	9.17k	14.4k	11.8k	7.86k	5.24k

(b)

Table 4.4: (a) Six data sets describing manifold cell 2-complexes: deg(V)=Average number of faces incident at a face, deg(F)=Average number of vertices on a face; (b) Storage cost of seven data structures for data sets in (a): CT (Cell Tuple), IG (Incidence Graph), WE (Winged Edge), DC (DCEL), HE (Half-Edge), QE (Quad-Edge), L (Lath), SV (Star-Vertex)

compactness of the lath-based data structures is achieved, however, through the implicit nature of the entities. Vertices, edges and faces are not explicitly addressable, but are implicitly encoded within the laths, and topological traversal is performed with laths as inputs and outputs. The IG, which is 1.33 times the size of the Laths data structures, on the other hand, explicitly represents all these entities. Explicit edge-based data structures generally are less space-efficient. Among them, the Half-Edge data structure has the largest space requirements, which is 1.8 times that of the Laths data structures. The Star-Vertex data structure is the most compact data structure, encoding only half the information of the IG. However, the Star-Vertex does not as the full navigation capacity as the other data structures.

The storage costs of the data structures for manifold simplicial 2-complexes and of the two dimensional instance of the EIA data structure are:

- Winged-Edge: $13n_2 + n_0 \approx 27n_0$
- DCEL: $10n_2 + n_0 \approx 21n_0$
- Half-Edge: $16n_2 + n_0 \approx 33n_0$
- Quad-Edge: $13n_2 + n_0 \approx 27n_0$
- Laths: $9n_2 \approx 18n_0$
- Corner Table: $6n_2 \approx 12n_0$
- Star-Vertex: $6n_2 \approx 12n_0$
- EIA: $6n_2 + n_0 \approx 13n_0$

We have evaluated the storage costs of these data structures for six data sets describing manifold simplicial 2-complexes and the results are reported in Table 4.5. We can see that the space requirements of the Corner-Table, of the EIA and of the Star-Vertex data structures are comparable, but all of them encode only vertices and triangles. When applied to simplicial complexes, the edge-based representations have the largest space requirements, at least twice the storage cost of those encoding only vertices and triangles. The lath-based ones are somehow in-between, and, as the edge-based representations, encode all the entities uniquely and explicitly.

Data set	n_0	n_1	n_2	deg(V)
Car	6.94k	18.0k	11.8k	5.09
Doll	551	1.38k	831	4.52
Face	2.09k	6.15k	4.05k	5.83
Temple	6.85k	17.8k	11.00k	4.82
Sofa	8.09k	23.5k	15.1k	5.61
Lion	5.17k	15.2k	10.1k	5.84

(a)

Data set	WE	DC	HE	QE	L	CoT	SV	EIA
Car	163k	127k	199k	163k	108k	70.7k	70.7k	77.7k
Doll	12.4k	9.65k	15.2k	12.4k	8.27k	5.0k	5.0k	5.54k
Face	55.3k	43.0k	67.6k	55.3k	36.9k	24.3k	24.3k	26.4k
Temple	160k	125k	196k	160k	107k	66.0k	66.0k	72.9k
Sofa	211k	164k	258k	211k	141k	90.8k	90.8k	98.9k
Lion	137k	106k	167k	137k	91.1k	60.4k	60.4k	65.5k

(b)

Table 4.5: (a) Six data sets of manifold simplicial 2-complexes: deg(V)=Average number of faces incident at a face; (b) Storage cost of nine data structures for data sets in (a): WE (Winged Edge), DC (DCEL), HE (Half-Edge), QE (Quad-Edge), L (Lath), CoT (Corner Table), SV (Star-Vertex) and EIA

Finally, we summarize in Table 4.6 the navigation costs by evaluating the optimality of algorithms for retrieving topological relations on the various representations.

Data Structure	Boundary relations	Co-boundary relations	Adjacency relations
Winged-Edge	Optimal	Optimal	Optimal
DCEL	Optimal	Optimal	Optimal
Half-Edge	Optimal	Optimal	Optimal
Quad-Edge	Optimal	Optimal	Optimal
Lath	Optimal	Optimal	Optimal
Star-Vertex	$R_{2,0}$: optimal Others: $O(n_0)$	$O(n_0)$	$R_{0,0}$: optimal Others: $O(n_0)$
Corner Table	Optimal	Optimal	Optimal

Table 4.6: Navigation performances of data structures for 2-dimensional complexes specific for manifold domains

4.1.2.2 Representations for Arbitrary Two-Dimensional Complexes

In this Subsection, we review representations for non-manifold shapes discretized through cell and simplicial complexes. The first data structure proposed in the literature for cell 2-complexes is the *Radial Edge (RE)* data structure [80], which has been extended and specialized in [42, 82]. More recent simplified representations are the *Partial Entities (PE)* data structure [51] and the *Loop Edge-use (LE)* data structure [58]. The PE data structure has the same representation power as the RE, but it is considerably more compact. The LE data structure is a specialization of the RE data structure to regular cell complexes.

We describe the Radial-Edge and the Partial-Entities data structures. Then,

we present and analyze in details data structures for simplicial 2-complexes, namely, an edge-based data structure, the *Directed Edge (DE)* data structure [7], which can be viewed as an extension of the Half-Edge data structure to the non-manifold simplicial case, an adjacency-based data structure, the *Triangle-Segment (TS)* data structure [25], which extends the EIA data structure to the non-manifold case, and an incidence-based data structure, called the *Vertex-Face (VF)* data structure [77]. We compare such representations based on their storage requirements and on their performance in retrieving topological relations, also with respect to the two-dimensional instance of the Incidence Graph described in Subsections 4.1.1.2. We also evaluate the scalability of the data structures to the manfiold case.

The Radial-Edge Data Structure The *Radial Edge (RE)* data structure [80] has been developed in order to describe the decomposition of the boundary of non-manifold and non-regular three-dimensional objects. As pointed out in Section 2, the decomposition is not a cell complex as defined in algebraic topology, since the 2-cells are not necessarily homeomorphic to closed disks, but they can be multiply connected 2-manifolds with boundary. The connected components formed by the edges bounding any 2-cell (face) are called *loops*. The entities in the RE data structure are thus: regions, shells, faces, loops, edges and vertices. A *region* is a solid objects, which is bounded by a collection of shells. (In Figure 4.8(a), there are three shells on a shape of two hollow cubes sharing a face.) A *shell* is thus an oriented boundary surface of a region, consisting of maximal connected sets of 2-cells (faces). In addition, faces, loops, edges and vertices are characterized by

orientations, namely *face-uses*, *loop-uses*, *edge-uses* and *vertex-uses*. A face f has two face-uses associated with it, which correspond to the two possible orientations of f (see Figure 4.8(b)). The oriented boundary of a face-use is described by loop-uses. A loop-use is composed of a circular list of edge-uses. Each edge-use associates an edge e with the orientation induced on e by the face-use to which it belongs. Thus, an edge-use represents the association between an edge and a face-use (see Figure 4.8(c)). A top 1-simplex, called a *wire-edge*, is described by two edge-uses forming a loop that connects the two boundary vertices of the wire-edge. Since each edge is bounded by two vertices, each edge-use is associated with a vertex-use. Thus, a vertex-use describes the association between a vertex and an edge-use that goes out from it (see Figures 4.8(d) and (e)).

Here, we present, for clarity, a simpler version of the RE data structure for representing an object described by a connected cell 2-complex, in which the 2-cells are homeomorphic to disks, and there are no isolated vertices. Thus, every face is bounded by exactly one loop. This simple version of the RE data structure does not contain high-level topological elements, namely, regions, and shells. This simple version of the RE data structure has the following entities: faces, edges, vertices, face-uses (which also capture their oriented boundaries originally described by loop-uses), edge-uses and vertex-uses. It encodes only the following information:

- For each face f, a reference to a face-use (for example, in Figure 4.9(a) f_1 points to f_{u1});

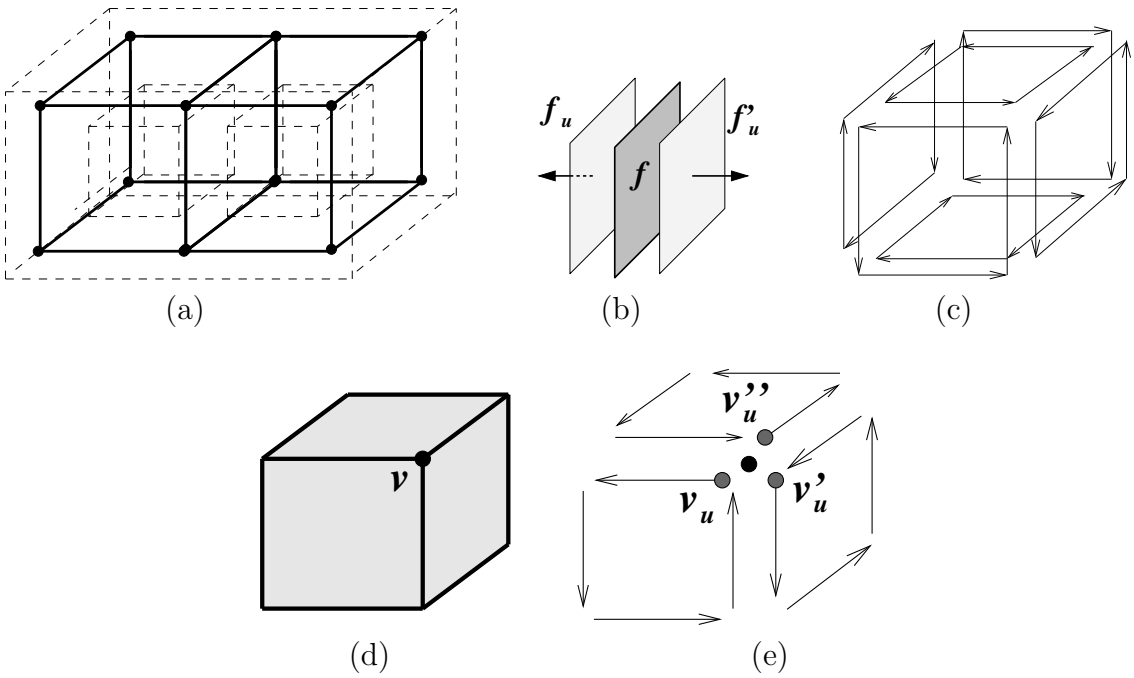


Figure 4.8: An illustration of the entities in the RE data structure: (a) Three shells exist in a complex describing two hollow cubes sharing a common face; (b) Each face f has two face-uses f_u and f'_u; (c) Each face-use on the inner shell of a cube is bounded by a loop-use which is composed of a circular list of edge-uses; (d) and (e) V vertex v shared by three faces, and the vertex-uses v_u, v'_u and v''_u that start at v.

- For each face-use f_u (see f_{u1} of Figure 4.9(a)):
  - the face f with which it is assocciated (f_1 in the example);
  - a reference to the other face-use associated with f (f_{u2} in the example);
  - a reference to an edge-use on f_u (e_{u1} in the example);
- For each edge e, a reference to one edge-use that is associated with e (for example, in Figure 4.9(b) e refers to e_{u1});
- For each edge-use e_u in the face-use f_u of face f (such as e_{u1} in Figure 4.9(b)):
  - the corresponding undirected edge e;
  - its face-use f_u (f_{u1} in Figure 4.9(b));
  - the mate edge-use in the other face-use of f (e_{u2} in Figure 4.9(b));
  - the adjacent edge-use radially ordered around e (that is e_{u6});
  - the previous edge-use in f_u (in Figure 4.10, the previous edge-use of e_{u1} is e_{u4});
  - the next edge-use in f_u (in Figure 4.10, the next edge-use of e_{u1} is e_{u2});
  - the start vertex-use of e_u (in Figure 4.10, the start vertex-use of e_{u1} is v_{u1});
- For each edge-use e_u which is associated with wire-edge e:
  - its wire-edge e;
  - the previous edge-use associated with e;

  - the next edge-use associated with e;
  - the start vertex-use of e_u;

- For each vertex v, a reference to one vertex-use associated with v (in Figure 4.11, v has a reference to v_{u1};
- For each vertex-use v_u that is associated with one edge-use e_u (such as vertex-use v_{u1} of Figure 4.11):
  - the corresponding undirected vertex v;
  - the previous vertex-use of v (v_{u5} in Figure 4.11);
  - the next vertex-use of v (v_{u2} in Figure 4.11);
  - its edge-use e_u (e_{u1} in Figure 4.11);

While the edge-uses around an edge can be ordered, the vertex-uses at a vertex cannot be ordered. Therefore, the list of vertex-uses at vertex v simply collect all the vertex-uses at v.

The RE data structure can be formalized in terms of topological relations as follows (note that the formalization does not take into account the orientations captured by face-uses, edge-uses and vertex-uses):

- For each face f: relation $R^*_{2,1}(f)$, which consists of one edge on the boundary of f,

- For each edge e:
  - relation $R_{1,2}(e)$, in which the faces are ordered around e;
  - partial relation $R^*_{1,1}(e)$, defined as the collection of the pair of edges adjacent to e and bounding the faces incident in e, ordered around e, so that both the $2i$-th element and the $(2i+1)$-element in this relation belong to the i-th face in $R_{1,2}(e)$;
  - relation $R_{1,0}(e)$, ordered by the indices of the vertices;
- For each vertex v: relation $R_{0,1}(v)$, unordered.

Relations $R_{1,2}(e)$, $R^*_{1,1}(e)$ and $R^*_{1,0}(e)$ for edge e describe the information encoded at edges. $R_{1,2}(e)$ describes the relation between an edge and a face defined by an edge-use. Relation $R^*_{1,1}(e)$ captures the association between an edge-use e_p and the edges following and preceding e_p in the boundary of the face f with which e_p is associated. The adjacency of edge-uses at the same edge e is implicitly expressed through the order in $R^*_{1,1}(e)$. It can be shown that all topological relations can be retrieved in optimal time from the RE data structure.

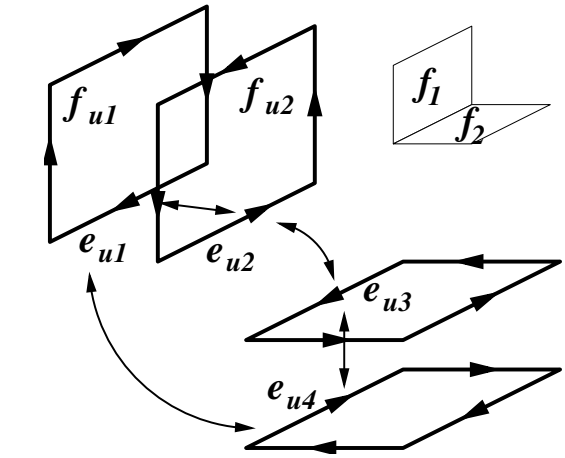
The Tri-Cyclic Cusp representation [42] extends the RE data structure with new elements (called *cusps*) introduced in order to handle the inclusion relations of topological disks at non-manifold vertices. The Coupling Entities representation [82] is an improvement over both the RE and the Tri-Cyclic Cusp data structures, obtained by introducing additional entities that describe the relationships at the loops formed by edges around faces, the radial cycles formed by faces around edges and cycles formed by faces at vertices. The RE representation is not highly scalable

to the degree of manifold singularities of a shape. It has been shown in [51] that, when the domain is manifold, the RE representation requires about four times as much storage space as the manifold representations such as the Winged-Edge data structure described in Section 4.1.2.1, which resembles the RE data structure in terms of the entities and relations encoded.

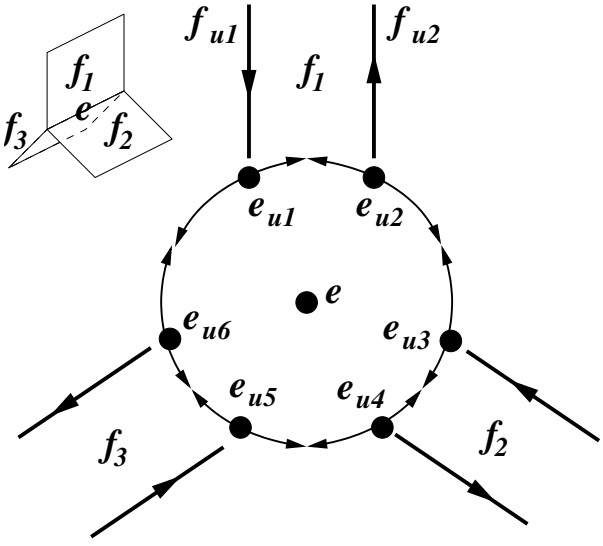
The Partial Entities Data Structure The *Partial Entities (PE) data structure* [51] describes the boundary description of a non-manifold solid through *regions*, *shells*, *faces*, *loops*, *edges*, and *vertices*, *partial-faces*, *partial-edges* and *partial-vertices*.

Each face has a unique orientation defined based on the geometry of its surface normal. A partial face describes one of the two orientations of a face. Each face is bounded by one loop, which consists of a cycle of partial-edges. Each partial-edge corresponds to the appearance of an edge on a loop bounding a face. Thus, if there are m faces incident at edge e, the PE data structure stores m partial-edges corresponding to e. A top 1-simplex we, called a *wire-edge*, has a loop that consists of two partial-edges of we. The partial-edge is comparable to the edge-use in the RE data structure, except that each edge-use is associated with one face-use, while a partial-edge is associated with a face. As there are two face-uses for each face in the RE data structure, the number of edge-uses in the RE data structure is twice the number of partial edges in the PE data structure. Partial-faces are the defining components of shells. Each face has two partial-faces as a face may belong to two shells. A partial-vertex is a copy of a vertex created for each manifold surface sharing it.

The PE data structure is designed to encode objects with several boundaries and several connected components. By limiting the domain to objects with just one connected component, and with faces that are homeomorphic to 2-disks, and are thus bounded by one loop, we have simplified the original PE data structure for the purpose of highlighting its capability in representing the connectivity among the entities of a cell complex. Therefore, high-level topological elements, namely, regions and shells are not represented. We also do not consider partial-faces because their primary function is just to describe the orientation of a face, but all other entities refer to faces and not to partial faces as described above. The simplified version of the PE data structure encodes the following information (we refer to Figures 4.12(a) and (b) to illustrate it):



(a)



(b)

Figure 4.9: (a) Edge-uses radially ordered around an edge between two faces; (b) Cross-section view of edge-uses radially ordered around an edge between three faces

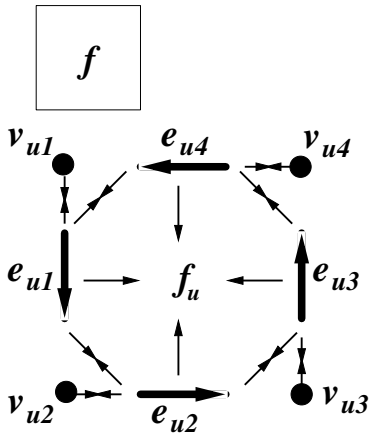


Figure 4.10: Planar view of a loop of edge-uses bounding a face-use f_u of face f;

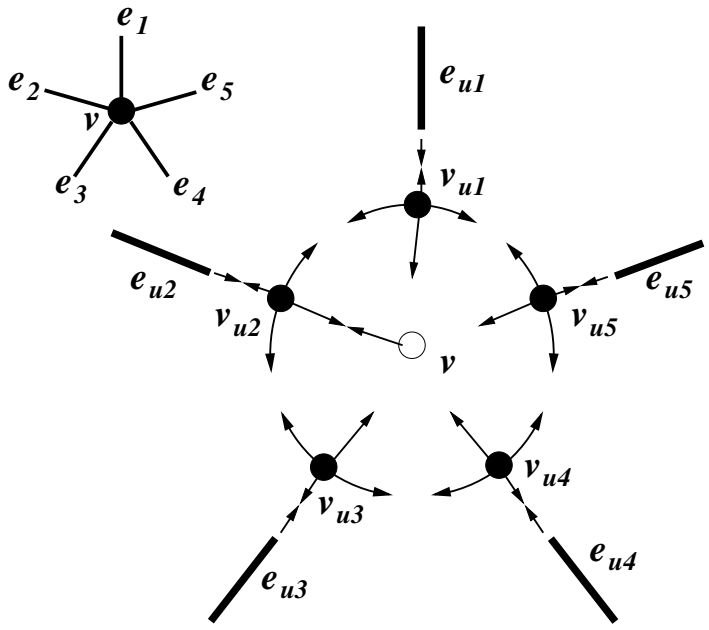


Figure 4.11: Vertex-uses of the same vertex v

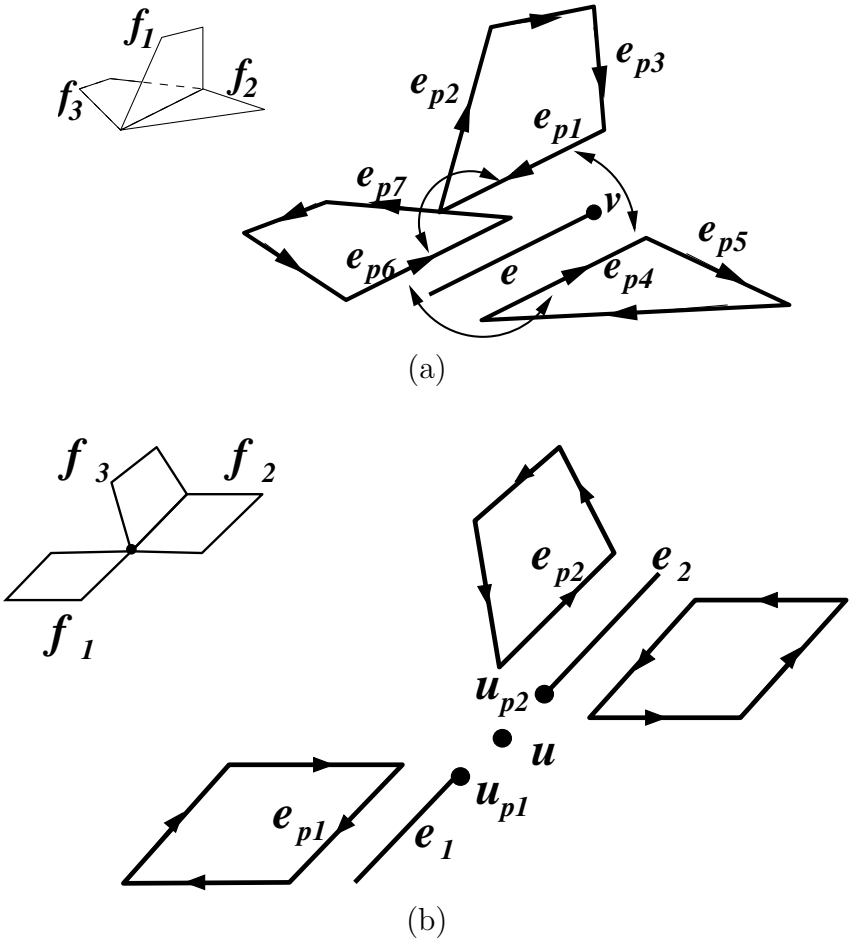


Figure 4.12: Elements of the PE structure: (a) relations at a non-manifold edge e shared by three faces: f_1, f_2, f_3; (b) relations at a non-manifold vertex u shared by two manifold components.

- For each face f: a reference to a partial-edge on its boundary (In Figure 4.12(a), f_1 has a reference to e_{p1});

- For each edge e, a reference to a partial-edge that describes e (In Figure 4.12(a), e has a reference to e_{p1});

- For each partial-edge e_p bounding face f, a reference to: (e_{p1} in Figure 4.12(a) as an example)

  - the corresponding edge e (e in the example);
  - the face f (f_1 in the example);
  - the previous adjacent partial-edge ordered in counter-clockwise direction around e (e_{p4} in the example);
  - the next adjacent partial-edge ordered around e (e_{p6} in the example);
  - the previous partial-edge in counter-clockwise direction on the boundary of f (e_{p3} in the example);
  - the next partial-edge on the boundary of f (e_{p2} in the example);
  - the partial-vertex of e_p (v in the example);

- For each partial-edge e_p bounding wire-edge e, a reference to:
  - the wire-edge e;
  - the next partial-edge bounding e;
  - the previous partial-edge bounding e;
  - the partial-vertex of e_p

- For each vertex v: the list of all partial-vertices v_p that are associated with v (In Figure 4.12(b), u has references to u_{p1} and u_{p2}.)

- For each partial-vertex v_p associated with vertex v (see u_{p1} in Figure 4.12(b) as an example):
  - the vertex v (u in the example);
  - a partial-edge that starts at v_p (e_{p1} in the example).

We can express the information encoded in the specialized PE data structure in terms of topological relations as follows:

- For each face f: relation $R^*_{2,1}(f)$, which encodes one edge on the boundary of f,
- For each edge e:
  - relation $R_{1,2}(e)$, ordered around edge e;
  - Partial relation $R^*_{1,1}(e)$ which is defined as follows: $R^*_{1,1}(e)$ consists of the edges on the boundary of the faces incident at e and sharing one extreme vertex with e. The elements in relation $R^*_{1,1}(e)$ are ordered so that both the $2i$-th and the $(2i+1)$-th elements in $R^*_{1,1}(e)$ are on the i-th triangle in $R_{1,2}(e)$.
  - Relation $R_{1,0}(e)$;
- For each vertex v: partial relation $R^*_{0,1}(v)$ which consists of one edge for each connected component of the link of v.

Relations $R_{1,2}(e)$, $R^*_{1,1}(e)$ and $R_{1,0}(e)$ for edge e describe the information encoded at edges. $R_{1,2}(e)$ describes the relation between an edge and a face defined by a partial-edge. Relation $R^*_{1,1}(e)$ captures the association between a partial-edge e_p and the edges following and preceding e_p in the boundary of the face f with which e_p is associated. The adjacency of partial-edges at the same edge e is implicitly expressed through the order in $R^*_{1,1}(e)$. Thus, the PE data structure encodes the same relations as the RE one, with the exception of $R_{0,1}(v)$ relation at vertex v

which is partially encoded in the PE but fully encoded in the RE. All topological relations can be retrieved in optimal time from the PE data structures, as described in [51].

In [51], an implementation of the PE representation is presented that has half the storage cost of the RE representation for non-manifold cell 2-complexes, and uses twice as much space as that of the Winged-Edge representation for manifold cell 2-complexes (see Section 4.1.2.1).

The primary difference between the RE and the PE data structure is that the PE data structure considers each face to have one orientation geometrically defined based on its face normal. The orientation of its boundary can thus be uniquely defined. In the RE data structure, a face entity is without orientation. The face-uses of the RE data structure describe all the possible orientations of each face. In the RE data structure, the connectivity among the faces, edges and vertices is defined through face-uses, edge-uses and vertex-uses. In the PE data structure, however, the connectivity among faces, edges and vertices is captured at the faces, at partial-edges and at partial-vertices.

The Directed Edge Data Structure The *Directed-Edge (DE) data structure* [7] is an extension of the Half-Edge data structure [57], proposed for cell 2-complexes with a manifold domain, to simplicial 2-complexes embedded in the three-dimensional Euclidean space. The DE data structure is based on the concept of directed edge. A *directed edge* e_d of an edge e in a simplicial 2-complex is an occurrence of e on the boundary a triangle incident at e. A directed edge is similar to the *edge-use* and to

the *partial-edge* in the RE and PE data structures, respectively.

In the DE data structure, the entities stored are directed edges and vertices. Triangles and undirected edges are not explicitly encoded. Triangles are implicitly referenced through the edges on their boundary. The association between a triangle and its three edges is through indexing. The i-th triangle f_i is implicitly described by the $3i$-th, $(3i+1)$-th and $(3i+2)$-th directed edges, which form the oriented boundary of f_i. Wire-edges are represented as directed edges. Thus, the DE data structure encodes the following information:

- Each triangle f is implicitly described by the three directed edges on the boundary of f;
- For each directed edge e_d on the boundary of face f, there is a reference to each of the following entities (see Figure 4.13 for the illustration of the symbols)
  - its start vertex v_1;
  - its end vertex v_2;
  - the adjacent directed edge e_r that is incident at v_1 and v_2;
  - the previous directed edge e''_d bounding f in counter-clockwise order;
  - the next directed edge e'_d bounding f in counter-clockwise order;
- For each directed wire-edge e:
  - its start vertex v_1;
  - its end vertex v_2;

- For each vertex v, one directed edge for each connected component of the link of v.

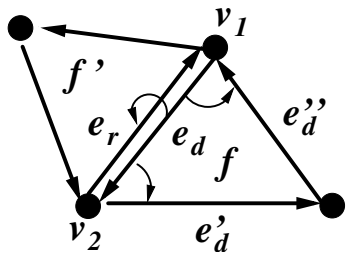


Figure 4.13: An illustration of the relations encoded at a directed edge e_d

The topological relations encoded in the DE data structure are:

- For each face f: $R_{2,1}(f)$, which is encoded implicitly (the i-th directed edge belongs to the $(i/3)$-th triangle);

- For each edge e:

  - Relation $R_{1,0}(e)$;

  - Partial relation $R^*_{1,1}(e)$, as defined for the RE data structure;

- For each vertex v: partial relation $R^*_{0,1}(v)$, which consists of one edge for each connected component of the link of v.

In our formalization, we have considered the undirected edge e, instead of its oriented version and, thus, the information in the directed edges in the DE data structure has been transferred to the undirected edge and described by partial

relation $R^*_{1,1}(e)$, and in its ordering. Note that the DE data structure encodes exactly the same relations as the PE data structure.

The DE data structure can implemented at three levels of detail. In the full-sized level, for each directed edge e_d, a reference is encoded to vertices v_1, v_2, and directed edges e_r, e'_d and e''_d (see Figure 4.13). In the medium-sized level, only v_2, e_r and e''_d are encoded for e_d. In the small-sized level, each e_d has a reference only to v_2 and e_r.

The DE data structure is highly scalable to the degree of manifoldness in a simplicial 2-complex. A cost-effective implementation reported in [7] has a storage cost of $68n_2$ bytes, which is 1.13 times the cost of the Winged-Edge data structure for representing 2-manifolds. The DE data structure is also highly adaptable to the availability of memory space by trading off the amount of topological information encoded with assess time. The full-sized level implementation has a storage cost of $68n_2$ bytes, while the medium-sized and the small-sized levels have respectively $44n_2$ bytes and $32n_2$ bytes.

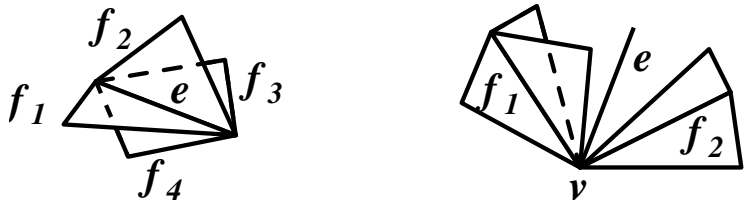
The Triangle-Segment (TS) Data structure The *Triangle-Segment (TS) data structure* [25] describes simplicial 2-complexes embedded in the two-dimensional Euclidean space. It extends the 2D instance of the EIA data structure (see Section 4.1.1.3) to the non-manifold domanin. It encodes all the vertices, the top 2-simplexes, i.e., the triangles, and the top 1-simplexes, that we call *wire-edges*, together with the following topological relations (see Figure 4.14):

- For each triangle t:

- boundary relation $R_{2,0}(t)$;
- a partial $R^*_{2,2}(t)$ relation defined as follows: for each edge e of t. $R^*_{2,2}(t)$ encodes the triangle(s) that are immediately preceding and succeeding t in counter-clockwise order around edge e. In the example of Figure 4.14(a), $R^*_{2,2}(f_2) = \{f_1, f_3\}$.

- For each wire-edge we, boundary relation $R_{1,0}(we)$;

- For each vertex v:
  - a partial $R^*_{0,2}(v)$ relation, which encodes one triangle for each connected component of the link of vertex v, as illustrated in the example of Figure 4.14(b), $R^*_{0,2}(v) = \{f_1, f_2\}$;
  - a partial $R^*_{0,1}(v)$ relation, which encodes the list of the wire-edges in the star of vertex v, for example, $R^*_{0,1}(v) = \{e\}$ in Figure 4.14(b).



(a) (b)

Figure 4.14: (a) Non-manifold edge e shared by three faces; (b) Non-manifold vertex v shared by two connected components and wire-edge e

In the TS data structure, only wire-edges are explicitly encoded, but not the edges bounding triangles. In a compact implementation of the TS data structure,

relation $R^*_{2,2}$ is implemented through arrays and bit flags, while relations $R^*_{0,1}(v)$ and $R^*_{0,2}(v)$ are implemented as linked lists. It has been shown in [25] that the TS data structure supports the retrieval of all topological relations in optimal time.

In [25], a highly manifold-scalable implementation of the TS data structure is reported which has a storage cost overhead of one byte per vertex in storage cost for manifold simplicial 2-complexes. This has been evaluated by comparing with the storage cost of the EIA data structure for manifold simplicial 2-complexes.

The Vertex-Face Data Structure The *Vertex Face (VF) data structure* [77] has been developed to describe regular simplicial complexes (i.e., simplicial 2-complexes without wire-edges). It encodes all vertices, edges, triangles explicitly and the following topological relations:

- For each triangle t, boundary relation $R_{2,1}(t)$
- For each edge e, boundary relation $R_{1,0}(e)$
- For each vertex v, co-boundary relation $R_{0,2}(v)$

Boundary relations as well as co-boundary relations based on vertices, namely relations $R_{0,0}(v)$, $R_{0,1}(v)$ and $R_{0,2}(v)$ can be retrieved in optimal time. Edge-based co-boundary relations are retrieved in sub-optimal time, since we need to consider $R_{0,2}$ relation for both the extreme vertices of any edge. As a consequence, all adjacency relations are sub-optimal. Algorithms for retrieving topological relations are reported in [46].

Comparisons We compare the data structures for cell and simplicial 2-complexes in terms of their characteristics, their space requirements and their efficiency in supporting topological navigation. Table 4.7 summarizes the domain, the kinds of complexes which can be described by the various data structures, and the representation methods.

Data Structure	Domain	Complexes	Method
RE	Non-manifold	Cell	Edge-based
PE	Non-manifold	Cell	Edge-based
DE	Non-manifold	Simplicial	Edge-based
TS	Non-manifold	Simplicial	Adjacency-based
VF	Regular	Simplicial	Incidence-based

Table 4.7: Characteristics of the data structures for arbitrary 2-dimensional complexes

We evaluate the storage costs of the data structures reviewed for simplicial 2-dimensional complex, plus the 2D instance of the Incidence Graph. Consider a simplicial 2-dimensional complex with n_2 triangles, n_1 edges, of which n_1^t are wire-edges, and n_0 vertices. The total number of connected components at the link of non-manifold edges is denoted by C_e, and the total number of connected-components at all vertices is denoted by C_v. The storage cost of each data structure is as follows:

- RE : $73n_2 + n_1 + 4n_1^t + n_0$
- PE : $22n_2 + n_1 + 4n_1^t + 3C_v$
- DE (full-sized) : $15n_2 + 2n_1^t + C_v$

- TS : $6n_2 + C_e + C_v$
- VF : $6n_2 + 2n_1$
- IG : $6n_2 + 4n_1$

In Table 4.8, we report an evaluation of the amount of information encoded by these data structures for some simplicial 2-complexes. Among the three edge-based data structures, namely, the RE, the PE and the DE data structures, the RE data structure encodes three times the number of topological elements encoded by the PE, which is about 1.6 times that of the DE. The TS is the most compact among the seven data structures compared in Table 4.8. The incidence-based data structures, i.e., the VF and the IG are less space-consuming than the edge-based ones, but not as compact as the adjacency-based TS data structure.

We summarize in Table 4.9 the navigation costs by evaluating the optimality of the algorithms for retrieving topological relations on the various representations.

4.1.3 Data Structures for 3D Complexes

In this Section, we discuss representations for cell and simplicial 3-complexes embedded in the three-dimensional Euclidean space. There are relatively few representations for describing 3D shapes discretized as cell and simplicial 3-complexes. Most of such representations are limited to the manifold domain. Representations for manifold cell complexes are the Facet-Edge [30, 60] and the Handle-Face [54] data

Data set	n_0	n_1	n_2	n_1^t	C_e	$C_v - n_0$
cylinders	91	300	204	0	16	2
pies	696	2.98k	2.30k	0	1.92k	72
frame	987	2.16k	1.08k	216	0	390
cubes	2.20k	10.7k	9.60k	0	14.9k	0
densetower2	9.13k	27.7k	18.4k	160	1.92k	480

(a)

Data set	RE	PE	DE	TS	VF	IG
cylinders	15.3k	5.07k	3.15k	1.33k	1.82k	2.42k
pies	172k	56.0k	35.3k	16.5k	19.8k	25.7k
frame	82.9k	30.1k	18.0k	7.86k	10.8k	15.1k
cubes	714k	228k	146k	74.7k	79.0k	100k
densetower2	1,380k	462k	286k	122k	166k	221k

(b)

Table 4.8: (a) Five non-manifold 2D simplicial data sets: C_e=# connected components at non-manifold edges, C_v=# connected components at all vertices; (b) Storage cost of seven data structures for 2D data sets in (a)

structures. The Compact Half-Face (CHF) data structure [50] is specific for manifold simplicial 3-complexes (these latter are usually called *tetrahedral meshes*). The only data structure that can represent 3D complexes with non-manifold properties is the 3D instance of the dimension-independent Incidence Graph (IG) representation. In this Section, we analyze and compare such representations also with respect to the three-dimensional instance of IG discussed in Section 4.1.1.

4.1.3.1 The Facet-Edge (FE) Data Structure

The *Facet-Edge (FE)* data structure [30] is an extension of the *Quad-Edge* data structure developed for cell 2-complexes (see Subsection 4.1.2.1), and thus it is an implicit representation for manifold cell 3-complexes. The basic entities

Data Structure	Boundary relations	Co-boundary relations	Adjacency relations
RE	Optimal	Optimal	Optimal
PE	Optimal	Optimal	Optimal
DE	Optimal	Optimal	Optimal
TS	Optimal	Optimal	Optimal
VF	Optimal	$R_{0,k}$: optimal Others: sub-optimal	Sub-optimal

Table 4.9: Navigation performance of data structures for non-manifold 2-dimensional complexes

encoded in the Facet-Edge data structure are the vertices and the so-called *facet-edges* defined on the 2-cells and 1-cells (faces and edges). The 3-cells and their topological information are encoded through the topological vertex-based relations of the complex which is dual to the given one. The 0-cells of the dual complex correspond to the 3-cells of the original complex, its 1-cells to the faces, its 2-cells to the edges and its 3-cells to the vertices.

The boundary of each face (2-cell) f contains a ring of edges (1-cells) $e_1, .., e_n$. This ring is called a *face-ring*, and may be ordered in two directions. The star of an edge e contains a ring of faces $f_1, ..., f_m$. This ring is called an *edge-ring* and can also be ordered in two directions. A *facet-edge* pair uniquely associates a face f with an edge e on the boundary of f. Each facet-edge pair exists in four versions. Each version is associated with exactly one face-ring of f and exactly one edge-ring of e. Each version of a facet-edge has its dual which is defined in the dual complex. Figures 4.15 to 4.17 illustrate the concept of facet-edge. The four unique facet-edges formed by face f_1 and edge e_1 in Figure 4.15(a) are shown in Figures 4.15(b)-(e). Figure 4.16(a) shows the dual complex of the one shown in Figure 4.15(a), in which

edge e_1^* is the dual of f_1 and the highlighted faces correspond to the edges of f_1. Figure 4.16(b) shows the dual of the facet-edge shown in Figure 4.15(b).

Given a face f and an edge e on its boundary, the four versions of facet-edges and the duals of each of them are related by the following operators.

- *Clock*, which returns the facet-edge with the face-ring in reversed direction;
- *Rev*, which returns the facet-edge with the edge-ring in reversed direction;
- *Fnext*, which returns the facet-edge of the next face in the same edge-ring as f;
- *Enext*, which returns the facet-edge of the next edge in the same face-ring as e;
- *Dual*, which returns the facet-edge with the same orientation in the dual complex.

Operators *Clock*, *Rev*, *Fnext* and *Enext* are illustrated in Figure 4.17 for the facet-edge shown in Figure 4.15(b).

In addition, the relations between the facet-edges and their vertices are described by the operator *Org*, which returns the start vertex of each facet-edge. *Org* induces a partition (known as *origin partition*) on the set of facet-edges in the complex and in its dual complex. Note that, while the origin partition in the 3-complex captures the incidence relations between edges and vertices, such a partition in the dual complex captures the incidence relations between the 3-cells and their bounding faces.

The Facet-Edge data structure describes a complex Σ by encoding, for each facet-edge pair a associated with edge e in a face-ring and with face f in an edge-ring, the preceding and succeeding facet-edges in both the face-ring and the edge-ring of a. The successors of a in the face-ring in two opposite directions correspond to *aFnext* and *aClockFnext*. The successors of a in the edge-ring in two opposite directions correspond to *aDualFnext* and *aDualClockFnext* in the dual-complex of Σ. For the example of Figure 4.15(a), the successors of (f_1, e_1) in the face-ring of e_1 in both directions are respectively (f_2, e_1) and (f_3, e_1). The successors of (f_1, e_1) in the edge-ring of f_1 are (f_1, e_2) and (f_1, e_3), which correspond to the facet-edges (f_2^*, e_1^*) and (f_3^*, e_1^*) shown in Figure 4.16(a).

The Facet-Edge data structure also encodes the vertex-based functions by implementing the partition of the facet-edges induced by the *Org* operator on the 3-complex and on the dual complex.

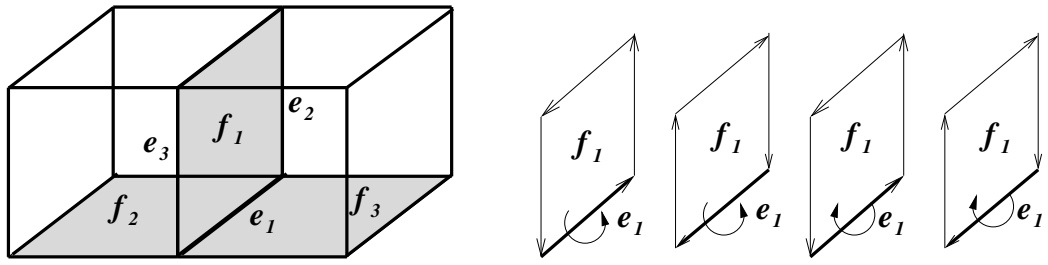


Figure 4.15: An illustration of the concept of facet-edge: (a) a model of two cubes; (b)-(e) the four facet-edge pairs formed by face f_1 and edge e_1

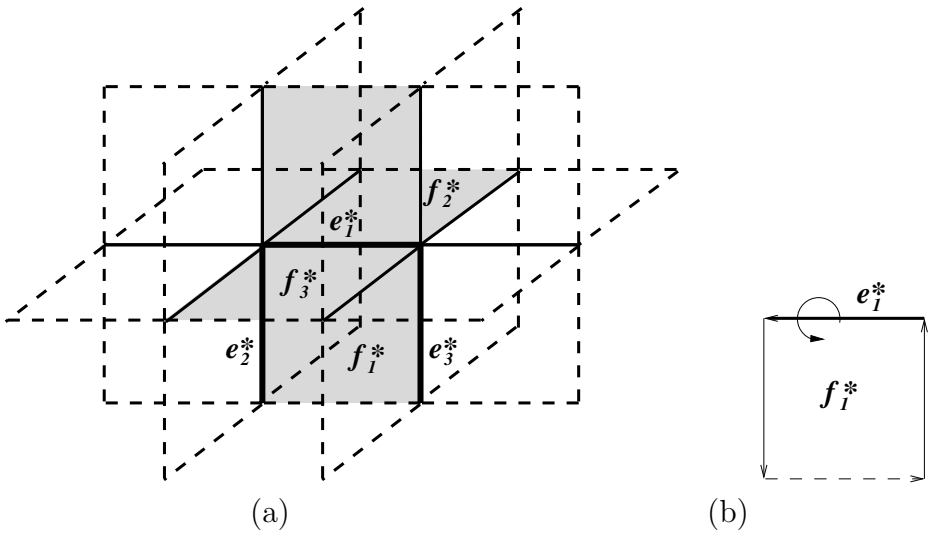


Figure 4.16: An illustration of the dual of a facet-edge: (a) the dual complex of Figure 4.15(a); (b) the dual of the facet-edge shown in Figure 4.15(b);

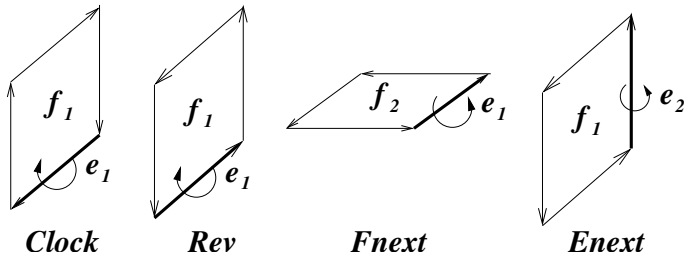


Figure 4.17: An illustration of the operators defined on a facet-edge: the facet-edges mapped from Figure 4.15(b) by operators *Clock*, *Rev*, *Fnext*, and *Enext*, respectively

In terms of topological relations, the FE data structure encodes relations $R_{2,1}$ in the form of facet-edges, relation $R_{1,2}$ in the form of face-rings, partial relation $R^*_{1,1}$ in the form of edge-rings, relations $R_{1,0}$ and $R_{2,3}$ implicitly as the incident vertices of the edges, and relation $R_{0,1}$ as the origin partition. Relation $R_{3,2}$ is implicitly encoded as the origin partition of the dual complex. It can be shown that topological relations can be retrieved from the FE data structure in optimal time.

The storage cost of the FE data structure for a manifold cell 3-complex with n_3 3-cells, n_2 faces, and n_0 vertices is $4\sum_{i=1..n_2} deg(f_i)+\sum_{i=1..n_3} deg(c_i)+\sum_{i=1..n_0} deg(v_i)$, where $deg(f_i)$ is the number of edges on the i-th face, $deg(c_i)$ the number of faces on the i-th 3-cell and $deg(v_i)$ the degree of vertex v_i. The term $4\sum_{i=1..n_2} deg(f_i)$ derives from the encoding of the successors on the edge-ring and face-ring of each facet-edge. The remaining two terms derive from the vertex-based relations. In the case of simplicial 3-complexes, $deg(f_i) = 3$, $deg(c_i) = 4$ and $\sum_{i=1..n_0} deg(v_i) = 2n_1$, so the amount of information encoded by the FE data structure is $4n_3 + 12n_2 + 2n_1$.

Unlike incidence-based and adjacency-based representations, the FE data structure does not explicitly encode cells as entities but it preserves the orientation of the cells, which makes it a suitable choice for applications that depend on the orientation of cells. The FE data structure has been specialized to the simplicial case without duals in *Triangle-Edge data structure* [60]. By limiting the scope to simplicial complexes, there is a constant number of triangle-edge pairs for each triangle, which allows for an efficient implementation. The Triangle-Edge data structure has been employed in the application of computing a tetrahedralization of a solid object and of the simplification of tetrahedral meshes [63].

4.1.3.2 The Handle-Face (HF) Data Structure

The *Handle-Face (HF) data structure* [54] is an explicit representation for manifold cell 3-complexes.

It is similar to representations for cell 2-complexes embedded in the three-

dimensional Euclidean space, such as the RE and the PE data structures (see Section 4.1.2.2), since the 3-cells are described in the HF data structure through their boundaries, made by faces, edges and vertices. The HF data structure contains two types of entities: the *basic entities*, which are the *faces*, *edges* and *vertices* in the cell complex, and the *surface entities*, which describe the boundary of each 3-cell of the complex. The HF data structure further distinguish between surface entities that are on the boundary of the entire manifold shape and surface entities that are in the interior. Here we present a simplified version of the HF data structure in which the surface entities simply describe the surface of each 3-cell. The surface entities are *surfaces*, *half-faces*, *surface edges*, *surface-oriented edges* and *surface vertices* which are described below.

- A surface refers to the surface of a 3-cell. Each surface is formed by a collection of half-faces which encloses the volume occupied by the 3-cell.

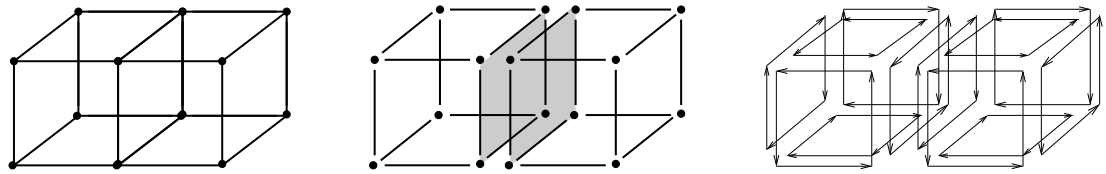
- A half-face corresponds to one orientation of a face in the cell complex. Each half-face hf associates face f to the surface of a 3-cell sharing f. hf is bounded by a cycle of surface-oriented edges, whose orientation is aligned with the orientation of the half-face.

- Each surface-edge se associates an edge e to a 3-cell sharing e. se also corresponds to exactly one pair of opposite surface-oriented edges on the same 3-cell.

- A surface-oriented edge soe associates a surface edge se to a half-face sharing se. Each surface-oriented edge has a start surface-vertex.

- A surface-vertex sv associates vertex v to the surface of a 3-cell sharing v.

Figure 4.18 illustrates the entities in the HF data structure. Figure 4.18(a) shows all the faces, edges and vertices in a cell complex that is composed of two 3-cells. Figure 4.18(b) shows the half-faces, surface-edges and surface-vertices on the surface of each 3-cell of Figure 4.18(a). Figure 4.18(c) shows the surface-oriented edges bounding each half-face shown in Figure 4.18(b).



(a) (b) (c)

Figure 4.18: (a) A 3-complex with two cubic 3-cells, composed of 11 faces, 20 edges, and 12 vertices; (b) There are 12 half-faces, 24 surface-edges, and 16 surface vertices in the whole complex; (c) Surface-oriented edges of the two 3-cells

Besides these relations, two adjacency relations are encoded, namely, the one between each pair of half-faces belonging to the same face, and the one of the surface-oriented edges that correspond to the same edge in the same pair of half-faces.

A half-face is equivalent to a face-use in the RE data structure. A surface-oriented edge corresponds to the half-edge in the Half-Edge data structure, to the edge-use in the RE data structure and to the partial-edge in the PE data structure.

We can formalize the HF data structure in terms of topological relations as follows:

- For each face f, partial $R^*_{2,1}(f)$ relation, which encodes one edge bounding

face f;

- For each edge e, partial $R^*_{1,1}(e)$ relation, which encodes all the edges that share a face with edge e, radially ordered around e and relation $R_{1,2}$, which encodes all faces around an edge in the same radial order;

- For each vertex v, partial relation $R^*_{0,1}(v)$, which encodes one edge incident in vertex v.

All the relations are ordered, but the above formalization does not capture the association between faces and half-faces and thus the orientations on the faces. Note that the 3-cells are not represented explicitly in the HF data structure. The drawback is that no attribute can be attached to the 3-cells as a consequence. Also, the HF representation encodes the same relations as in the RE and PE data structures. On the other hand, unlike the RE and the PE data structures, the HF data structure cannot represent shapes with dangling edges or faces, as well as 3D shapes with non-manifold vertices and edges. As all representations which encode orientations by duplicating the basic entities, the HF data structure is quite verbose.

All topological relations at faces, edges and vertices can be retrieved in optimal time from the HF data structure, and the representation of surface entities allows retrieving the boundaries of the 3-cells even if these latter are not explicitly represented.

4.1.3.3 The Compact Half-Face (CHF) Data Structure

The *Compact Half-Face (CHF) data structure* [50] is a specialization of the HF data structure for representing manifold simplicial 3-complexes (usually called *tetrahedral meshes*). It is a multi-level data structure that encodes simplexes and the connectivity among them at different levels of detail. It is designed with four levels (0-3). We describe levels 0 to 2 in detail. Level 3 addresses the boundary information of the 3-manifold, whereby boundary cells are encoded. This level is not elaborated here as it is an application-specific feature of the data structure.

The entities in the CHF fall into two groups: the *basic entities* which are the tetrahedra, triangles, edges and vertices of a simplicial 3-complex (see Figure 4.19(a)), and the *surface entities* which are the half-faces and half-edges. Each half-face corresponds to an orientation of a triangle and belongs to at most one tetrahedron (see Figure 4.19(b)). Each half-face is bounded by a circular list of half-edges whose orientation is aligned with that of the half-face (see Figure 4.19(c)).

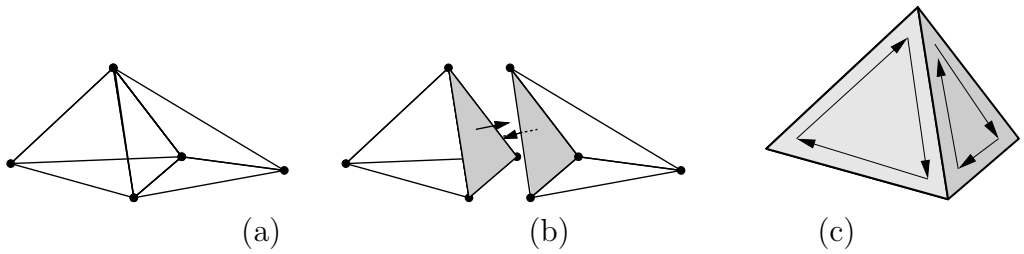


Figure 4.19: (a) A simplicial 3-complex with two tetrahedra sharing one triangle. There are two tetrahedra, seven triangles, nine edges and five vertices; (b) Two half-faces corresponding to the face shared between two tetrahedra (c) Half-edges on two half-faces of one tetrahedron.

Level 0 encodes vertices and tetrahedra as explicit entities. The topological relations explicitly encoded are $R_{3,0}(t)$ that is the vertices of each tetrahedron t. Half-faces and half-edges not explicitly encoded. Given a tetrahedron t, based on the encoded ordering of the vertices of t, the half-faces and half-edges of t can be combinatorially computed and expressed in terms of the order of their vertices. In navigation, only topological relation $R_{3,0}(t)$ can be retrieved in optimal time.

At level 1, both relations $R_{3,0}(t)$ and $R_{3,3}(t)$ are encoded for each tetrahedron t. The encoding of relation $R_{3,3}(t)$ provides the pairing information of the half-faces that belong to the same face. Based on the pairing information, mate half-edges can be computed combinatorially, and thus partial relation $R_{1,3}(e)$, is partially encoded for each edge e implicitly represented by its half-edges. The CHF data structure at level 1 supports the retrieval of all relations $R_{3,*}(t)$ at tetrahedra, and the retrieval of relations $R_{2,*}(f)$ at face f and $R_{1,*}(e)$ at edge e given that f and e are expressed as half-faces and half-edges of tetrahedra.

Level 2 explicitly encodes relations $R_{3,0}(t)$ and $R_{3,3}(t)$ for each tetrahedron t

and partial relations $R^*_{0,3}(v)$ which encodes an incident tetrahedron for each vertex v. In addition, the faces and edges are encoded explicitly for the purpose of attribute assignment. Each face f entity is mapped to one half-face sharing it. A half-edge defined by its vertices is mapped to one of its incident half-faces.

The CHF data structure at level 2 supports full topological navigation. Namely, the retrieval of tetrahedron-based relations $R_{3,3}$ and $R_{3,0}$, and face-based relations $R_{2,*}$ can be performed in constant time. The retrieval of vertex-based and edge-based co-boundary relations can be performed in time linear with respect to the size of the relations.

The amount of information encoded by the CHF data structure up to level 2 is $8n_3 + n_2 + n_1 + n_0$, where each of level 0 and level 1 contributes to $4n_3$, level 2 contributes to $n_2 + n_1 + n_0$. In total, the amount of information encoded is $O(8n_3 + n_2 + n_1 + n + 0)$.

4.1.3.4 Comparisons

We compare the above data structures in terms of their characteristics, their storage costs and efficiency in supporting topological navigation. Table 4.10 summarizes the comparison among the various data structures in terms of their domain, represented complex, and representation method.

We evaluate here the storage costs of the various data structures for simplicial 3-complexes except the HF data structure, which is represented into the CHF, and the specialization of the dimension-independent ones for the manifold and non-

Data Structure	Domain	Complexes	Method
HF	Manifold	Cell	Edge-based
CHF	Manifold	Simplicial	Edge-based
Facet-Edge	Manifold	Cell	Implicit

Table 4.10: Characteristics of data structures for 3-complexes

manifold domains.

In the case of manifold simplicial 3-complexes, the numbers of elements encoded in the data structures are evaluated to be:

- EIA: $8n_3 + n_0$
- IG : $8n_3 + 6n_2 + 4n_1$
- CHF: $8n_3 + n_2 + n_1 + n_0$
- FE: $4n_3 + 12n_2 + 2n_1$

A comparison of their storage costs is made experimentally based on five data sets of manifold simplicial 3-complexes shown in Table 4.11. The implicit FE representation encodes the largest number of topological elements, The incidence-based IG is more compact than the FE. The CHF at level 2 is more compact than the IG. The most compact is the adjacency-based EIA data structure.

Topological relations can be retrieved in optimal time from the HF, and the FE data structures.

Data set	n_0	n_1	n_2	n_3
Rings	2.52k	13.2k	18.8k	8.13k
Basket	1.21k	6.43k	9.22k	4.00k
Cylinder	1.31k	7.79k	11.6k	5.16k
Gargoyle	2.73k	14.7k	22.0k	10.0k
Torus	2.29k	15.4k	24.0k	10.9k

(a)

Data set	EIA	IG	CHF	FE
Rings	67.6k	231k	99.6k	285k
Basket	33.2k	113k	48.9k	139k
Cylinder	42.6k	142k	62.1k	176k
Gargoyle	82.7k	271k	119k	333k
Torus	89.2k	293k	129k	362k

(b)

Table 4.11: (a) Five manifold 3D simplicial data sets; (b) Storage cost of six data structures for the data sets in (a)

Data Structure	Boundary relations	Co-boundary relations	Adjacency relations
HF	Optimal	Optimal	Optimal
FE	Optimal	Optimal	Optimal

Table 4.12: Performances in retrieving topological relations from data structures for 3-complexes

The only data structure that can describe non-manifold simplicial 3-complexes is the Incidence Graph (IG).

4.2 Decomposition Approach to Shape Representation

Another way to represent non-manifold shapes consists of decomposing them into manifold, or nearly-manifold, components. The various decomposition approaches proposed in the literature try to realize in the discrete case a stratification

of the shape which has been defined for analytic sets (see [81]). In this Section, we focus on those approaches which have been developed as a basis of data structures for non-manifold shapes.

Selective Geometric Complexes (SGCs) [72] describe arbitrary-dimensional non-manifold objects through collections of mutually disjoint cells, which are defined as open subsets of d-manifolds. Thus, the cells can be either open, and not simply connected, and they form a stratification of the shape. In SGCs, cells and their neighbourhood information are encoded in a graph whose nodes represent the cells and whose arcs describe their incidence relations. This graph notion is similar to that of the Incidence Graph (see 4.1.1.2). SGC is designed for high-level geometric representation of shape, and thus is not equivalent to a combinatorial simplicial complex which is a topological notion. Each cell in SGC is the representation that requires minimum fragmentation consistent with explicit presence of boundary data and with connectedness of cell. The example shown in Figure 4.20(a) shows the SGC description of a cube, which uniquely captures its shape and boundary. Figure 4.20(b) shows the same cube described as a simplicial complex. The simplicial complex representation of the cube is not unique due to the non-uniqueness of the tetrahedralization. The example in Figure 4.20(c) shows the SGC description of two rectangular boxes sharing an edge.

Some techniques have been proposed in the literature for decomposing the boundary of regular non-manifold 3D shapes (the so-called *r-sets*) into manifolds [34, 40, 70]. The objective is to apply modeling tools developed for manifold shapes

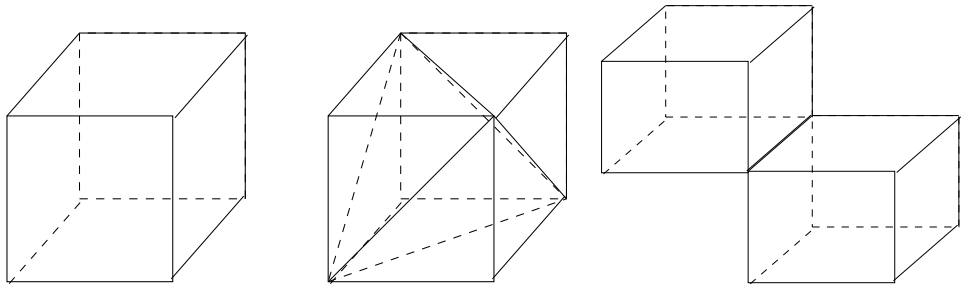


Figure 4.20: (a) In SGC, the minimal representation of a cube that takes into account its boundary data consists of 1 volume, 6 faces, 12 edges and 8 vertices; (b) The same cube when described as a minimal simplicial complex consists of 6 tetrahedra, 16 triangles, 22 edges and 8 vertices; (c) In SGC, the minimal representation of two rectangular boxes sharing an edges consists of 2 volumes, 12 faces, 23 edges and 14 vertices

(data structures and manipulation operators) to non-manifold ones. In [34], the result of the decomposition is represented as a graph in which the arcs describe non-manifold singularities. The approach has been applied for identifying form features in r-sets. In [70], a decomposition algorithm for a non-manifold object is presented which minimizes the number of duplications introduced by the decomposition process. In [40], the idea of cutting a non-manifold 2-complex into manifold pieces is exploited to develop compression algorithms. A cut-and-stitch technique is proposed for handling non-manifold cell 2-complexes. The cutting part of the cut-and-stitch technique decomposes the star of every non-manifold vertex in such a way that 2-cells (faces) that share manifold edges in the star are in the same component. Each such component is homeomorphic to a disc or to a half-disc. After cutting, the non-manifold edges in the original complex become boundary edges in the new complex. Also, multiple components may have resulted from cutting. The stitching part of

the cut-and-stitch technique merges selected edges that are created in the cutting phase.

4.2.1 Combinatorial Stratification

Pesco et al. [67] propose an approach inspired from stratification to represent non-manifold shapes. A cell 2-complex describing the boundary of a 3D non-manifold shape is decomposed into subcomplexes, which are the analogous of the strata (the components) in a stratification of an analytic set. They define a *combinatorial stratification* of a cell 2-complex Γ as a collection of k-dimensional connected combinatorial manifolds $S = \{M_1, \cdots, M_n\}$ $(k = 0, 1, 2)$ with or without boundary such that the union $\cup_i M_i$ gives Γ and the intersection between any two elements M_i and M_j in S is either empty of a sub-complex of both M_i and M_j. A combinatorial stratification is not necessarily unique. As an example, two valid stratifications of a simplicial 2-complex in Figure 4.21(a) are shown in Figures 4.21(b) and (c). The resulting set of strata and their connectivity provides a description of the original shape which is used as the basis for a data structure for non-manifold shapes discretized as cell 2-complexes, called the *Handle-Cell (HC) data structure*.

The *Handle-Cell (HC) data structure* consists of two sets of cells, namely the *global cells* and the *local cells*. The global cells, i.e., global vertices, global edges and global faces are the vertices, edges and faces of the given cell complex. The local cells are the cells that describe the strata. The strata are points, curves and surfaces. Curves are composed of curve-vertices and curve-edges. Surfaces are composed of

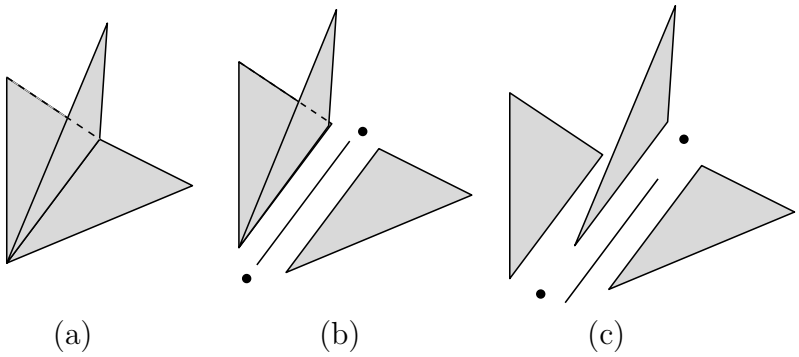


Figure 4.21: (a) A 2-complex that consists of three triangles sharing an edge; (b) and (c) two valid stratifications of (a) which result in different manifold components

surface-vertices, surface-edges, boundary-curves and surface-faces. Since strata are 2-complexes with a manifold domain, surfaces are represented through the Half-Edge data structure. This is conceptually similar to the representation of the surfaces of the 3-cells in the Handle-Face (HF) data structure (see Section 4.1.3.2). Curves are described as lists of edges connected by vertices. The connectivity among the strata is captured through the sharing of global vertices and global edges.

The HC data structure can be formalized in terms of topological relations as follows:

- For each face f: Relation $R^*_{2,1}(f)$ which consists of one edge on the boundary of f,
- For each edge e:
  - Relation $R_{1,2}(e)$, which consists of all faces incident in e;
  - Partial relation $R^*_{1,1}(e)$, ordered around edge e, so that both the $2i$-th element and the $(2i+1)$-element in this relation are on the i-th face of e;

– Relation $R_{1,0}(e)$, which consists of the extreme vertices of edge e;

- For each vertex v: Relation $R_{0,1}(v)$, which consists of the set of edges incident in a vertex v.

The HC data structure supports efficient topological navigation, as the incidence relations among q-cells and $(q-1)$-cells are fully encoded and the edge-based adjacency relations among edges in the 2-manifold strata are encoded. It encodes a large number of topological relations in order to support incremental shape construction in the non-manifold domain through a specific category of topology-modifying operators.

The Handle-Cell data structure is closely related to the Handle-Face data structure for manifold 3D cell complexes, and it is similar to the data structures for non-manifold 2-complexes, such as the Radial-Edge data structure (Section 4.1.2.2) and Partial-Edge data structure (Section 4.1.2.2). The primary difference between the HC representation and the latter group lies in the explicit description of the stratification encoded in the HC data structure.

4.2.2 Initial Quasi-Manifold Decomposition

A decomposition of a non-manifold shape into simpler parts can be obtained by splitting the shape at those elements (vertices, edges, faces, etc.) where singularities occur. In order to be effective, the decomposition process should remove as many singularities as possible, without introducing artificial, or arbitrary, "cuts" through manifold parts. Under these assumptions, a decomposition into manifold

components is possible, in general, only for 2-complexes. In three or higher dimensions, a decomposition into manifold components may need to introduce artificial cuts through the object. In six or higher dimensions, a decomposition into manifold components is not feasible in general, since the class of d-manifolds has been proven to be not decidable for $d \geq 6$ [62].

In [26], a decomposition of a non-manifold complex in arbitrary dimensions is proposed, which is *unique*, since it does not make any arbitrary choice in deciding where the object has to be decomposed, and *natural*, since it removes singularities by splitting the complex at non-manifold simplexes only. Such a decomposition is known as the *standard* decomposition of the original complex. The components of such decomposition, called *Initial Quasi-Manifolds (IQMs)*, admit a local characterization in terms of combinatorial properties around each vertex. A d-dimensional IQM is a simplicial d-complex Σ in which all top simplexes have dimension d and such that the star of each vertex of Σ is $(d-1)$-connected, i.e., can be traversed by moving between adjacent d-simplexes through their common $(d-1)$-face. If an IQM is embeddable in R^d where $d \geq 3$, it must be a pseudo-manifold complex (i.e., a $(d-1)$-connected complex in which every $(d-1)$-simplex is on the boundary of one, or two d-simplexes).

The properties of the IQM decomposition makes it a good basis for defining representation for non-manifold simplicial shapes.

The *Initial Quasi-Manifold (IQM)* data structure [26] is built on these properties. The IQM data structure describes the decomposition of a simplicial complex into k-dimensional initial quasi-manifold components.

The basis of the IQM data structure are an extended indexed data structure with adjacencies to encode each IQM component and a hypergraph describing how the components are connected together in the decomposition. An h-dimensional IQM can be effectively described by an extended indexed data structure with adjacencies, since the star of each vertex in the IQM can be traversed by using relations $R^*_{0,h}$ plus $R_{h,h}$.

The connection among components is described through the vertices bounding the k-simplexes, which are shared by more than one IQM component. A vertex v of Σ, which is shared by several IQM components, is called a *split vertex*. The copy of split vertex v in a component C_i, to which vertex v belongs, is denoted as v_i and it is called a *vertex copy*. The relations among the components in an IQM decomposition of a complex described by the split vertices is represented as a hypergraph H, in which the nodes correspond to IQM components and each hyperarc corresponds to a split vertex v and it connects all components C_i sharing v. In the example shown in Figure 4.22, vertex v in Figure 4.22(a) is split into vertices v_1, v_2 and v_3 in the decomposition shown in Figure 4.22(b). In the hypergraph shown in Figure 4.22(c), a hyperarc associates v with the three components C_1, C_2 and C_3 through the three vertex copies.

The hypergraph is encoded in the following data structure:

- for each component C_i: a reference to the EIA data structure describing C_i;
- for each hyperarc: the corresponding split vertex v and the vertex copies of v;

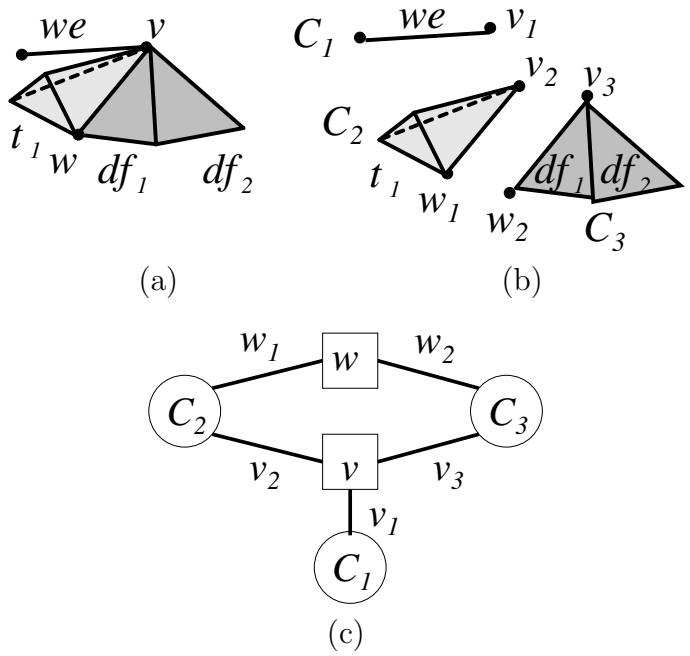


Figure 4.22: IQM decomposition of a complex

- for every vertex copy v_i corresponding to split vertex v:
  - the component containing v_i;
  - a reference to its hyperarc, i.e., v.

The hypergraph supports a vertex-based traversal among components connected through the same hyperarc. Given a vertex copy v_i from any component C_i, we can follow the reference to its hyperarc and find all other vertex copies v_j connected with v, as well as all other components sharing v.

The IQM decomposition approach taken by the IQM data structure differs from the stratification approach of the HC representation in the following two aspects. First, the HC representation is based on a decomposition for cell 2-complexes, approach while the IQM decomposition is dimension-independent. Also, unlike the

combinatorial stratification, an IQM decomposition is unique.

4.3 Updates on Representations

The algorithms for extracting topological relations from a complex are the basis for performing update operations on the complex. There is a vast literature on update and construction operators and a review of such operators goes beyond the scope of this survey. Updating a manifold 2D cell complex describing the boundary of a 3D object has been extensively studied in the solid modeling literature for more than twenty years, and several proposals exist for primitive update operators which maintain the validity of Euler's formula, the so-called *Euler operators* (see, for instance, [57]). Such operators have also been defined for non-manifold 2-complexes (see, for instance, [51, 56, 82, 80]) by considering different variants of Euler's formula. In both the manifold and non-manifold cases, the effect of any other operation on the complex is then expressed as a suitable sequence of Euler operators. Higher-level operators based on the Handle-Body theory have been proposed [67]. The handle-body theory studies the topological changes generated by attaching handles to a manifold without boundary. Handle-body operators change also the topological type of the domain of the complex.

Primitives for updating simplicial complexes have been proposed in the literature, mainly for triangle and tetrahedral meshes (see, for instance, [23, 76]). Some of them do not affect the topology of the domain of the complex, but only the combinatorial structure of the subdivision (see, for instance, [76]). The most common

update operators for simplicial complexes are those applied in mesh simplification algorithms. The problem of simplification of simplicial complexes has been extensively studied in computer graphics for triangle meshes (see, e.g., [23, 37, 55], for a survey), and, more recently, some algorithms have been developed for tetrahedral meshes. These approaches are based on contracting one edge to one of its extreme vertices or to a new vertex [8, 9, 39, 43, 69, 75]. In [68], the problem of applying a vertex-pair contraction (which consists of contracting a pair of vertices to a new vertex) on a d-dimensional simplicial complex is addressed. The complex is represented as an Incidence Graph. In [25], algorithms for performing vertex-pair contraction and its inverse, vertex expansion, have been developed on a simplicial 2-complex described as a TS data structure [25].

Specific simplification algorithms have been proposed for finite element mesh generation from CAD models [11, 35, 77, 78]. In this case, the idealization of a simplicial complex is performed through a set of geometrical and topological transformations [78], involving detail removal operators (e.g., vertex removal and remeshing), which change the shape of a component without modifying its topology, topological detail removal operators (e.g., hole removal), which change the topology of the complex while preserving the dimension of the part, and dimension-reduction operators, which reduce the dimension of a part, by contracting, for instance, a tubular part to a wire.

4.4 Summary

In this Chapter, We reviewed, analyzed and compared data structures for simplicial and cell complexes, with a special emphasis on data structures for simplicial complexes. We classified the data structures in each group according to the basic kinds of the topological entities they represent. We described each data structure in terms of the entities and topological relations it encodes, and we evaluated it based on its expressive power, on its storage cost, on the efficiency in supporting navigation inside the complex. We also discuss a decomposition approach to modeling non-manifold shapes, which has led to powerful and highly scalable representations. This work has been published first at a preliminary level for simplicial shapes in [15], then as a thorough state-of-the-art report in [19] and was invited for publication as [20].

Based on this survey, we observe that majority of existing work falls in the category for representing 2D complexes. As discussed in Section 4.1.3, there is only one data structure, namely the dimension-independent Incidence Graph (IG), that can represent non-manifold 3D complexes. This absence of representations is due to a lack of understanding of the non-manifold properties in a 3-complex. When employed on manifold simplicial complexes, we have shown that the IG has a large storage overhead compared with the extended Indexed Data Structure with Adjacencies (EIA), a data structure specialized for such complexes. On the other hand, the IG encodes all cells explicitly, which is necessary for some applications. Thus, there is a lack of cost-efficient representation with the same express power as

the IG but specialized for non-manifold simplicial complexes. Therefore, this work begins with research on representations for non-manifold simplicial d-complexes with primary interest in the case for $d = 3$.

Chapter 5

Two Dimension-Independent Data Structures for Non-manifold Shapes

In the design of topological data structures, the primary issue is their representation power. The issues of the storage cost and the navigation efficiency are next in significance. Storage cost depends on the amount of topological information explicitly encoded in the specific data structure, which in turn is dependent on the operational efficiency that is needed of the data structure.

Basic geometric modeling operations, such as Boolean operations, as well as algorithms for manipulating and updating an object described by a simplicial complex (e.g, simplification algorithms) require being able to extract the simplexes on the boundary of a given simplex, or belonging to its star, or those adjacent to it. This requires efficient algorithms for retrieving the simplexes, which are in some topological relation with a given simplex, from the data structure encoding a simplicial complex. The objective is to have algorithms which require only examining the neighborhood of the given simplex, and, thus, exhibit a time complexity linear in the number of simplexes in such neighborhood.

We have shown in our review of the literature that the only existing data structure that has the representation power to describe non-manifold simplicial complexes of dimension three or above, is the Incidence Graph (IG) (see Section 4.1.1.2). The IG philosophy is to encode all the cells and a selected subset of the incidence relations so that it is possible to perform navigation of the whole complex efficiently. The IG encodes all simplexes explicitly, which makes it suitable for finite element analysis (FEA) applications, in which it is necessary to assign attributes (such as geometric and thermal properties) to the vertices, edges and faces of a 3D model. However, our comparison of the IG with the extended Indexed Data Structure with Adjacencies (EIA) shows that the IG does not scale well to the manifold case. In simplicial complexes, there is a constant number of boundary relations associated with each simplex. This property enables a data structure design which gives significant reduction to the storage cost without trading off the efficiency of navigation.

We are, thus, interested in a data structure for simplicial complexes, which encodes all simplexes, supports efficient retrieval of topological relation and is cost efficient. To investigate this, we study the topological relations encoded by an explicit data structure for a simplicial complex through a directed graph.

In addition, for some applications, it is necessary to perform modifications on the model. An example is the idealization process in CAD tools, in which a model is abstracted into simple forms through the removal of details, such as closing through holes and reducing the dimension of selected parts of the model [52]. These operations are implemented as a series of elementary topology-modifying operations. Vertex-pair contraction, which consists of contracting a pair of vertices

to a single one, is the basic operation used for modifying the topological type of a complex. Therefore, in our design of new data structures, we also consider the support for modifications for the construction of a Non-manifold Multi-Tesselation model (NMT), which is a new paradigm of multi-resolution modeling.

In Section 5.1, we discuss the directed graph as a tool for visualizing and analyzing an explicit data structure. In Sections 5.2 and 5.3, we present two proposals of cost-efficient dimension-independent data structures, namely the Simplified Incidence Graph (published in [12]) and the Incidence Simplicial Data Structure (in preparation to be published). In Section 5.4 an evaluation of these two proposals are made by a comparison with existing data structures. Section 5.5 discusses the application of one of the proposed data structures in a technique of constructing a multi-resolution model, called Non-manifold Multi-Tesselation (NMT). Work on this application has been published in [17].

5.1 Directed Graph Representation of Explicit Data Structures Encoding Simplicial Complex

The simplicial complex can be described as a directed graph $G =< N, A >$ in which the nodes N represent simplexes and are arranged in a hierarchy in the order of their dimensions, and the directed arcs A represent the topological relations among the simplexes. A directed arc that points from node m at level i to node n at level j (for $j > i$) indicates that n is a coface of m. The inverse arc from n to m indicates that m is a face of n. A bi-directed arc between nodes n and n' at level i

indicates that n and n' are $(i-1)$-adjacent.

The topological relations captured by Incidence Graph (IG) [32] are all those boundary and co-boundary relations between simplexes that differ by one dimension. These relations are represented by the directed edges between nodes at two adjacent levels. Figure 5.1 shows a simplicial complex formed by four tetrahedra and two dangling-faces. Figures 5.2 and 5.3 show, respectively, the boundary and co-boundary relations encoded by the IG. Observe that the difference between the two diagrams is in the direction of the arcs. The co-boundary relations encoded are the inverse of the boundary relations. The size of $\bigcup(ST)$ in the IG is the same as the size of all the boundary relations encoded by the IG.

It can be observed that, if a data structure encodes all boundary relations between simplexes differing by one dimension (such as the Incidence Graph), then it is necessary to encode only a selected subset of relations in the star of each simplex in order to provide navigation efficiency. Two proposals are made, which are presented in Sections 5.2 and 5.3 respectively.

5.2 Simplified Incidence Graph[1]

In this Section, we present the Simplified Incidence Graph which specializes the Incidence Graph for simplicial complexes. We discuss the design of this data struc-

[1]Originally published in [12] Copyright ©2004 Eurgraphics.

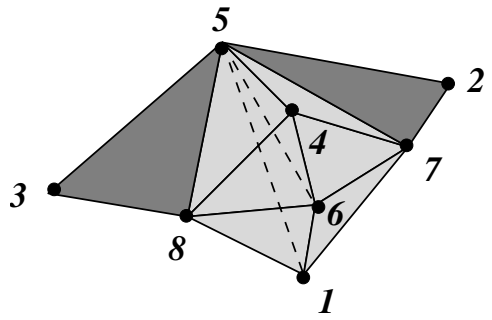


Figure 5.1: Example of a 3D simplicial complex that consists of four face-adjacent tetrahedra shown in light shaded gray, and two dangling triangles shown in dark gray. The vertices are labeled with numbers 1 to 8.

ture, evaluate its cost efficiency and discuss its scalability to the manifold domain. Lastly, we discuss how asymemtric vertex-pair contraction may be implemented on this data structure.

5.2.1 Design of the Data Structure

The *Simplified Incidence Graph (SIG)* is a representation for a d-dimensional Euclidean simplicial complex embedded in the n-dimensional Euclidean space, with $d \leq n$. When $d = n$, every $(d-1)$-simplex is shared by at most two d-simplexes, since any d-dimensional simplicial complex embedded in the d-dimensional Euclidean space is a pseudo-manifold. Given a d-dimensional simplicial complex Σ, the SIG encodes all p-simplexes for $p = 0, 1, \ldots d$ in Σ, and

- for each p-simplex σ, where $0 < p \leq d$, it encodes boundary relations $R_{p,p-1}(\sigma)$,
- for each p-simplex σ, where $0 \leq p < d$, partial co-boundary relations $R^*_{p,g}(\sigma)$ (where $g > p$), which consists of one arbitrarily-selected top g-simplex for each

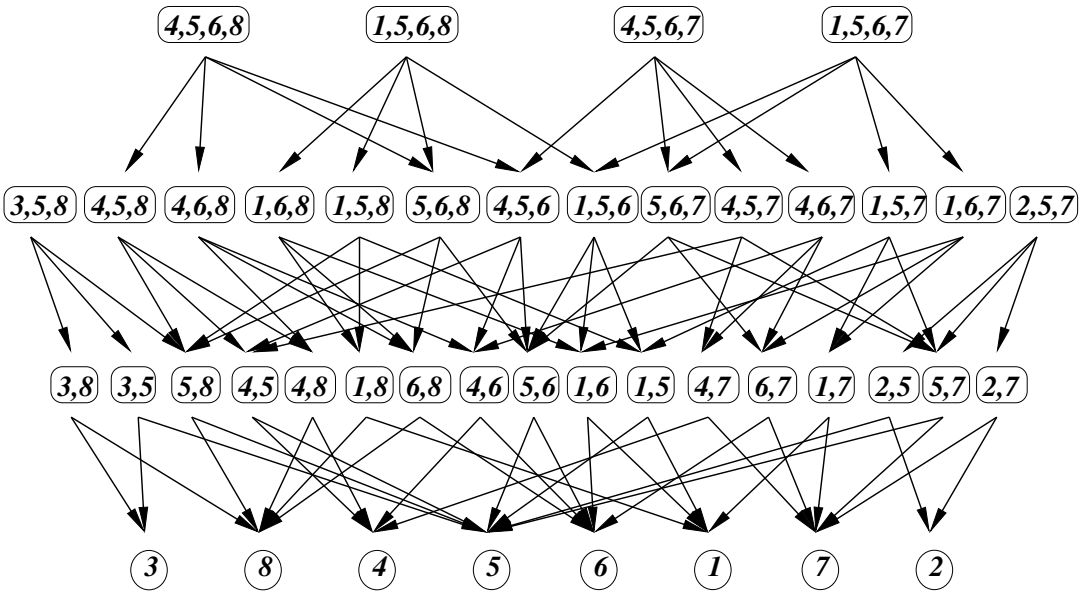


Figure 5.2: The directed graph showing the boundary relations encoded by the IG for the example shown in Figure 5.1. Simplexes are identified by the indices of the vertices spanning them. For example, the node $(4, 5, 6, 8)$ denotes the tetrahedron that is incident at vertices 4, 5, 6 and 8.

$(g - p - 2)$-connected regular $(g - p - 1)$-dimensional component in the link of σ.

Note that partial co-boundary relation $R^*_{d-1,d}(\sigma)$ is the same as co-boundary relation $R_{d-1,d}(\sigma)$. Moreover, when the domain is a manifold, all partial co-boundary relations are empty with the exception of $R^*_{p,d}(\sigma)$. In this case, co-boundary relation $R^*_{p,d}(\sigma)$ encodes one or two d-simplexes incident at σ when $p = d-1$, or just one d-simplex incident at σ when $p < d-1$.

Figure 5.4(a) shows an example of the encoding of a vertex of a 2-complex in a SIG. Two partial co-boundary relations are defined at v, namely, $R^*_{0,g}(v)$, for $g = 1, 2$. Relation $R^*_{0,1}(v) = \{e\}$ and relation $R^*_{0,2}(v) = \{f_1, f_2\}$. Figure 5.4(b)

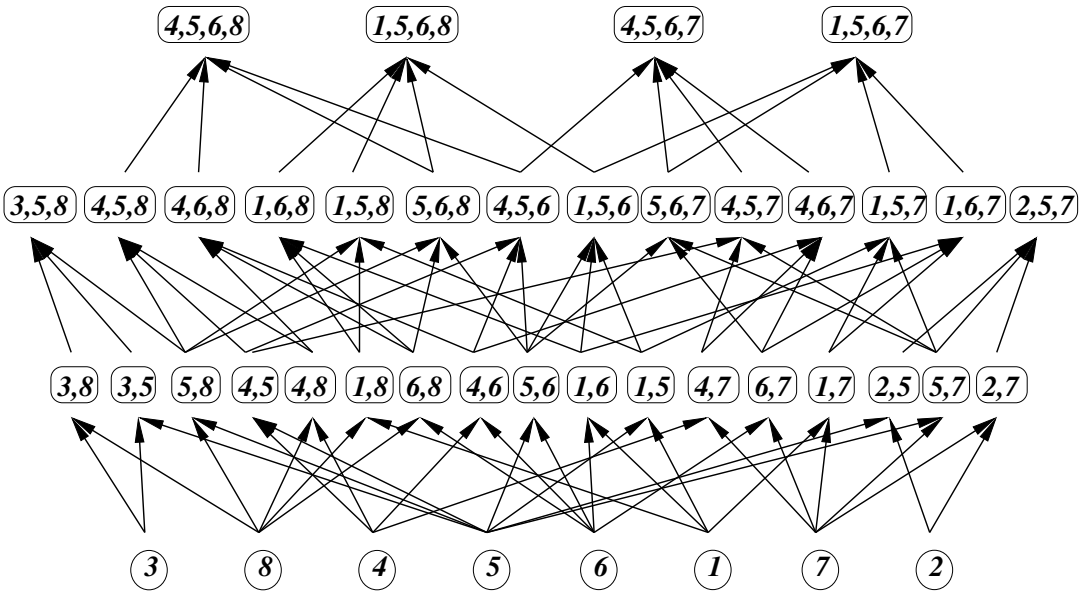


Figure 5.3: The directed graph showing the co-boundary relations encoded by the IG for the example shown in Figure 5.1. Simplexes are identified by the indices of the vertices spanning them.

shows an example of the encoding of an edge of a 2-complex in a SIG. The partial co-boundary relation defined at e is $R^*_{0,1}(e)$, which consists of $\{f_1, f_2\}$.

Figure 5.5(a) shows an example of the encoding of a vertex of a 3-complex in a SIG. The restricted star of v $st(v)$ consists of edge we, of triangles df_1 and df_2, of the edges of df_1 and df_2 which are incident at v, of tetrahedra t_1, t_2 and t_3, together with all the faces and edges of t_1, t_2 and t_3 which are incident at v. Three partial co-boundary relations are defined at v, namely, $R^*_{0,g}(v)$, for $g = 1, 2, 3$. Relation $R^*_{0,1}(v) = \{we\}$, relation $R^*_{0,2}(v) = \{df_1\}$, and relation $R^*_{0,3}(v) = \{t_1, t_2\}$.

Figure 5.5(b) shows an example of the encoding of an edge of a 3-complex in the SIG. The restricted star $st(e)$ of edge e is composed of triangles df_1 and df_2, of tetrahedra t_1, t_2, and t_3, and their faces which are incident at e. Since $e \notin st(e)$, df_1

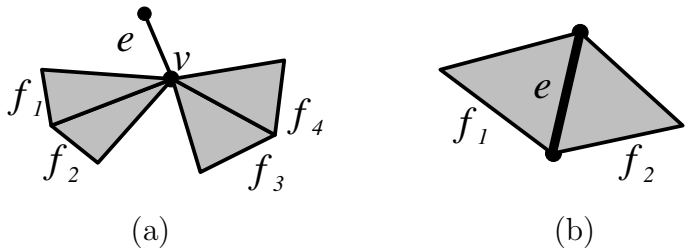


Figure 5.4: Two examples showing the relations stored at a vertex (a) and at an edge (b) in a SIG describing a 2-complex. In (a), the top simplexes incident at vertex v are triangles f_1, f_2, f_3 and f_4, and edge e. In (b), the top simplexes incdient at edge e are triangles f_1 and f_2.

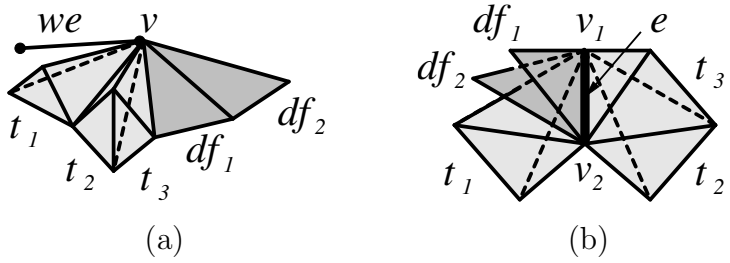


Figure 5.5: Two examples showing the relations stored at a vertex (a) and at an edge (b) in a SIG describing a 3-complex. In (a), the top simplexes incidnet at vertex v are tetrahedra t_1, t_2 and t_3, triangles df_1 and df_2, and edge we. In (b), the top simplexes incident at edge $e = \{v_1, v_2\}$ are tetrahedra t_1, t_2 and t_3, and triangles df_1 and df_2.

and df_2 are not 1-connected in $st(e)$, and, thus, they form two separate 0-connected 1-components in $st(e)$. Boundary relation $R_{1,0}(e)$, and the two partial co-boundary relations $R^*_{1,g}(e)$, for $g = 2, 3$, are stored at edge e. Boundary relation $R^*_{1,0}(e)$ consists of the set of extreme vertices of e, namely $\{v_1, v_2\}$. Partial co-boundary relations $R^*_{1,2}(e)$ and $R^*_{1,3}(e)$ consist of $\{df_1, df_2\}$ and $\{t_1, t_2\}$, respectively.

The directed graph that describes the boundary relations encoded by the SIG is the same as that for the IG, whereby edges exist between adjacent layers of nodes.

In the directed graph that describes the partial co-boundary relations encoded by the SIG, the end nodes of each directed edge is a top simplex. Consider the example shown in Figure 5.1, the topological relations encoded by the SIG are shown in the form of directed graphs in Figures 5.6 and 5.7. Note that the diagram in Figure 5.6 is the same as that in Figure 5.2. It can be observed that all the arcs in the diagram in Figure 5.7 end at top simplexes.

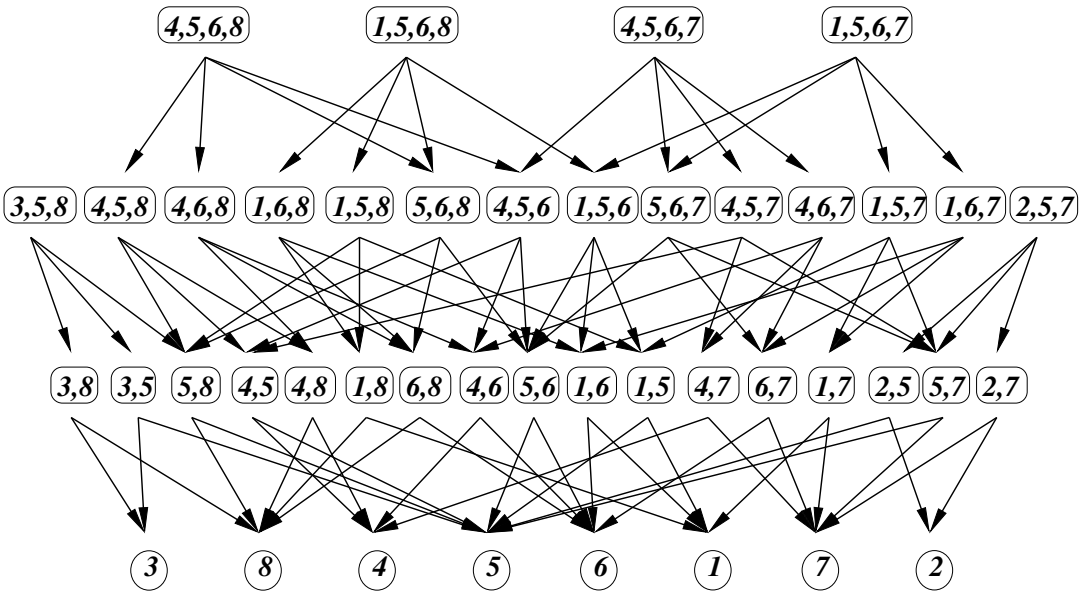


Figure 5.6: The directed graph showing the boundary relations encoded by the SIG for the example shown in Figure 5.1. Nodes representing simplexes are identified by the indices spanning them.

5.2.2 Implementation and Storage Cost

In this subsection, we describe the implementation of the SIG, and discuss the storage cost for this implementation. For simplicity, in our current implementation,

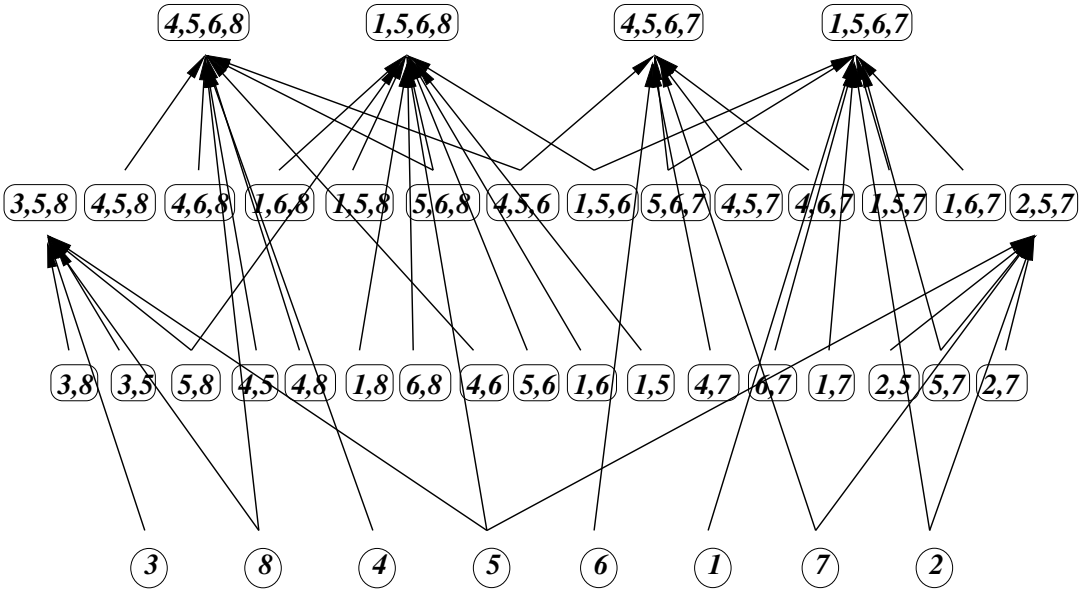


Figure 5.7: The directed graph showing the partial co-boundary relations encoded by the SIG for the example shown in Figure 5.1

we use one integer to index a simplex.

All simplexes are stored in ascending order of their dimensions. Each simplex has a unique index. A lookup-table is used to encode the starting and ending indices of the simplexes for each dimension. For each simplex, we also store a one-bit flag that is used by the navigation algorithms to mark a simplex as visited during traversal, and reset after traversal is completed.

For each p-simplex σ, with $0 < p \leq d$, boundary relation $R_{p,p-1}(\sigma)$ is stored in a fix-sized array, each element of which is an index to a simplex on the boundary of σ. For each p-simplex σ, with $0 \leq pr < d$, partial co-boundary relations $R^*_{p,g}(\sigma)$ for $p < g \leq d$ are stored, in decreasing order of g, in a variable-sized array. Each entry of the array consists of the index of a simplex containing σ in its boundary. The end of the array is marked by a stop code. An integer pointer is associated with simplex σ,

which is the starting index of the variable-sized arrays of the co-boundary relations. In the manifold case, for all p-simplexes σ, with $p < d-1$, only relation $R^*_{p,d}(\sigma)$ exists. Thus, the integer pointer associated with σ references directly the d-simplex in $R^*_{p,d}(\sigma)$ relation. A flag is used to indicate whether the manifold condition holds at a p-simplex, when $p < d-1$.

We denote with n_p, for $0 \leq p \leq d$, the number of p-simplexes in a simplicial complex Σ, with $\kappa^g(\sigma)$, for $dim(\sigma) < g \leq d$, the total number of $(g-p-2)$-connected regular $(g-p-1)$-dimensional components in the link of a simplex σ in Σ, and with $\kappa^g_p = \sum_{dim(\sigma)=p} \kappa^g(\sigma)$, for $0 \leq p < d$, and $g > p$, the total number of $(g-p-2)$-connected regular $(g-p-1)$-dimensional components summed over the links of all the p-simplexes in Σ.

The lookup-table, that stores the starting and ending indices for the simplexes in each dimension, requires $d+1$ integers. A total of $\sum_{0 \leq p \leq d} n_p$ bits is needed for the flag used for navigation. The space use for all boundary relations $R_{p,p-1}$ for $0 < p \leq d$ is equal to $\sum_{0<p\leq d}(p+1)n_p$ integers.

The storage space required for encoding the partial co-boundary relations can be evaluated as follows. Each p-simplex, $0 \leq p < d$, has a link to the partial co-boundary relations $R^*_{p,g}$ associated with it. This requires $\sum_{0 \leq p < d} n_p$ integers in total. In addition, the flag, that indicates whether the manifold condition holds at a simplex, requires one bit for each simplex, and, thus, $\sum_{0 \leq p < d} n_p$ bits in total. The total space use for all the variable-sized arrays that store the partial co-boundary relations $R^*_{p,g}$ is equal to $\sum_{0 \leq p < d} \sum_{p < g \leq d} \kappa^g_p + \sum_{0 \leq p < d} n_p$ integers, where the first term accounts for the indices of the simplexes and the second term $\sum_{0 \leq p < d} n_p$ accounts

for the stop codes.

Thus, the total space used for encoding all topological relations (i.e., both boundary and partial co-boundary relations) is equal to $\sum_{0\leq p<d}\sum_{p<g\leq d}\kappa_p^g + 2\sum_{0\leq p<d} n_p + \sum_{0<p\leq d}(p+1)\,n_p$ integers and $\sum_{0\leq p<d} n_p$ bits.

If Σ is a manifold complex, the variable-sized arrays are not used for the partial co-boundary relations associated with the p-simplexes, when $0 \leq p < d-1$. For the $(d-1)$-simplexes, the variable-sized arrays are still used because each $(d-1)$-dimensional simplex may be shared by either one or two d-simplexes. The space used for encoding the partial co-boundary relations thus becomes $\sum_{0\leq p<d} n_p + \kappa_{d-1}^d + n_{d-1}$ integers and $\sum_{0\leq p<d} n_p$ bits. Also, $\kappa_{d-1}^d = 2n_{d-1}$. The space used for the simplexes and the boundary relations is the same as in the non-manifold case. Thus, the overhead with respect to a simplified incidence graph specific for a d-complex with a manifold domain is equal to $\sum_{0\leq p<d} n_p$ bits + n_{d-1} integers.

5.2.3 Building a Simplified Incidence Graph

The Simplified Incidence Graph representation can be constructed from the Incidence Graph. The approach is to identify the $(q-1)$-connected q-components in the star of each simplex σ in the complex Σ. The construction algorithm consist of the following steps:

1. Find the set S_q of all the top q-simplexes in $st(\sigma)$

2. Partition the set S_q into $\{S_q^1, S_q^2, \cdots, S_q^j\}$, such that γ belongs to S_q^i if and only if either γ shares no $(q-1)$-face with any top simplex in S_q or γ shares

a $(q-1)$-face with some other simplex γ' in S_q^i.

3. Select one top k-simplex from each set S_q^i to be the representative of in partial co-boundary $R_{p,q}^*(\sigma)$

5.2.4 Topological Navigation in the SIG

Navigation in a complex requires retrieving topological relations. The retrieval of topological relations is also the basis for performing simplification operations on a simplicial complex. The SIG supports a simple recursive strategy to retrieve all topological boundary relations. Boundary relation $R_{p,q}(\sigma)$ for a p-simplex σ is retrieved by cascading the retrieval of boundary relations $R_{p,p-1}, R_{p-1,p-2}, \cdots, R_{q+1,q}$ on σ and its faces.

The general strategy for retrieving co-boundary relations at a simplex σ consists of performing a traversal of the star of σ, and then retrieving the boundary relations of the top simplexes in the star of σ. The star of σ is retrieved by traversing all the components belonging to it. The traversal of an h-dimensional component starts with a representative in partial co-boundary $R_{p,h}^*(\sigma)$. It visits each h-simplex τ and all its $(h-1)$-adjacent simplexes incident at σ, and it terminates when all h-simplexes in the component are visited. The strategy for retrieving adjacency relations at σ consists of retrieving the co-boundary relations for the simplexes that are in the boundary relation of σ.

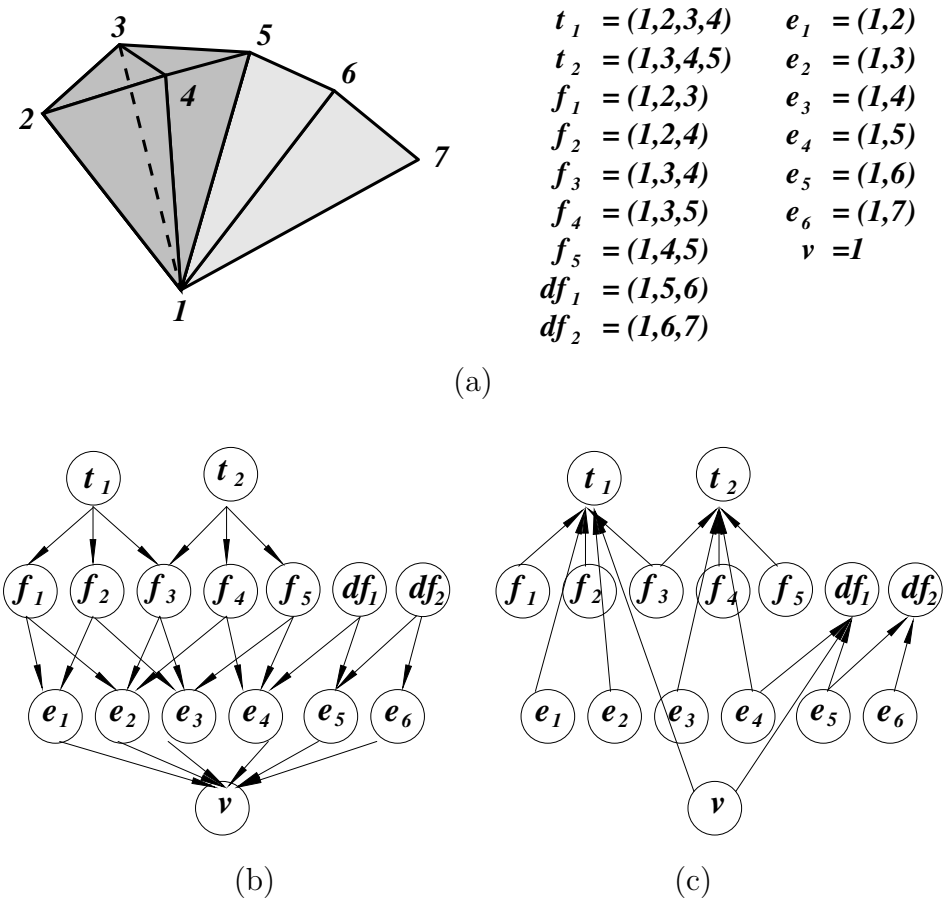


Figure 5.8: Example of retrieving $R_{0,1}(v)$ from the SIG through a traversal of the star of vertex v using boundary and partial co-boundary relations: (a) the star $st(v)$ of a vertex v, vertices are labeled by their indices and the simplexes are defined by the indices of the vertices that span them; (b) boundary relations encoded by the SIG among simplexes in $st(v)$; (c) partial co-boundary relations encoded by the SIG among simplexes in $st(v)$ (part 1)

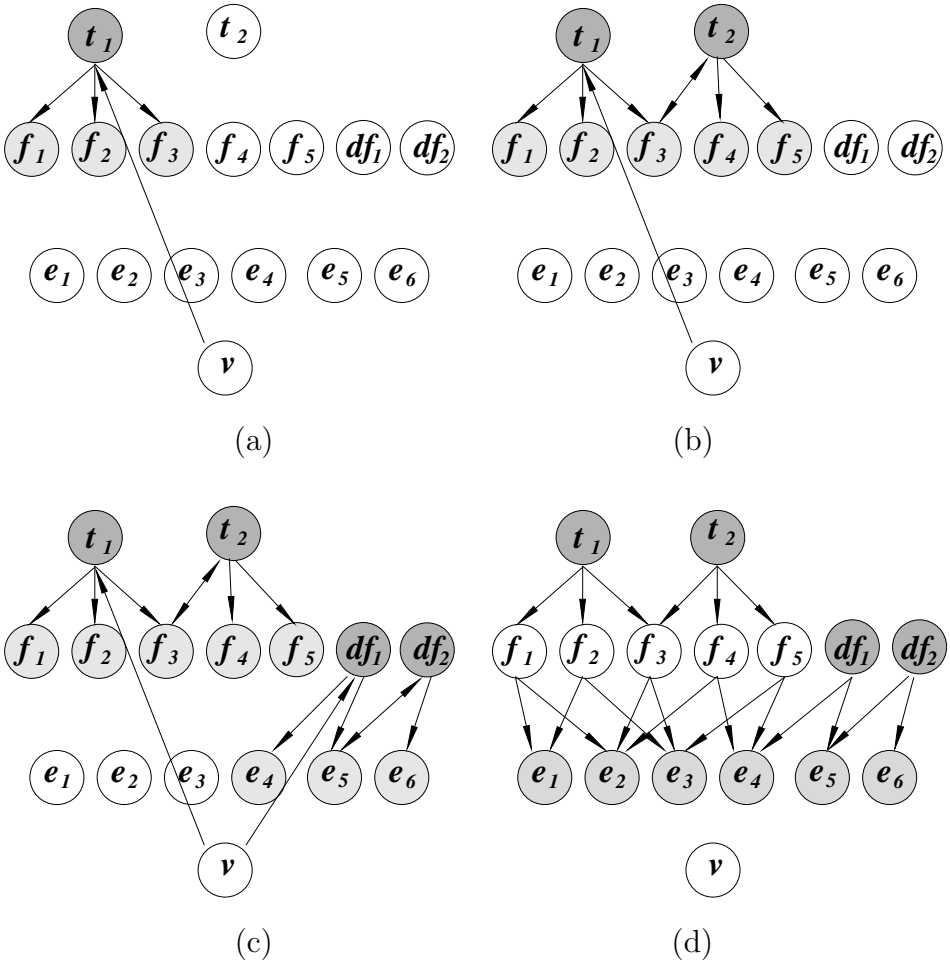


Figure 5.9: Example of retrieving $R_{0,1}(v)$ from the SIG through a traversal of the star of vertex v using boundary and partial co-boundary relations encoded by the SIG: (a) to (d) are four stages of the traversal of $st(v)$ (part 2)

We illustrate the retrieval of relation $R_{0,1}(v)$ for a vertex v in the example shown in Figures 5.8(a)-(c). The traversal of the star of v is shown in Figures 5.9. Figures 5.9(a) to (d) show four stages of the traversal of the star of vertex v (from the example of Figure 5.8(a)) for retrieving $R_{0,1}(v)$. Figure 5.9(a) shows that the traversal starts from v at which the partial co-boundary relations encoded are $R^*_{0,3}(v) = \{t_1\}$ and $R^*_{0,2}(v) = \{df_1\}$. Following relation $R^*_{0,2}(v) = \{t_1\}$, tetrahedron

t_1 is visited. The boundary relation $R_{3,2}(t_1)$ is examined and faces f_1, f_2 and f_3 are found to be incident at v. In Figure 5.9(b), the partial co-boundary relations of the faces f_1, f_2 and f_3 have been examined. The partial co-boundary relation of face f_3 leads to tetrahedron t_2. The faces of t_2 are examined, and they lead to no new tetrahedra. So the whole 3D component associated with t_1 in the star of v is visited. Figure 5.9(c) shows that the 2D component associated with df_1 in $R^*_{0,2}(v) = \{df_1\}$ is examined likewise. Figure 5.9(d) shows that the edges incident at v are retrieved from all the top simplexes in the star of v.

The time complexity for the retrieval of the various relations is summarized in Table 5.1. The time complexity for the retrieval of boundary relations is constant for all dimensions, while that for retrieving co-boundary relations depends on the dimension of the complex. Retrieving co-boundary relations $R_{p,q}(\sigma)$ requires time linear in the number of top simplexes in the restricted star $st(\sigma)$ of σ. For simplicial 2- and 3-complexes, the number of top simplexes is linear in the number of q-simplexes in $st(\sigma)$, as a consequence of Euler's formula (see the description of the properties of non-manifold 3D shapes under Section 3.2). This is no longer true for higher dimensions.

5.2.5 Vertex-Pair Contraction on the SIG

In many applications of shape modeling, it is necessary not only to have an efficient representation that captures topological information within the shape, but also to have tools for updating this representation to preserve its topological intigrity

Dimension d	Boundary $R_{p,q}(\sigma)$	Co-boundary $R_{p,q}(\sigma)$	Adjacency $R_{p,p}(\sigma)$
2 and 3	constant	linear in terms of p-simplexes in $R_{p,q}(\sigma)$	linear in terms of p-simplexes in $R_{p,p}(\sigma)$
4 and above	constant	linear in terms of top simplexes in $st(\sigma)$	linear in terms of top simplexes in $st(\gamma)$, where $\gamma \in R_{p,p-q}(\sigma)$

Table 5.1: Time complexity of the retrieval strategies for topological relations of a p-simplex σ

as the shape is modified. One elementary mesh update operator is the Vertex-Pair Contraction (VPC) operator. In Section 3.3, we have formulated the VPC operation and characterized its effect when two vertices v_1 and v_2 are merged into one vertex. In the assymetric variation of VPC, vertex v_2 is merged to vertex v_1 and no new vertex is created. In this Section, we present an algorithm for performing asymmetric vertex-pair contraction on a simplicial complex encoded as a SIG. We describe how the entities and topological relations encoded in the SIG are updated as an effect of such operation.

Let Σ be a simplicial complex. Let $st(v_1)$ and $st(v_2)$ denote the stars of v_1 and v_2, and let $lk(v_1)$ and $lk(v_2)$ denote their links. Let the complex resulted from the assymetric VPC be denoted by Σ_R, and let $st_R(v_1)$ and $lk_R(v_1)$ denote, respectively, the resultant star and link of v_1.

The procedure of performing vertex-pair contraction on the SIG is described in the following four subsections. In each subsection, we describe how the entities and topological relations encoded in the SIG are updated.

5.2.5.1 Labeling Simplexes

The first step of the algorithm consists of labeling the simplexes incident at v_1 and v_2 and in constructing the transformation map F. For each p-simplex $\sigma \in \Sigma$, we define two labels $l_1(\sigma)$ and $l_2(\sigma)$ as follows:

- $l_i(\sigma) = 1$ if and only if $\sigma \in st(v_i)$, $i = 1, 2$
- $l_i(\sigma) = -1$ if and only if $\sigma \in lk(v_i)$, $i = 1, 2$
- $l_i(\sigma) = 0$, $i = 1, 2$, otherwise

In what follows, we write l_i for short whenever the simplex being addressed is clear. Figures 5.10(a) and (b) give examples of labeling of the simplexes in the star and link of v_1 and v_2. The example in Figure 5.10(a) illustrates the case in which the stars of v_1 and of v_2 have a non-empty intersection, i.e., there is an edge connecting v_1 and v_2. The example in Figure 5.10(b) illustrates the case in which the intersection of the two stars is empty, because there is no edge connecting v_1 and v_2.

All information necessary for updating the SIG can be deduced from the labels of each p-simplex in Σ and of its $(p-1)$-faces:

- Any p-simplex with labels $(0, 0)$, $(-1, 0)$, $(0, -1)$ and $(-1, -1)$ is not affected, since it is not incident at v_1 or v_2.
- Any p-simplex labeled $(1, 0)$ or $(1, -1)$ will remain the same in the reduced complex.

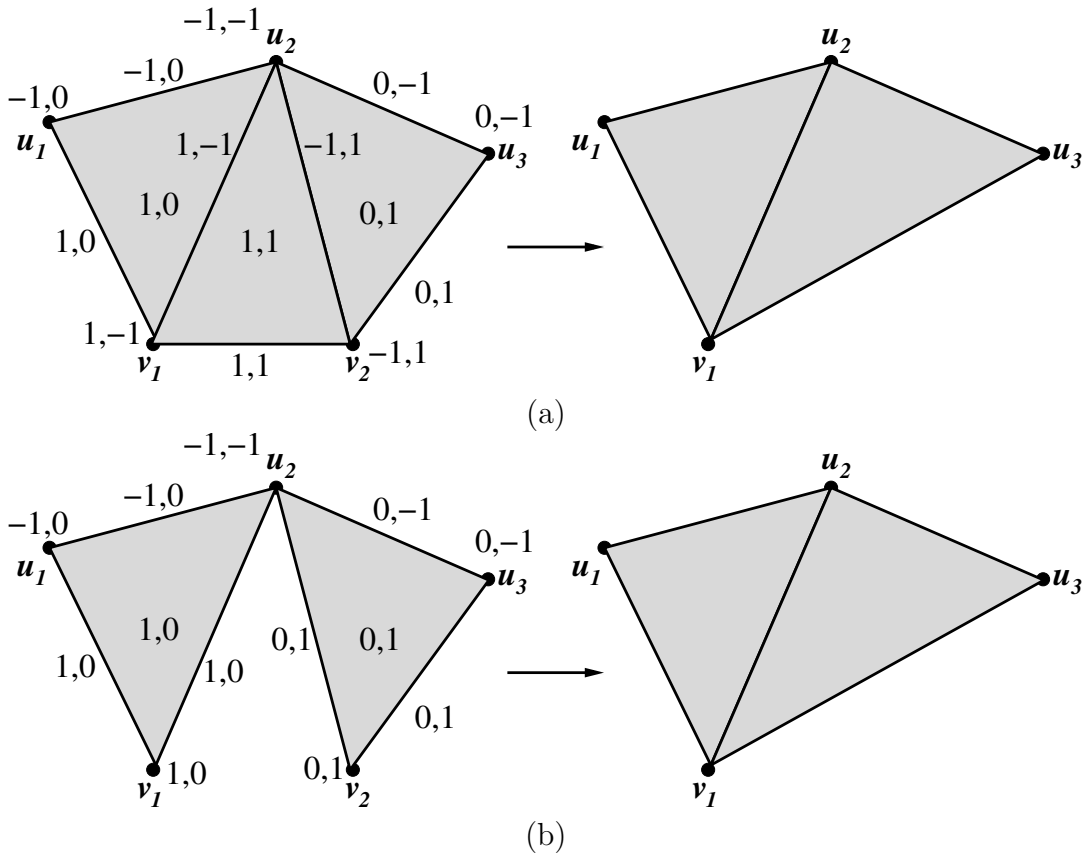


Figure 5.10: The labeling of the restricted stars and the links of vertices v_1 and v_2

- Any p-simplex σ labeled $(-1, 1)$ is merged into a p-simplex σ' incident at v_1, i.e., v_2 is replaced in σ with v_1, thus giving σ' (see, for instance, edge $\{u_2, v_2\}$ in Figure 5.10(a)).

- Any p-simplex labeled $(0, 1)$, which has a $(p-1)$-face labeled $(-1, -1)$ is also merged into a p-simplex σ' incident at v_1, as in the case above (see, for instance, edge $\{u_2, v_2\}$ in Figure 5.10(b)). Any p-simplex labeled $(0, 1)$, which does not satisfy the latter condition, is transformed into a new p-simplex incident at v_1 (see, for instance, edge $\{u_3, v_2\}$ in Figures 5.10(a) or 5.10(b)).

- Any p-simplex σ labeled $(1,1)$ is mapped to its $(p-1)$-face, σ' which is labeled $(1,-1)$, i.e. that face which is incident at v_1 and in the link of v_2 (see, for instance, the triangle $\{u_2, v_1, v_2\}$ in Figure 5.10(a)).

5.2.5.2 Updating Boundary Relations

In the vertex-pair contraction algorithm, boundary relations are affected for all simplexes generated by transforming simplexes of Σ incident at v_2 into simplexes incident at v_1 in Σ_R, and that are not merged into other simplexes incident at v_1.

Boundary relation $R_{1,0}$ is updated for every edge e labeled $(0,1)$, which is not incident at a vertex labeled $(-1,-1)$. In other words, edge e must be incident at vertex at v_2, but not at v_1 and its other extreme vertex cannot be in the intersection of the links of v_1 and v_2. For example, boundary relation $R_{1,0}$ for edge e_3, in Figure 5.11, consists of $\{v_2, u_2\}$ in Σ, and it consists of $\{v_1, u_2\}$ in Σ_R.

Boundary relations are affected for all p-simplexes labeled $(0,1)$, which do not have a $(p-1)$-face labeled $(-1,-1)$. In the SIG we are interested only in boundary relations of type $R_{p,p-1}, 2 \leq p \leq d$. These latter are updated for every p-simplex σ labeled $(0,1)$ such that:

- σ is not merged in a p-simplex incident at v_1
- σ has a $(p-1)$-face σ_2 mapped into a $(p-1)$-simplex σ_1 incident at v_1.

Simplex σ_2 is identified as having label $l_2 = 1$ and having a $(p-2)$-face labeled $(-1,-1)$. For instance, edge e_1 in Figure 5.11 replaces edge e_2 in the boundary relation $R_{2,1}$ of triangle f. The algorithm for updating boundary relations applies

the rule above recursively on the dimensions of the simplexes. The simplex σ_1 on which σ_2 is merged is retrieved from the hash table.

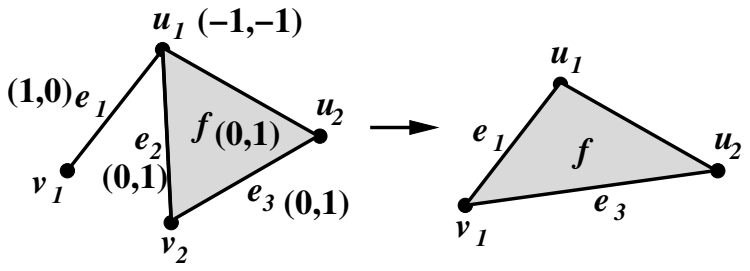


Figure 5.11: Example of the update of the boundary relations of simplexes in $st(v_2)$

5.2.5.3 Updating Partial Co-boundary Relations of Type $R^*_{p,p+1}$

Co-boundary relations are affected in two situations:

1. for any p-simplex incident at v_1, on which a p-simplex incident at v_2 is mapped,
2. for any p-simplex incident at v_2 which is not mapped into any p-simplex incident at v_1.

We consider here partial co-boundary relations of type $R^*_{p,p+1}$. In the restricted star of a p-simplex, each p-connected $(p+1)$-component consists of just one $(p+1)$-simplex. Thus, partial co-boundary relations $R^*_{p,p+1}$ is updated on the basis of the original relations $R^*_{p,p+1}$ in Σ, of boundary relations $R_{p,p-1}$ in Σ_R and of the labels of the simplexes.

We apply the update of partial co-boundary relations in decreasing order of dimension. We consider a p-simplex σ_1 incident at v_1 and we perform the following

steps:

1. For every simplex $\sigma \in R^*_{p,p+1}(\sigma_1)$ in Σ,

   - if σ is labeled $(1,1)$, then σ is not in $R^*_{p,p+1}(\sigma_1)$ in Σ_R, since σ is reduced to σ_1, (for example, in Figure 5.12(a), tetrahedron t is not in $R^*_{2,3}(f_1)$ in Σ_R);

   - if σ is labeled $(1,0)$ or $(1,-1)$, and there is a same-dimensional simplex σ' incident at v_2 such that $\sigma = F(\sigma')$ and σ' is not a top simplex, then σ is not in $R^*_{p,p+1}(\sigma_1)$ in Σ_R, since σ becomes a face, (for example, in Figure 5.12(b) triangle f_1 merges with triangle f_2, but f_2 is a face of tetrahedron t, so f_1 is not in $R^*_{1,2}$ relation for edge $\{v_1, u\}$ in Σ_R);

   - otherwise, σ remains in $R^*_{p,p+1}(\sigma_1)$ in Σ_R, since it is not affected by the transformation, (for example, in Figure 5.12(c), triangle f_1 remains in $R^*_{1,2}(e_1)$)

2. If there exists a p-simplex σ_2 incident at v_2 such that $\sigma_1 = F(\sigma_2)$, then, for every $\sigma \in R^*_{p,p+1}(\sigma_2)$ in Σ,

   - if σ is labeled $(1,1)$, then σ is not in $R^*_{p,p+1}(\sigma_1)$ in Σ_R, since σ is reduced in dimension (in Figure 5.12(a), for example, tetrahedron t is not in $R^*_{2,3}(f_1)$ in Σ_R);

   - if σ is labeled $(0,1)$ and does not have a p-face labeled $(-1,-1)$, then σ is in $R^*_{p,p+1}(\sigma_1)$ in Σ_R, (for example, in Figure 5.12(c), triangle f_2 becomes an element in $R^*_{1,2}(e_1)$).

3. After updating $R^*_{p,p+1}(\sigma_1)$, if σ_1 becomes a top simplex, then σ_1 is added to the $R^*_{p-1,p}$ relation of all its $(p-1)$-faces. These latter can be retrieved from boundary $R_{p,p-1}$ relation of σ_1 (for example, in Figure 5.12(a), f_1 is a top-simplex after contraction, so it becomes an element in the $R^*_{1,2}$ relation of all its edges).

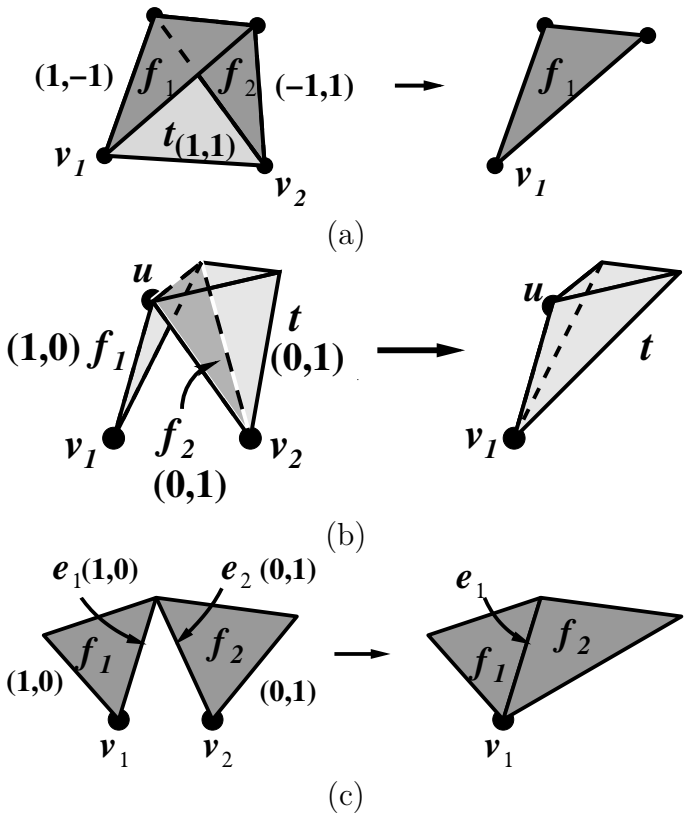


Figure 5.12: Examples of the update of the partial co-boundary $R^*_{1,2}$ relations of simplexes in $st_R(v_1)$

5.2.5.4 Updating the Remaining Partial Co-boundary Relations

The problem with updating partial co-boundary relations $R^*_{p,q}$ with $q > p+1$ is that the SIG does not store all q-simplexes incident at a given simplex σ, but just one representative for each $(q-1)$-connected q-component incident at σ. Thus, we need to traverse all $(q-1)$-connected q-component in $st_R(v_1)$, and select one representative for each q-component in the $R^*_{p,q}$ relation. The traversal algorithm uses the boundary relations $R_{q,q-1}$ for the q-simplexes in the q-component, and the updated partial co-boundary relations $R^*_{q-1,q}$ of their (q–1)-faces as well as the labels of such faces.

We start with one q-simplex σ in the $(q-1)$-connected q-component, and visit each $(q-1)$-face τ of σ. If τ was not labeled in Σ (-1,0) or (0,-1), i.e., τ is not in $lk_R(v_1)$, then all the top q-simplexes in $R^*_{q-1,q}(\tau)$ are visited.

At the end of the q-component traversal, we update $R^*_{p,q}(\sigma)$, for each σ such that there exists exactly one q-simplex in $R^*_{p,q}(\sigma)$ in Σ_R that is marked as visited. In the example of Figure 5.13, let us assume that $R^*_{1,3}(e_1)$ consists of t_1 and $R^*_{1,3}(e_2)$ consists of t_2 before contraction. After the contraction, we will have $R^*_{1,3}(e_1) = t_1$, since t_1 and t_2 form just one 3-component at e_1.

Also, we need to update the same partial co-boundary relations for the simplexes in the link of v_1 in Σ_R which were originally the simplexes in the intersection of the links of v_1 and v_2. The update is performed in a similar way as describe above. The description is omitted for brevity.

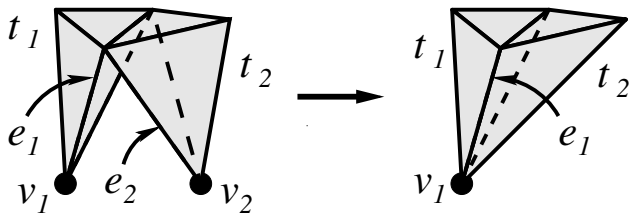


Figure 5.13: Example of the update of the partial co-boundary $R^*_{1,3}$ relations of simplexes in $st_R(v_1)$

5.2.6 Encoding a Vertex Expansion for the SIG

Vertex expansion is the inverse operation with respect to vertex-pair contraction. It consists of expanding a vertex v_1 in a simplicial complex Σ into two vertices v_1 and v_2. There may or may not be an edge $e = (v_1, v_2)$ joining them. The k-simplexes in $st(v_1)$ either expand into $(k+1)$-simplexes forming $st(e)$, if e exists, or become incident at v_1 or v_2, or are duplicated. A vertex expansion transformation reverses the vertex-pair contraction transformation and, thus, produces the simplicial complex Σ from the reduced complex Σ_R.

While a vertex-pair contraction is entirely specified by the two vertices to be contracted, for the expansion of vertex v_1 into pair (v_1, v_2) we need not only to specify the new vertex v_2, but also how the simplexes incident at v_1 are transformed. We associate a two-bit code with each p-simplex σ belonging to the star of v_1 in Σ_R, denoted $cd(\sigma)$ (we write cd for short where it is clear which simplex is being addressed). Thus, the value for cd for each such p-simplex σ is: 00 if σ is not affected by the expansion, 01 if σ is transformed into a p-simplex in Σ, incident at v_2, 10 if σ is expanded into two p-simplexes in Σ, one incident at v_1 and the other incident at v_2, and, 11 if σ is expanded into a $(p+1)$-simplex incident at both v_1 and v_2.

Such codes are computed during vertex-pair contraction as described below. We consider a p-simplex σ in Σ_R, incident at v_1. We denote by σ' the same simplex in Σ (i.e., $\sigma = \sigma'$), by γ the p-simplex of Σ (if it exists) which results from replacing v_1 by v_2 in σ', and by π the $(p+1)$-simplex in Σ (if it exists) which results from expanding σ'. The code of simplex σ is computed as follows:

- $cd = 00$, if and only if σ' is labeled $(1, 0)$ and none of its $(p-1)$-faces is labeled $(-1, -1)$; in vertex expansion, σ remains incident at v_1 (for example, in Figure 5.14(a), triangle f_1 is given a code 00);

- $cd = 01$, if and only if γ is labeled (0,1), but none of its $(p-1)$-faces is labeled (-1,-1); in vertex expansion, this simplex becomes incident at v_2 (for example, in Figure 5.14(a), the code of triangle f_2 is 01);

- $cd = 10$, if and only if σ' is labeled (1,0) and one of its $(p-1)$-faces is labeled (-1,-1); in vertex expansion, this simplex is expanded into the two simplexes σ' and γ (for example, in Figure 5.14(a), edge e_1 has code 10);

- $cd = 11$, if and only if σ' is labeled (1,-1), meaning that σ' is a face of some $(p+1)$-simplex π that is reduced by the contraction. (for example, in Figure 5.14(b), edge e_1 has code 11).

A code is associated with every top simplex in $st_R(v_1)$ and every non-top simplex in $st_R(v_1)$ that is split or expanded in vertex split. All other simplexes in $st_R(v_1)$ behave the same way as their co-faces in the vertex split, so they do not need

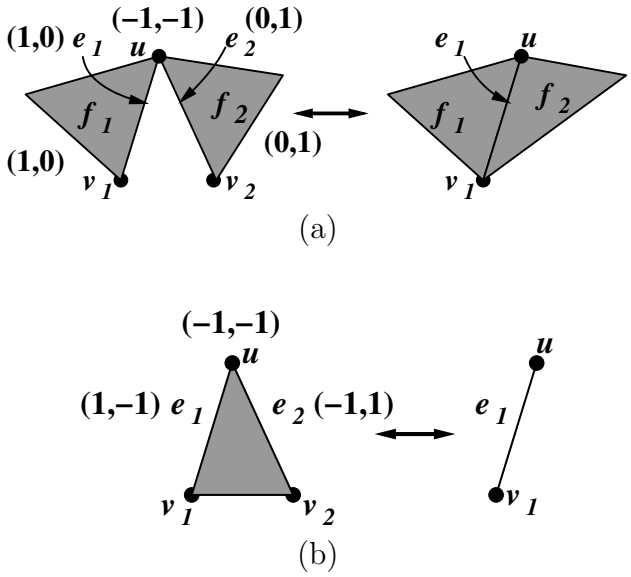


Figure 5.14: Example of the generation of code *cd* during vertex-pair contraction

to be explicited encoded. The encoding of $st_R(v_1)$ is organized by decreasing order of simplex dimension first, and then according to the lexicographical order of the vertices defining the simplexes. This gives a unique sequence, which is independent of the data structure used for encoding the simplicial complex, and, thus, a unique encoding for $st_R(v_1)$.

An encoding scheme of $st_R(v_1)$ that associates a code with every edge instead of top-simplex is generally more compact in terms of the storage cost [36, 68], but it will work well only when the edges in $st_R(v_1)$ are all explicitly encoded in the data structure. Our encoding is not optimized for compactness since a compressed representation will affect the efficiency of the vertex split operation on the SIG.

As vertex expansion operation reverses the effect of vertex-pair contraction, for brevity, we will not discuss the steps involved in this operation.

5.3 The Incidence Simplicial (IS) Data Structure

In this Section, we consider a second paradigm of cost-efficient dimension-independent data structure, the Incidence Simplicial Data Structure, that encodes all simplexes. We discuss its design, evaluate its storage cost and its navigation efficiency. We also discuss its scalability to manifold models. In addition, we consider how it can support the vertex-pair contraction operation.

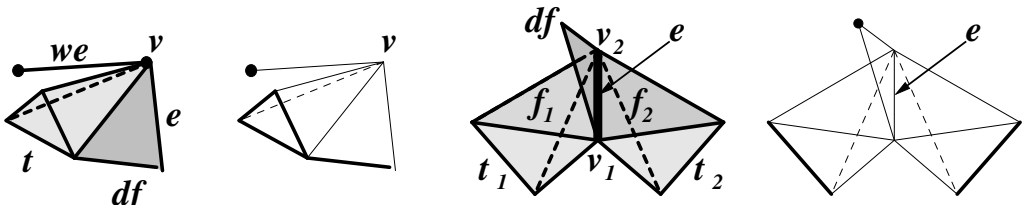
5.3.1 Design of the Data Structure

The *Incidence Simplicial (IS)* data structure is a new dimension-independent data structure for representing Euclidean simplicial complexes in arbitrary dimensions. It encodes all simplexes of a d-dimensional simplicial complex Σ embedded in the n-dimensional Euclidean space (with $d \leq n$), plus the following relations:

- for each p-simplex σ, where $0 < p \leq d$, boundary relation $R_{p,p-1}(\sigma)$,
- for each p-simplex σ, where $0 \leq p < d$, a partial version of partial co-boundary relation $R_{p,p+1}(\sigma)$, denoted as $R^*_{p,p+1}(\sigma)$, that consists of one arbitrarily-selected $(p+1)$-simplex for each connected component of the link of σ.

In the example of Figure 5.15(a), the link of vertex v (shown in Figure 5.15(b)) consists of two connected components. There are in total five edges incident at v, of which four come from the same component. Thus, partial co-boundary relation $R^*_{0,1}(v)$ consists of $\{we, e\}$, where e is an edge of triangle df. In the example of Figure 5.15(c), the link of edge e (shown in Figure 5.15(d)) is composed of three connected

components corresponding to triangle df, and to tetrahedra t_1, t_2 and their faces which are incident at e. Thus, partial co-boundary relation $R^*_{1,2}(e)$ consists of just three elements, namely $\{df, f_1, f_2\}$, where f_1 is a face of t_1 and f_2 is a face of t_2.



(a) (b) (c) (d)

Figure 5.15: Two examples of 3-complexes with non-manifold singularities: (a) shows an example of a non-manifold vertex v, whose link is shown in thick lines and a vertex in (b); (c) shows an example of a non-manifold edge e; (d) shows the link of e

In general, for each $(d-1)$-simplex σ, partial co-boundary relation $R^*_{d-1,d}(\sigma)$ is the same as co-boundary relation $R_{d-1,d}(\sigma)$. If the domain of Σ is a manifold, then $R^*_{p,p+1}(\sigma)$ contains just one $(p+1)$-simplex since the link of σ consists of one single connected component. Also, when $d = n$, every $(d-1)$-simplex is shared by at most two d-simplexes, since any d-dimensional simplicial complex embedded in the d-dimensional Euclidean space is a pseudo-manifold. Non-manifold singularities are defined by simplexes of dimension lower than the dimension n of the embedding space whose link consists of more than one connected component. Such singularities are made explicit by the encoding of the partial co-boundary relations, in which one simplex is considered for each connected components in the link of each simplex.

Consider the example in Figure 5.1, the topological relations encoded by the IS are shown in the form of directed graphs in Figures 5.16 and 5.17. Observe that

the directed graph in Figure 5.16 is the same as those in Figure 5.2 and Figure 5.6 because the IS, the SIG and the IG encode the same boundary relations. The diagram in Figure 5.17 has significantly fewer arcs than the one in Figure 5.3. In Figure 5.17, arcs connect nodes of two adjacent levels while this is not the case in Figure 5.7. Note that in Figures 5.17, 5.3 and 5.7, the arcs going out from level $d-1$ into level d are the same.

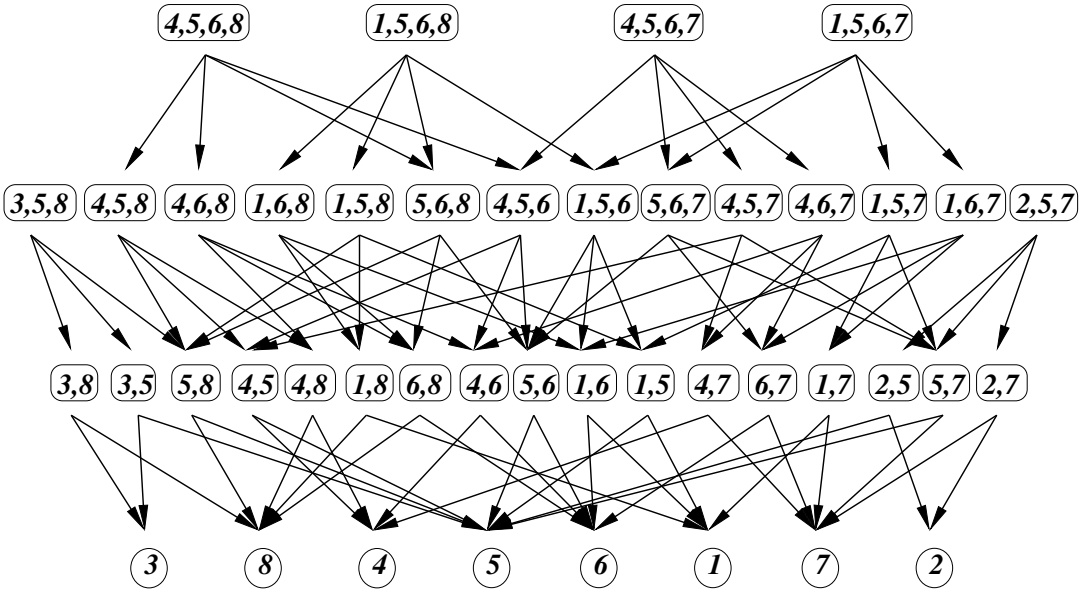


Figure 5.16: The directed graph showing the boundary relations encoded by the IS for the example shown in Figure 5.1. Nodes representing simplexes are identified by the indices of the vertices spanning them.

5.3.2 Encoding Data Structure

In the following, we describe our implementation of the IS data structure, and discuss its storage cost. Simplexes are stored in ascending order of their dimensions.

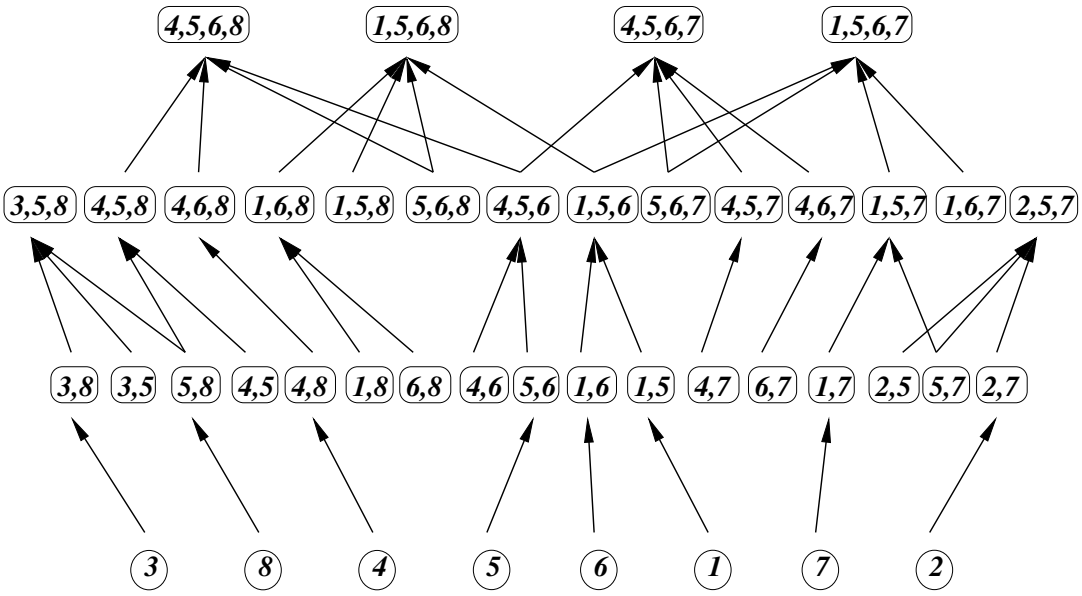


Figure 5.17: The directed graph showing the co-boundary relations encoded by the IS for the example shown in Figure 5.1

Each simplex has a unique index, and attributes can be associated to it. For each simplex, we also assign a one-bit flag that is used by the navigation algorithms to mark a simplex as visited during traversal, and is reset after traversal is completed. For each p-simplex σ, with $0 < p \leq d$, boundary relation $R_{p,p-1}(\sigma)$ is stored in a fix-sized array, each element of which is an index to a $(p-1)$-simplex on the boundary of σ.

For each p-simplex σ, with $0 \leq p < d$, we encode partial co-boundary relation $R^*_{p,p+1}(\sigma)$ in a variable-sized array. The first entry of the array stores its length. Each of the remaining entries of the array consists of the index of a $(p+1)$-simplex to which σ is a boundary. Simplex σ has an integer pointer that points to the beginning of this variable-sized array. In the manifold case, relation $R^*_{p,p+1}(\sigma)$ consists of just one element. Thus, the integer pointer for the partial co-boundary relations of σ directly

stores the index of the $(p+1)$-simplex in $R^*_{p,p+1}(\sigma)$ relation. A bit-flag is used to indicate whether the manifold condition holds at a p-simplex, when $p < d-1$.

We evaluate the storage cost of the IS data structure by assuming that both indices and pointers have the size of an integer. We denote the number of p-simplexes in a simplicial complex Σ as n_p, for $0 \leq p \leq d$. We denote the total number of connected components in the link of a simplex σ as $\kappa(\sigma)$ (when $dim(\sigma) < d$), and the total number of connected components summed over the links of all p-simplexes in Σ as $\kappa_p = \sum_{dim(\sigma)=p} \kappa(\sigma)$, for $0 \leq p < d$. Each p-simplex has a pointer to a structure in which all its attributes are stored. The total number of such pointers is $\sum_{0\leq p\leq d} n_p$ integers. As each p-simplex has $(p+1)$ $(p-1)$-faces, the encoding of boundary relations $R_{p,p-1}$ for $0 < p \leq d$ requires $\sum_{0<p\leq d}(p+1)n_p$ integers. The encoding of the partial co-boundary relations requires $\sum_{0\leq p<d}(\kappa_p + 2)$ integers (for the variable-sized arrays and the pointers that refers to them) and $\sum_{0\leq p<d-1} n_p$ bits (for the bit flags). Thus, the total storage cost of the IS data structure for a general simplicial d-complex can be summarized as follows:

- For entities: $\sum_{0\leq p\leq d} n_p$ integers;
- For boundary relations: $\sum_{0<p\leq d}(p+1)n_p$ integers;
- For co-boundary relations: $\sum_{0\leq p<d}(\kappa_p + 2)$ integers and $\sum_{0\leq p<d-1} n_p$ bits;

If Σ is a manifold complex, $\kappa(\sigma) = 1$ for $dim(\sigma) < d-1$ and $\kappa(\sigma) \leq 2$ for $dim(\sigma) = d-1$. So $\kappa_p = n_p$ for $p < d-1$. The variable-sized arrays are not needed for $0 \leq p < d-1$ because the pointer to the array can directly store the index of one simplex. For the $(d-1)$-simplexes, fix-sized arrays are used because each

$(d-1)$-dimensional simplex can be shared by either one or two d-simplexes. The storage cost of the IS data structure for a manifold d-complex reduces to:

- For entities and boundary relations: no change
- For co-boundary relations: $\sum_{0 \leq p < d-1} n_p + 2n_{d-1}$ integers and $\sum_{0 \leq p < d-1} n_p$ bits;

The overhead of the IS data structure with respect to a data structure that encodes the same topological relations but specific for a manifold d-complex is thus equal to $\sum_{0 \leq p < d-1} n_p$ bits. This is just the cost of encoding the bit flags that indicate whether a simplex is a non-manifold singularity. This means a overhead of one byte plus one bit per vertex for manifold 2-complexes, and of four bytes plus four bits per vertex for manifold 3-complexes. This shows that the IS data structure scales very well to the manifold case.

5.3.3 Building an Incidence Simplicial Data Structure

The most common exchange format for a simplicial complex consists of a collection of top simplexes described by their vertices. This representation is known as a *soup of top simplexes*, which is the collection of all the top simplexes of a simplicial complex. In this Section, we describe how to generate the IS data structure from a soup of top simplexes.

As a soup representation does not explicitly describe any simplex that is on the boundary of other simplexes, we need to generate all the simplexes of the input complex first, and then establish the topological relations among them. This is performed in four steps (note that we do the computation in decreasing order of

simplex dimension):

1. For each p-simplex, we generate its $(p+1)$ $(p-1)$-faces. Each $(p-1)$-simplex is represented by its vertices (sorted in lexicographic order).

2. All $(p-1)$-simplexes are sorted by the lexicographic order of their vertices and duplicate simplexes are removed. In this way, each simplex is given a unique integer index.

3. Boundary relations $R_{p,p-1}$ and complete co-boundary relations $R_{p-1,p}$ for all simplexes are computed as follows. For each p-simplex σ of index i, we simply consider all its $(p-1)$-faces γ. A$(p-1)$-face γ is defined as $V_\sigma - \{u\}$, where V_σ denotes the set of vertices of σ and u the vertex of σ not belonging to γ. We locate the index j of γ by binary search on the lexicographical order of its vertices, and we add γ to $R_{p,p-1}(\sigma)$ and σ to $R_{p-1,p}(\gamma)$.

4. For each $(p-1)$-simplex, its partial co-boundary relation $R^*_{p-1,p}$ (with $p < d$) is computed from the corresponding complete co-boundary $R_{p-1,p}$ relations as described below. Recall that $R^*_{d-1,d}$ is the same as $R_{d-1,d}$

The computation of the partial co-boundary relation of p-simplex σ in Step 4 is performed based on the topological information available in the intermediate structure obtained in Step 3. The topological information available for each p-simplex σ in the intermediate structure are the boundary $R_{p,p-1}(\sigma)$ relations and the co-boundary $R_{p,p+1}(\sigma)$ relations. This information can be represented as a graph in which the nodes are the simplexes in the star of σ and the arcs describe the partial

co-boundary relations encoded among these simplexes. We call this graph a *star-graph*.

Figure 5.18(a) shows an example of the star of a vertex v. Figure 5.18(b) and Figure 5.18(c) illustrate the boundary $R_{p,p-1}$ and co-boundary $R_{p,p+1}$ relations that are computed as the intermediate results at vertex v through a star-graph.

The partial co-boundary relation of σ is computed by performing a traversal on the star-graph of σ. For each p-simplex σ (with $p < d-1$), we need to identify the $(p+1)$-simplexes that belong to the same connected component of its star. This is performed by a connected component labeling algorithm. The simplest situation arises when a connected component is formed only by one $(p+1)$-simplex τ. In this case, τ will be labeled as belonging one component and added to $R^*_{p,p+1}(\sigma)$. Otherwise, a $(p+1)$-simplex in the star of σ will be on the boundary of $(p+2)$-simplexes belonging to the same connected component.

Referring to the star-graph, we need to traverse the simplexes belonging to levels $(p+1)$ and $(p+2)$ in such graph, by using relations $R_{p,p+1}$, $R_{p+1,p+2}$, and $R_{p+2,p+1}$. We start from an unlabeled $(p+1)$-simplex τ incident at σ, i.e., in $R_{p,p+1}(\sigma)$, and we mark τ with a new label. For each $(p+2)$-simplex θ in $R_{p+1,p+2}(\tau)$, we retrieve the $(p+1)$-simplexes in its boundary, i.e., belonging to $R_{p+2,p+1}(\theta)$. All $(p+1)$-simplexes μ in $R_{p+2,p+1}(\theta)$ that are incident at σ are marked with the same label. This traversal is repeated recursively for all the simplexes in $R_{p+1,p+2}(\mu)$ until all $(p+1)$-simplexes incident at σ and belonging to the same connected component are visited. Then, for each connected component in the star of σ, one $(p+1)$-simplex is selected as an element of $R^*_{p,p+1}(\sigma)$.

As an example, we consider computing $R^*_{0,1}(\sigma)$ in the complex of Figure 5.18(a) in terms of a graph traversal. Figure 5.18(d) shows the traversal of one component: starting at vertex v, we move one level up to edge e_1. By visiting all the nodes through the arcs between levels 1 and 2, we find all the edges, namely e_1, e_2 and e_3, that are in the same connected component. Edge e_1 is then selected as a representative for this component. Figure 5.18(e) shows the traversal of the other connected component in $st(v)$. Edge e_4 is selected to represent the same component. The partial $R^*_{0,1}(\sigma)$ relation thus consists of e_1 and e_4.

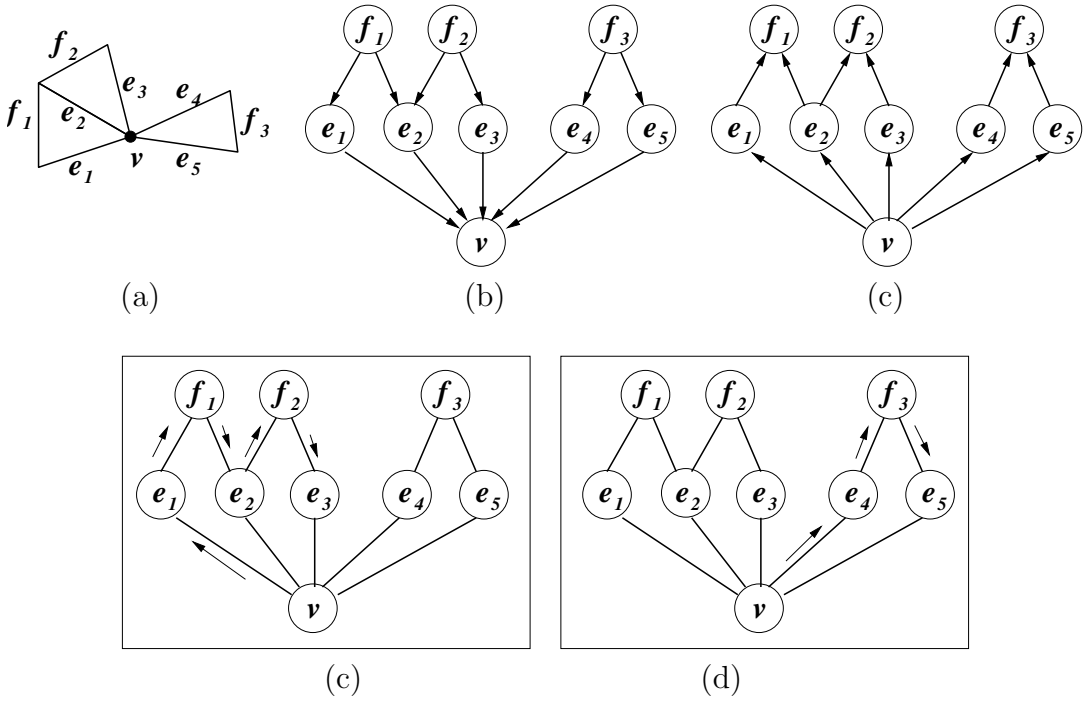


Figure 5.18: The topological relations of simplexes in the star of vertex v encoded as a star-graph: (a) The star, $st(v)$, of vertex v; (b) the boundary $R_{1,0}$ and $R_{2,1}$ relations; (c) the co-boundary $R_{0,1}$ and $R_{1,2}$ relations computed as intermediate results in the construction of the IS; (d) a traversal of one component in $st(v)$ using relations $R_{0,1}, R_{1,2}$ and $R_{2,1}$ only; (e) a traversal of the other component in $st(v)$

It should be pointed out that it is not necessary to store the complete co-boundary relations of all simplexes throughout the construction process. At any level p, only the boundary and co-boundary relations for levels $(p+1)$ and $(p+2)$ are needed.

Let us now evaluate the time complexity of the various steps in the IS construction algorithm. In step 1, at any level p, the total number of $(p-1)$-simplexes created equals to $(p+1)\cdot n_p$, where n_p is the number of p-simplexes in the whole complex. Thus, the creation of the $(p-1)$-simplexes takes time linear with respect to the number of p-simplexes. In step 2, the time required for sorting all $(p-1)$-simplexes is bounded by $O(n_p log(n_p))$. The computation of boundary and co-boundary relations in step 3 takes time linear with respect to the number of number of p-simplexes, since boundary relations are constant relations. In step 4, the traversal of the star of a simplex σ visits every arc and every node of the star-graph of $st(\sigma)$ once. In the entire complex, each q-simplex is in the stars of $(q+1)$ $(q-1)$-simplexes and in the stars of $\frac{(q+2)\cdot(q+1)}{2}$ $(q-2)$-simplexes. Thus, the computation of partial relations for all $(q-2)$-simplexes has time complexity of $\frac{(q+2)\cdot(q+1)}{2}n_q + (q+1)n_q$, which is $O(n_q)$.

5.3.4 Retrieving Topological Relations

In this Section, we present algorithms for retrieving the simplexes which are in a boundary, co-boundary or adjacency relation with a given simplex from the IS data structure.

5.3.4.1 Retrieving Boundary Relations

Boundary relations of type $R_{p,p-1}$ are directly encoded in the IS representation, while boundary relation of type $R_{p,q}$, with $q < p$ can be easily retrieved through relations $R_{p,p-1}$, $R_{p-1,p-2}$,$\cdots$, $R_{q+1,q}$. For instance, the vertices of a tetrahedron σ, i.e., in $R_{3,0}(\sigma)$, are retrieved by applying $R_{3,2}(\sigma)$, then $R_{2,1}(\tau)$, for each triangle τ in $R_{3,2}(\sigma)$, and then $R_{1,0}(\gamma)$, for each edge γ in $R_{2,1}(\tau)$.

The time complexity of this process is equal to $\Pi_{r=q+1,p+1}r$, where c is a constant. This quantity is bounded by a constant which depends on the dimension p of the simplex and on the dimension q of its faces. For instance, retrieving $R_{d,0}(\sigma)$ relation requires $O((d+1)!)$ time.

5.3.4.2 Retrieving Co-boundary Relations

Co-boundary relation $R_{d-1,d}(\sigma)$ for any $(d-1)$-simplex σ is directly encoded in the IS data structure, because $R^*_{d-1,d}(\sigma)$ is the same as $R_{d-1,d}(\sigma)$. Since only partial co-boundary relations are encoded in the IS representation, the challenge is to retrieve all complete co-boundary relations efficiently. Recall that co-boundary relation $R_{p,q}(\sigma)$ consists of all q-simplexes in the star of p-simplex σ. We will show that we can retrieve such relations in time linear in the number of top simplexes incident in simplex σ.

Observe that the q-simplexes incident at σ are either top simplexes, or faces of top simplexes in the star of σ of dimension greater than q. Thus, in order to compute the general co-boundary $R_{p,q}(\sigma)$ relation, we need to retrieve the top simplexes of

dimensions q and above from each connected component of the star of σ, since all q-simplexes that are faces of higher-dimensional simplexes incident at σ can be only retrieved from boundaries of the top simplexes incident in σ.

To illustrate the algorithm, we consider the graph describing the topological relations encoded by the IS for any q-simplex γ. We call this graph encoding the boundary relations $R_{q+1,q}$ and the partial co-boundary relations $R^*_{q,q+1}$ of γ the *IS star-graph*. Figure 5.19(a) shows an example of the star of a vertex v. The arcs of the minimal star-graph representing the boundary and the partial co-boundary relations encoded in the IS data structure for this complex are separately shown in Figures 5.19(b) and (c). Note that the vertices bounding simplexes in the star of v are not shown for clarity.

The algorithm for retrieving co-boundary relation $R_{p,q}(\sigma)$ for a p-simplex consists of a breadth-first traversal of the minimal star-graph of σ, starting at σ, as described in Algorithm 1. For each p-simplex τ in the star of σ, the traversal algorithm visits every simplex θ in its partial co-boundary relation, and visits every simplex γ in its boundary relation provided that γ is incident at τ. The traversal of the arcs representing partial co-boundary relations is described in lines 11-18 of Algorithm 1, while the traversal of the arcs representing boundary relations is described in lines 20-29. Note that simplexes of dimension p or lower are not visited because they are not in the star of σ.

Figures 5.20(a) to (d) show four stages of the traversal of the star of vertex v (from the example of Figure 5.19(a)) for retrieving $R_{0,1}(v)$. The traversal starts

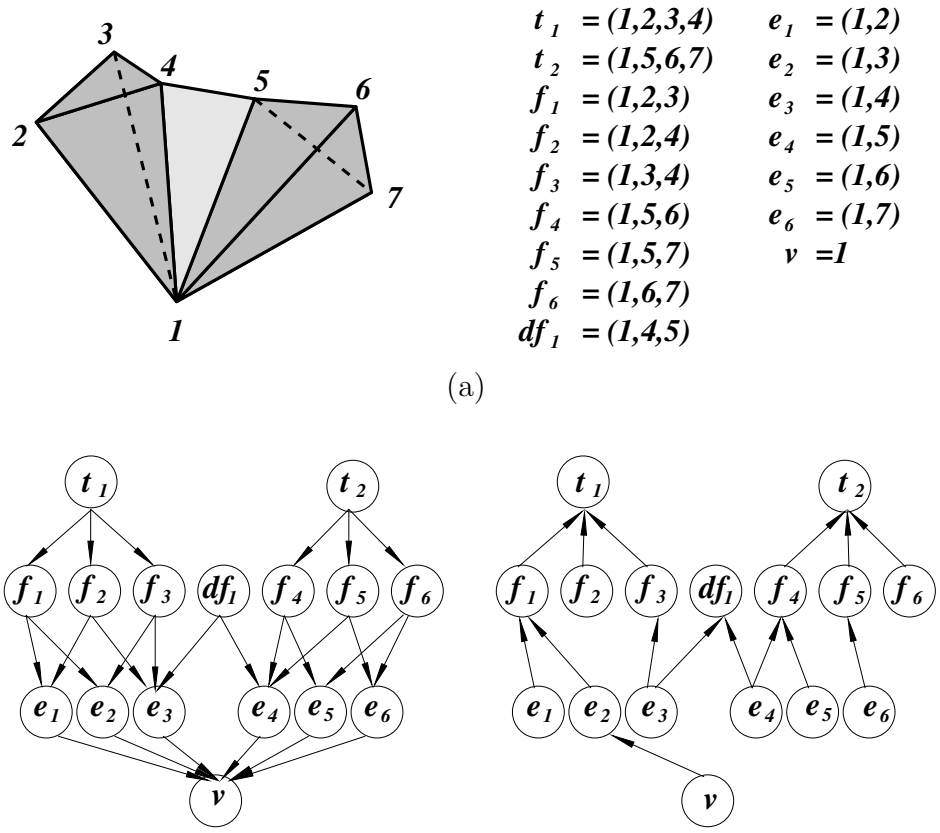


Figure 5.19: Example (part 1) of retrieving $R_{0,1}(v)$ through a traversal of the star of vertex v using boundary and partial co-boundary relations encoded by the IS: (a) the star $st(v)$ of a vertex v; (b) boundary relations encoded by the IS among simplexes in $st(v)$; (c) partial co-boundary relations encoded by the IS among simplexes in $st(v)$

from v and it is initialized by using $R^*_{0,1}(v)$, which leads to edge e_2. Through partial co-boundary relations $R^*_{1,2}(e_1)$ and $R^*_{2,3}(f_1)$, tetrahedron t_1 is visited (as shown in Figure 5.20(a)). Through the boundary relation $R_{3,2}(t_1)$, all faces of t_1 are visited. Similarly, and all the edges of the faces f_1, f_2 and f_3 are visited through their boundary relations $R_{2,1}$ (see Figure 5.20(b)). The partial co-boundary relation $R^*_{1,2}(e_3)$ of edge e_3 leads to dangling-face df_1. The boundary relation $R_{2,1}(df_1)$ of

df_1 leads to edge e_4 (as shown in Figure 5.20(c)). Through edge e_4, all the faces and edges of tetrahedron t_2 are visited in a similar fashion as those of tetrahedron t_1 (see Figure 5.20(d)). At the end of the traversal, all the edges that are in the co-boundary $R_{0,1}(v)$ relation of v are retrieved.

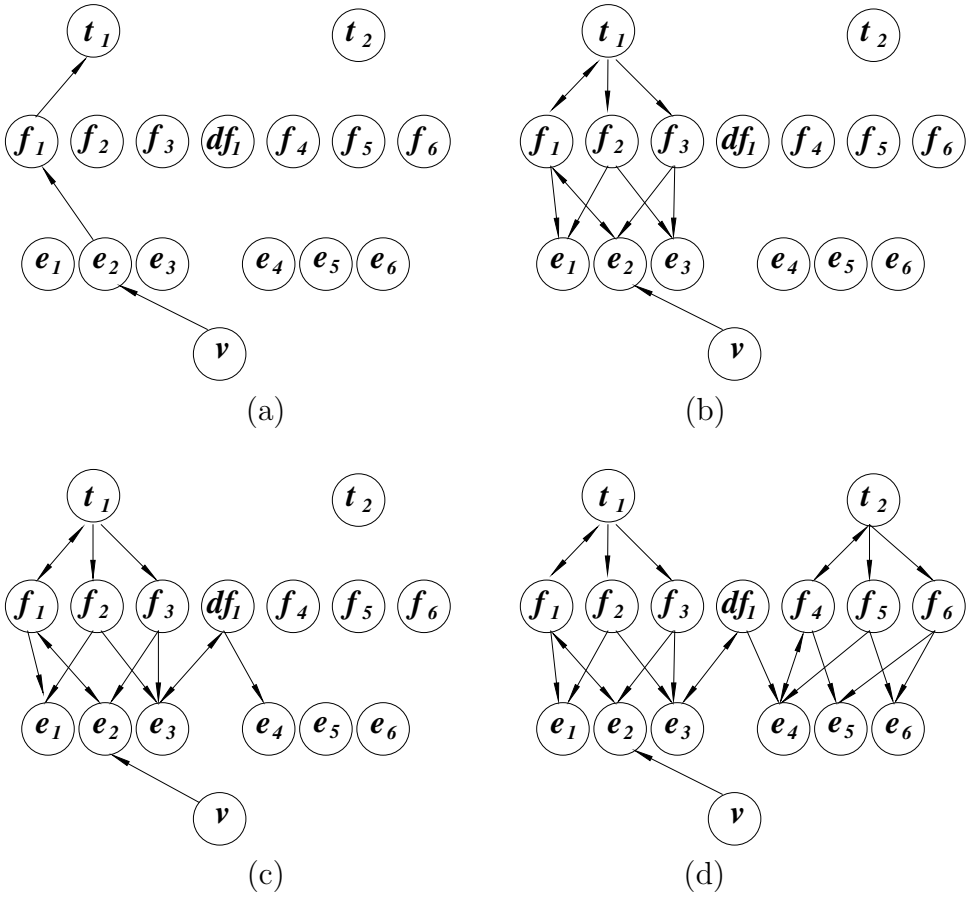


Figure 5.20: Example (part 2) of retrieving $R_{0,1}(v)$ through a traversal of the star of vertex v using boundary and partial co-boundary relations encoded by the IS: (a) to (d) are four stages of the traversal of $st(v)$

In Algorithm 1, a simplex is inserted and deleted from the queue exactly once and each arc in the incidence subgraph associated with a simplex σ is traversed exactly once. Note that the number of arcs is linear in the number of nodes in

the incidence subgraph since each simplex is bounded by a constant number of simplexes. Moreover, the total number of simplexes in the star of a simplex is linear in the number of top simplexes in the star. The time complexity of the algorithm for retrieving co-boundary $R_{p,q}(\sigma)$ is, thus, linear with respect to the number of top simplexes in the star of σ.

For simplicial 2-complexes and for simplicial 3-complexes embedded in the three-dimensional Euclidean space, any co-boundary $R_{p,q}(\sigma)$ can be retrieved in time linear in the number of q-simplexes in the star of σ. For instance, in a simplicial 3-complex, $R_{0,1}(v)$ for a vertex v can be retrieved in time linear in the number of edges incident at v, since the number of tetrahedra, triangles and edges incident at a vertex v are all linear in each other (from Euler's formula). On the contrary, for instance, in a simplicial 4-complex, $R_{0,1}(v)$ for a vertex v cannot be retrieved in time linear in the number of edges incident at v, since the number of 4-simplexes incident at a vertex v can be quadratic in the number of such edges [59]. In general, when $p \leq d-3$, the number of q-simplexes in co-boundary $R_{p,q}(\sigma)$ is linear with respect to the number of top simplexes, because the link of σ is homeomorphic in this case to a triangulated sphere and, thus, the vertices, edges and faces in the link are related by Euler' formula.

5.3.4.3 Retrieving Adjacency Relations

Adjacency relation $R_{p,p}(\sigma)$ for a p-simplex σ, when $p > 0$, is simply retrieved by first extracting the $p+1$ $(p-1)$-faces of σ, and then retrieving, for each such

face τ of σ, co-boundary relation $R_{p-1,p}(\tau)$. When $p = 0$, adjacency relation $R_{0,0}(v)$ for a vertex v is obtained by first retrieving the set of edges in the co-boundary relation $R_{0,1}(v)$, and then retrieving the other extreme vertex of each edge e in $R_{0,1}(v)$ through boundary relation $R_{1,0}(e)$.

For $p > 0$, the running time of the algorithm for retrieving $R_{p,p}(\sigma)$ is dominated by the time required to retrieve the co-boundary relations at the $(p-1)$-faces of σ. Thus, the complexity of the algorithm is linear in the total number of top simplexes incident at the $(p-1)$-faces of σ. Similarly, the time complexity of the algorithm for retrieving $R_{0,0}(v)$ is linear in the number of top simplexes incident at vertex v.

5.3.5 Vertex-Pair Contraction on the IS Data Structure

In this Section, we describe an algorithm for performing the vertex-pair contraction operation on a simplicial complex described as an IS data structure. (The formulation of the vertex-pair contraction on simplicial complex has been presented in Section 3.3. See also [68] for an informal definition of vertex-pair contraction).

The Vertex-Pair Contraction (VPC) operates on a simplicial complex Σ by merging two vertices v_1 and v_2 into a new vertex v, thus creating a new complex Σ' in which the neighborhood of v_1 and v_2 is replaced by the neighborhood of the new vertex v. Let $st(w)$ and $lk(w)$ denote the star and the link of a vertex v. VPC is defined by a map F which maps simplexes in $st(v_1) \cup st(v_2)$ onto $st(v)$, so that for each simplex $\sigma \in st(v_1) \cup st(v_2)$, $F(\sigma) = \sigma - \{v_1, v_2\} \cup \{v\}$ (see Section 3.3).

The algorithm for performing a vertex-pair contraction first retrieves the stars

and the links of v_1 and v_2. Next, it computes the vertex-pair contraction map F for all the simplexes in the stars of v_1 and v_2, thus defining the simplexes in $st(v)$. Then, it updates the boundary and the partial co-boundary relations of the simplexes in $st(v)$ as well as the partial co-boundary relations of the simplexes in $lk(v)$. The link of v is affected since any simplex τ incident in the link of v_1 or v_2 and belonging to the star of v_1 or v_2 has to be replaced with $F(\tau)$ in $lk(v)$. Moreover, connected components formed by simplexes incident in a simplex μ belonging to the two links of v_1 and v_2 may merge as an effect of the vertex-pair contraction (such as at vertex u_2 in Figure 5.21). This also affects the partial co-boundary relations of the simplexes in the link.

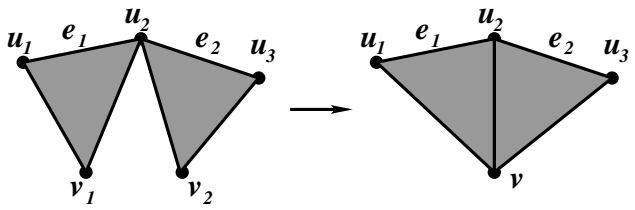


Figure 5.21: An example showing the effect of vertex-pair contraction on simplexes in the intersection of $lk(v_1)$ and $lk(v_2)$. Vertex u_2 is in $lk(v_1) \cup lk(v_2)$. Contraction of v_1 and v_2 causes two connected components at u_2 to merge into one single component.

A high-level description of the vertex-contraction algorithm is shown in Algorithm 2. Boundary relation $R_{p,p-1}(\gamma)$ for a p-simplex γ in the star of v is computed by considering just one simplex σ such that $\gamma = F(\sigma)$ and applying map F to the $(p-1)$-simplexes belonging to the boundary of σ and incident in v_1 or v_2 (see lines $9-16$ in Algorithm 2).

Complete co-boundary relations $R_{p,p+1}(\sigma)$ are computed for each p-simplex incident in v_1 or v_2. Given the stars of v_1 and of v_2, complete co-boundary relations for all p-simplexes in each star can be computed based on the boundary relations of all $(p+1)$-simplexes in the same star. This strategy is similar to that used in the construction algorithm described in Section 5.3.3. Then, co-boundary relation $R_{p,p+1}(\gamma)$ for a p-simplex γ in the star of v is computed by considering the p-simplexes σ incident in v_1 or v_2 such that $\gamma = F(\sigma)$, and applying map F to the $(p+1)$-simplexes θ belonging to the star of simplexes σ (see lines $18-20$ in Algorithm 2). Note that that we do not need to consider any $(p+1)$-simplex incident in both v_1 and v_2 and mapped by F into γ, if there exists one. Then, the partial co-boundary relations of each simplex in $st(v)$ can be found by a traversal of the star of each simplex simply using relations $R_{p,p+1}, R_{p+1,p+2}$ and $R_{p+1,p}$.

Co-boundary relations are updated for the simplexes in the links of v_1 and v_2. For simplexes not in the intersection of $lk(v_1)$ and $lk(v_2)$, the update is simply based on the boundary relations of their incident simplexes. But, for the simplexes in the intersection of $lk(v_1)$ and $lk(v_2)$, complete co-boundary relations are computed. Then the complete co-boundary relations for the simplexes in the link of v are retrieved in a similar way as for the simplexes in the star of v (see lines $24-30$ in Algorithm 2). As mentioned above, we need to retrieve the complete co-boundary relations since connected components formed by simplexes incident in the link may merge as an effect of vertex-pair contraction. Partial co-boundary relations are then obtained from complete ones as in the case of simplexes belonging to the star of v.

Note that the algorithm for the retrieval of the star, $st(\sigma)$, of a simplex σ is

very similar to that for retrieving the co-boundary relations of σ. The only difference is that the star of σ consists of every simplex encountered in the traversal, while co-boundary relation $R_{p,q}(\sigma)$ consists of only the q-simplexes encountered. Thus, Algorithm **Star(σ)**, which retrieves $st(\sigma)$, can be obtained by changing the test condition of line 7 of Algorithm 1 from $(r = q)$ to $(\tau \neq \sigma)$. In a similar manner, $lk(\sigma)$ can be computed by collecting all simplexes visited during the traversal that are not incident at σ. Algorithm **Link(σ)**, which computes $lk(\sigma)$, can be obtained by adding an else-statement to Algorithm 1 when the test condition in line 22 fails. Function F can be implemented as a hash table.

Let $\mathcal{S}(\sigma)$ be the size of $st(\sigma)$. The retrieval of the stars and links of v_1 and v_2 is of the order of $\mathcal{S}(v_1)$ and $\mathcal{S}(v_2)$ respectively. The retrieval of the co-boundary relations of simplexes in $lk(v_1) \cap lk(v_2)$ takes linear time with respect to the star of each simplex σ in $lk(v_1) \cap lk(v_2)$. The update of all boundary and co-boundary relations is linear with respect to $\mathcal{S}(v_1) + \mathcal{S}(v_2)$ The complexity of vertex-pair contraction on an IS data structure is thus $\mathcal{S}(v_1) + \mathcal{S}(v_2) + \sum_{\sigma \in lk(v_1) \cap lk(v_2)} \mathcal{S}(\sigma)$.

A high-level description of the vertex-contraction algorithm is shown in Algorithm 2. Boundary relation $R_{p,p-1}(\gamma)$ for a p-simplex γ in the star of v is computed by considering just one simplex σ such that $\gamma = F(\sigma)$ and applying map F to the $(p-1)$-simplexes belonging to the boundary of σ and incident in v_1 or v_2 (see lines $9 - 16$ in Algorithm 2).

A vertex expansion operation, seen as the reverse of a vertex-pair contraction, can be fully encoded by encoding the change of the simplexes in the star of the vertex v to be expanded. The encoding of vertex expansion is data structure dependent

in the sense that the encoding should provide sufficient information for the vertex expansion to be performed on the data structure. In Section 5.2.6, we presented an encoding scheme for performing vertex expansion on the Simplified Incidence Graph (SIG). Both the SIG and the IS encode the same sets of entities. This similarity enables the same encoding to be used for both.

5.4 Evaluation, Comparison and Discussison

In this Section, we present comparisons among our proposed and existing dimension-independent data structures for simplicial complexes. Subsection 5.4.1 presents a comparison between the IS data structure and the SIG. In Subsection 5.4.2, we discuss the differences between the IS data structure and the Incidence Graph (IG). In Subsection 5.4.3, we compare the IS data structure with the EIA for manifold simplicial complexes.

5.4.1 Comparison between the SIG and the IS

In this Subsection, we compare the SIG and the IS data structure in terms of their designs. Then, we evaluate their difference in terms of the storage cost.

The SIG stores the same boundary relations as the IS data structure, but the SIG and the IS data structure encode different subsets of co-boundary relations. In the SIG, the star of each non-manifold p-simplex σ is encoded through the partial co-boundary relation $R^*_{p,g}(\sigma)$, with one representative top simplex in the star of σ for each $(g-p-2)$-connected regular $(g-p-1)$-dimensional component in the link

of σ. In the IS, the same star is encoded with one representative $(p+1)$-simplex in the star of σ for each connected component in the link of σ. The difference occurs when the star of σ consists of top simplexes of mixed dimensions, because a $(g-p-2)$-connected regular $(g-p-1)$-dimensional component is a special class of connected component.

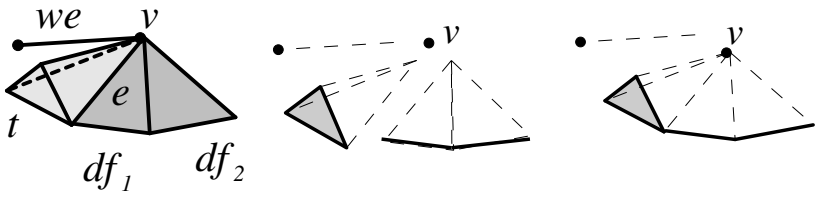


Figure 5.22: Illustrations of difference in the subsets of co-boundary relations encoded by the SIG and by the IS: (a) a vertex whose star consists of tetrahedron t, dangling-faces df_1 and df_2, and wire-edge we; (b) the $(h-1)$-connected regular h-components in the link of v, for $h = 0, 1, 2$; (c) the connected components in the link of v

This is illustrated in Figures 5.22(a)-(c). Figure 5.22(a) shows a vertex whose star consists of one tetrahedron, two dangling-faces and one wire-edge. Figure 5.22(b) shows the $(h-1)$-connected regular h-components in the link of v for $h = 0, 1, 2$, superimposed on the top simplexes in the star of v which correspond to these components. There is one such component of dimension 2, one of dimension 1 and one of dimension 0. Thus the SIG encodes $R^*_{0,1} = \{we\}$, $R^*_{0,2} = \{df_1\}$ and $R^*_{0,3} = \{t\}$. Figure 5.22(c) shows the connected components in the link of v. There are only two connected components. The IS encodes $R^*_{0,1} = \{we, e\}$, where e is an edge in the star of v which represents the mixed-dimensional component.

Note that a complex is two-dimensional, there are only vertices, edges and triangles. Therefore, every component in the link of a vertex is either a single vertex or a connected set of edges. As a result, the total size of the partial co-boundary relations encoded by the SIG and the IS is the same. Therefore, the storage costs of the IS data structure and the SIG are identical for simplicial 2-complexes. In the case of a manifold d-complex, the SIG and the IS also have the same storage cost because the link of every p-simplex is just one $(d-p-2)$-connected regular $(d-p-1)$-dimensional component.

We experimented on six simplicial 3-complexes with non-manifold edges shown in Figure 5.2(a). The storage costs of the two data structures are evaluated respectively for the boundary relations encoded, and the different sets of partial co-boundary relations encoded. The predominant component in the storage costs of both data structures arise from the encoding of the boundary relations. The different sets of partial co-boundary relations encoded are about half the size of the boundary relations. The ratio of the storage that the IS spent on encoding its subsets of partial co-boundary relations to that of the SIG varies from 85% to 99%. The IS is slightly more compact than the SIG. Comparing between the partial co-boundary relations encoded by the IS with those by the SIG, the IS is 85% to 99% that of the SIG. This saving comes from the different sets of partial relations encoded at the 3D non-manifold vertices (see the Example in Figure 5.22). Note that the components in the star of 3D non-manifold edges are uniformly-dimensional. Therefore, they do not contribute to the difference between the IS and the SIG in their storage costs.

Data set	n_0	n_1	n_2	n_3	C_e	$C_v - n_0$
Blocks	5.00k	2.83k	40.9k	17.6k	3,25k	0
400-cubes	2.82k	13.6k	17.0k	6.23k	4,29k	0
200-cubes	1.90k	77.6k	9.00k	3.29k	1,10k	0
Star1	41	120	112	32	48	0
Star2	33	80	64	16	32	0
Hollow cube	8	18	16	4	6	0

(a)

Data set	Boundary	co-boundary subset IS	co-boundary subset SIG
Blocks	249,363	105,141	106,844
400-cubes	103,139	43,465	46,007
200-cubes	55,684	23,383	24,295
star1	704	313	352
star2	416	193	224
Hollow cube	100	48	56

(b)

Table 5.2: (a) Six tetrahedral data sets with non-manifold edges; (b) Sizes of the boundary relations encode, and the different sets of partial co-boundary relations encoded by the IS and by the SIG

On the efficiency of navigation, both the SIG and the IS data structure support the retrieval of boundary relations in optimal time. On co-boundary and adjacency relations, both data structures support their retrieval in optimal time for complexes up to dimension 3. In the following, we discuss the navigation efficiency of each data structure for general simplicial complexes of dimension beyond 3.

The SIG supports the retrieval of co-boundary relations $R_{p,d}(\sigma)$ in optimal time for any p-simplex $p < d$. This is because the SIG encodes the partial co-boundary $R^*_{p,d}$ relations which enables the efficient traversal of each $(d-1)$-connected d-components in the star of σ. The retrieval of all other co-boundary and adjacency relations are sub-optimally supported.

The retrieval of all co-boundary and adjacency relations is sub-optimally supported by the IS at all times because the retrieval of any co-boundary relation at σ involves the traversal of all the simplexes in the star of σ. Note that when the complex is manifold, the retrieval of co-boundary $R_{p,d}$ relation is also supported in optimal time by the IS because the star of each simplex consist of just one d-dimensional component.

A summary of the comparisons between the SIG and the IS for navigation efficiency is shown in Table 5.3.

Algorithm 1 Co-boundary(q, σ)

Require: $q > dim(\sigma)$

1: $S \leftarrow \emptyset$
2: Mark σ as visited
3: Enqueue(Q, σ)
4: **while** not Empty(Q) **do**
5: $\tau \leftarrow$ Dequeue(Q)
6: $r \leftarrow dim(\tau)$
7: **if** $r = q$ **then**
8: Add τ to S
9: **end if**
10: { *Visit co-boundary of* τ }
11: **if** $r < d$ **and** $R^*_{r,r+1}(\tau) \neq \emptyset$ **then**
12: **for** each $\theta \in R^*_{r,r+1}(\tau)$ **do**
13: **if** θ is not visited **then**
14: Mark θ as visited
15: Enqueue(Q, θ)
16: **end if**
17: **end for**
18: **end if**
19: { *Visit boundary of* τ }
20: **if** $r > dim(\sigma) + 1$ **then**
21: **for** each $\gamma \in R_{r,r-1}(\tau)$ **do**
22: **if** σ is on the boundary of γ **then**
23: **if** γ is not visited **then**
24: Mark γ as visited
25: Enqueue(Q, γ)
26: **end if**
27: **end if**
28: **end for**
29: **end if**
30: **end while**
31: **return** S

Algorithm 2 VertexPairContract(v_1, v_2)

1: $S \leftarrow$ **Star**(v_1) $\cup$ **Star**(v_2)
2: $L \leftarrow$ **Link**(v_1) $\cup$ **Link**(v_2)
3: Compute and store $F(\sigma)$ for each p-simplex σ in S
4: Retrieve $R_{p,p+1}(\sigma)$ for each p-simplex σ in S
5: **for** each p-simplex σ in S **do**
6: $\quad\gamma \leftarrow F(\sigma)$
7: $\quad$**if** $dim(\gamma) = dim(\sigma)$ **then**
8: $\quad\quad$\{*Update boundary relations of* γ\}
9: $\quad\quad$**for** each $(p-1)$-simplex τ in $R_{p,p-1}(\sigma)$ **do**
10: $\quad\quad\quad$**if** τ is in S **then**
11: $\quad\quad\quad\quad$Add $F(\tau)$ to $R_{p,p-1}(\gamma)$
12: $\quad\quad\quad$**else**
13: $\quad\quad\quad\quad$\{ τ *is in* L\}
14: $\quad\quad\quad\quad$Add τ to $R_{p,p-1}(\gamma)$
15: $\quad\quad\quad$**end if**
16: $\quad\quad$**end for**
17: $\quad\quad$\{*Update co-boundary relations of* γ\}
18: $\quad\quad$**for** each $(p+1)$-simplex θ in $R_{p,p+1}(\sigma)$ **do**
19: $\quad\quad\quad$Add $F(\theta)$ to $R_{p,p+1}(\gamma)$
20: $\quad\quad$**end for**
21: $\quad$**end if**
22: **end for**
23: \{*Update co-boundary relations of the links* L\}
24: **for** each p-simplex σ in L **do**
25: $\quad$**for** each $(p+1)$-simplex θ in $R_{p,p+1}(\sigma)$ **do**
26: $\quad\quad$**if** θ is in S **then**
27: $\quad\quad\quad$Replace θ by $F(\theta)$ in $R_{p,p+1}(\sigma)$
28: $\quad\quad$**end if**
29: $\quad$**end for**
30: **end for**
31: Compute partial co-boundary $R^*_{p,q}(\sigma)$ for each p-simplex σ in $S \cup L$

Dimension	Data Structure	Boundary relations	Co-boundary relations	Adjacency relations
$d \leq 3$	SIG	Optimal	Optimal	Optimal
	IS	Optimal	Optimal	Optimal
$d > 3$	SIG	Optimal	$R_{p,d}$: optimal Others: sub-optimal	$R_{d,d}$: optimal Others: sub-optimal
	IS	Optimal	Sub-optimal	Sub-optimal

Table 5.3: Navigation efficiency of the SIG and the IS data structure for general d-dimensional complexes

5.4.2 Comparison with the Incidence Graph (IG)

The Incidence Graph (described in Subsection 4.1.1.2) is the one dimension-independent data structures suitable for general simplicial complexes. In Section 5.4, we have compared the SIG and the IS data structure for general d-dimensional simplicial complexes. In this Section, we compare the IS data structure with the Incidence Graph (IG) in terms of the storage costs, of efficiency in retrieving topological relations and performing update operations.

The IS data structure and the IG store the same entities, and boundary relations, but the IG encodes the complete co-boundary relations of type $R_{p,p+1}$. Thus, the IG thus occupies $\sum_{p=1}^{d-1}(p+1)n_p - \sum_{q=0}^{d-2}(\kappa_q)$ integers more than the IS. Recall that we denote the total number of connected components in the link of a simplex σ as $\kappa(\sigma)$ (when $dim(\sigma) < d$), and the total number of connected components summed over the links of all p-simplexes in Σ as $\kappa_p = \sum_{dim(\sigma)=p} \kappa(\sigma)$, for $0 \leq p < d$. The above difference is maximized when encoding a manifold complex. In this case,

$\kappa_q = n_q$ and thus the difference is $(d+1)n_d + \sum_{q=1}^{d-1} qn_q - n_0$. Figure 5.23 provides an example at a vertex v whose star is a manifold 3-complex. The IG encodes all the seven edges incident at v while the IS encodes only edge e. The difference between the IG and the IS data is minimum when only $(q+1)$-simplexes are incident at all q-simplexes, in which case, $\kappa_q = (q+2)n_{q+1}$. An example for this case is when the whole complex consists of isolated edges. Only relations $R_{0,1}$ and $R_{1,0}$ are encoded. Then the IG and the IS have the same storage cost.

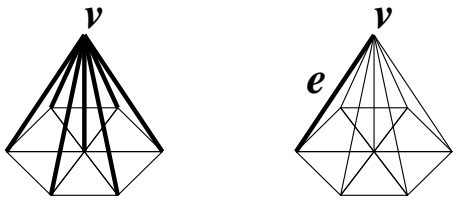


Figure 5.23: A comparison between $R_{0,1}(v)$ and $R^*_{0,1}(v)$ in the manifold case. In (a) IG encodes all the edges that are incident at v. In (b), IS encodes only edge e in the star of v

Retrieving boundary relations is performed in the same way for IG and IS data structures, and requires constant time. Retrieving co-boundary relation $R_{p,q}(\sigma)$, with $p < q+1$ from the IS data structure requires time linear in the number of top simplexes in the star of σ. An illustrative example has been given in Figure 5.19, in which all co-boundary relations of vertex v are retrieved as faces of the top simplexes in the star of v. It can be shown that the retrieval of co-boundary relations requires time linear only in the number of q-simplexes in the star of σ from the IG when dealing with regular objects. In the case of simplicial 2- and 3-complexes, co-boundary relations can be retrieved in optimal time from both data

structures. For both the IS and the IG, the time required for retrieving adjacency relations depends linearly on the retrieval of boundary and co-boundary relations.

Update operations can be done efficiently on both the IS data structure and on the IG. The three basic steps for updating these two data structures are: first, to retrieve all the simplexes in the stars of the vertices that are to be contracted; second, to compute the simplexes that result from the contraction, and third, to update the co-boundary relations of simplexes in the star of the new vertex. The update of the IG for vertex-pair contraction is a local operation involving only simplexes in the stars of the vertices contracted. An algorithm for vertex-pair contraction operation on the IG is presented in [68]. These data structures exhibit a very low overhead when encoding manifold complexes. The IG does not distinguish between manifold and non-manifold simplexes, while these latter and their incident components are explicit in the IS data structure. This can be illustrated through the example in Figure 5.19(a). Vertex v in this example is non-manifold because nodes e_3 and e_4 in the partial incidence graph of v (as shown in Figure 5.19(c)) have more than one arc pointing up.

The simplicity of the partial co-boundary relations encoded by the IS data structure allows it to support update operations in a fashion that is conceptually comparable to the IG.

The IS data structure has a storage cost of about 80% of the IG for manifold triangle meshes, and for triangle meshes with a small amount of non-manifold singularities. The IS and the IG all have the same storage cost for encoding simplexes and boundary relations (which are the predominant cost). The partial co-boundary

relations encoded by the IS occupy as little as 50% the storage that the IG spends on encoding the full co-boundary relations for simplicial 3-complexes.

5.4.3 Comparison with the Extended Indexed Data Structure with Adjacencies (EIA)

Among dimension-independent data structures suitable for manifold simplicial complexes, the Indexed data structure with Adjacencies (IA) (see Section 4.1.1.3) is the most compact. In Section 5.4.1, we have shown that for manifold complexes, the SIG and the IS are equivalent in terms of the types of topological information they encode. In this Section, we compare the SIG and the IS data structure with the EIA.

The EIA data structure is an adjacency-based data structure. It encodes vertices, top simplexes, their incident vertices, and the adjacency relations among the top simplexes. The EIA data structure differs from the SIG and the IS data structure primarily in the entities encoded. The former encodes only the top simplexes in the complexes, which is much smaller in number than the total number of simplexes. As a result, the way a simplex is addressed in the SIG and the IS is different from that in the EIA. Since the SIG and the IS encodes all simplexes, each simplex has a unique identifier. In the EIA, a p-simplex (for $0 < p < d$) needs to be described as a face of one of the d-simplexes incident at it.

Now we compare the data structures in terms of their storage costs. Consider a complex with n_d d-simplexes and n_0 vertices. Let n_p be the number of p-simplexes.

The number of topological relations encoded by the three data structures for manifold d-complexes are respectively:

- EIA: $2(d+1)n_d + n_0$
- SIG and IS: $\sum_{0<p\leq d} n_p(p+1) + 2n_{d-1} + \sum_{0\leq p<d-1} n_p$

For the case of $n = 2$, the EIA encodes $6n_2 + n_0 \approx 13n_0$ pieces of information on topological relations while both the SIG and the IS encode $9n_2 + n_0 \approx 19n_0$ pieces of information. The storage costs of the two data structures have been evaluated for data sets of manifold 2-complexes in Table 5.4. The the case of $n = 3$, comparisons are made in Section 6.3 along with data structures specialized for 3-complexes.

Data set	n_0	n_1	n_2	deg(V)	IS/SIG	EIA
Car	6.94k	18.0k	11.8k	5.09	114k	77.7k
Doll	551	1.38k	831	4.52	8.56k	5.54k
Face	2.09k	6.15k	4.05k	5.83	38.8k	26.4k
Temple	6.85k	17.8k	11.00k	4.82	111k	72.9k
Sofa	8.09k	23.5k	15.1k	5.61	147k	98.9k
Lion	5.17k	15.2k	10.1k	5.84	96.1k	65.5k

Table 5.4: Comparison between the IS (or SIG) and the EIA on the number of topological relations they encode on manifold triangle meshes

On the efficiency of navigation, the EIA provides optimal support for the retrieval of co-boundary $R_{0,d}$ and adjacency $R_{d,d}$ relations. Other co-boundary relations $R_{p,q}$ are sub-optimally supported as their retrieval involves the retrieval of all the d-simplexes in the star of the query p-simplex σ. The retrieval of adjacency relations is dependent on that of the co-boundary relations, and are thus sub-optimal.

The SIG and the IS are comparable to the EIA in their support for efficient retrieval of boundary relations for complexes of any dimension and for efficient retrieval of co-boundary relations for complexes of dimension up to 3. Details on the navigation efficiency of the SIG and the IS data structure can be found in Section 5.4. A summary of the navigation comparison is shown in Table 5.5.

Dimension	Data Structure	Boundary relations	Co-boundary relations	Adjacency relations
$d \leq 3$	SIG/IS	Optimal	Optimal	Optimal
	EIA	Optimal	Optimal	Optimal
$d > 3$	SIG/IS	Optimal	$R_{p,d}$: optimal Others: sub-optimal	$R_{d,d}$: optimal Others: sub-optimal
	EIA	Optimal	$R_{0,d}$: optimal Others: sub-optimal	$R_{d,d}$: optimal Others: sub-optimal

Table 5.5: Navigation efficiency of SIG, IS and EIA for d-dimensional manifold complexes

5.4.4 Comparison with Existing Data Structures for Non-manifold Simplicial 2-Complexes

In this Section, we compare the storage cost of the IS data structure and the SIG with existing data structures for non-manifold simplicial 2-complexes reviewed in Section 4.1.2.2: namely, the Radial-Edge (RE), the Partial Entities (PE), the Directed Edge (DE), the Triangle-Segment (TS) and the Vertex-Face (VF) data structures. Note that the IS and the SIG have the same storage cost for 2-complexes.

We evaluate the storage costs of each of these data structures for a simplicial 2-complex with n_2 triangles, n_1 edges (including n_1^t wire-edges) and n_0 vertices. The total number of connected components at the link of non-manifold edges is denoted by C_e, and the total number of connected-components at all vertices is denoted by C_v. The storage cost of each data structure is as follows:

- RE : $73n_2 + n_1 + 4n_1^t + n_0$
- PE : $22n_2 + n_1 + 4n_1^t + 3C_v$
- DE (full-sized) : $15n_2 + 2n_1^t + C_v$
- TS : $6n_2 + C_e + C_v$
- VF : $6n_2 + 2n_1$
- IS and SIG : $6n_2 + 2n_1 + C_v$

We compare the IS and the SIG with these data structures on the data sets shown in Table 4.8(a). The results are shown in Table 5.6. It can be observed that the IS and the SIG are less space-consuming than the edge-based data structures (RE, PE and DE), but they are not as compact as the adjacency-based TS data structure.

5.5 A Multi-resolution Model

A Non-manifold Multi Tessellation is a generalization of the Multi-Tessellation proposed in [27] for manifold simplicial complexes to arbitrarily-dimensional sim-

Data set	RE	PE	DE	TS	VF	IS/SIG
cylinders	15.3k	5.07k	3.15k	1.33k	1.82k	1.92k
pies	172k	56.0k	35.3k	16.5k	19.8k	20.5k
frame	82.9k	30.1k	18.0k	7.86k	10.8k	12.2k
cubes	714k	228k	146k	74.7k	79.0k	81.2k
densetower2	1,380k	462k	286k	122k	166k	175k

Table 5.6: Storage cost of seven data structures for 2D data sets in Table 4.8(a)

plicial complexes. The basic ingredients in a Non-manifold Multi-Tessellation are updates and a dependency relation among updates. An *update* of a complex Σ is an operation that replaces a set of simplexes of Σ with another set of simplexes, under the constraint that the result is still a simplicial complex. Here, we focus on updates that change the size of a mesh by either increasing it (*refinement*), or decreasing it (*coarsening*). The *dependency relation* among refinement updates is defined as follows: an update u depends on another update u' if u deletes some simplexes introduced by u'. Under certain assumptions (see [27]), the transitive closure of the dependency relation defines a partial order among a set of refinement modifications applied to the complex at coarsest resolution. This latter is called the *base complex*. A *Non-manifold Multi-Tessellation* is defined as the base complex plus a partially ordered set of *updates* $\{\mathcal{U}\} = (\{u_0, u_1, \ldots, u_h\}, \prec)$, where each update u_i, $i = 1, 2, .., h$ represents both a refinement update and its inverse coarsening update. Figure 5.24 gives an example of an update on a non-manifold model.

A subset S of the updates of an NMT is called *closed* with respect to the partial order if, for each update $u_j \in S$, all updates u_i, such that u_i precedes u_j, are

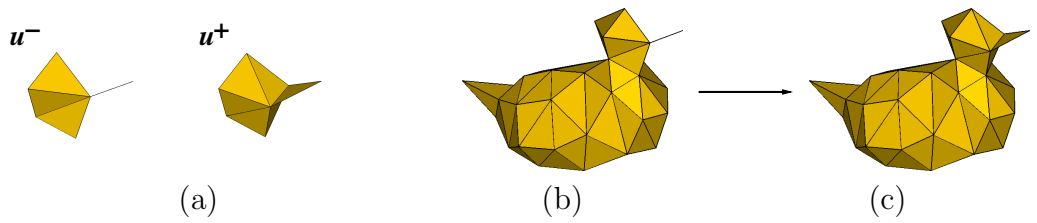


Figure 5.24: An update u applied on the beak of a non-manifold duck model: (a) u^- and u^+; (b)→(c) application of $u = (u^-, u^+)$ on the complex shown in (b); (c)→(b) application of the inverse of u to the complex shown in (c)

also in S. The refinement updates corresponding to a closed subset of nodes can be applied to the base complex Σ_0 in any total order extending the partial order. This produces an *extracted mesh* Σ_S at a level of resolution intermediate between the base complex and the complex at full resolution.

Here, we are interested in an NMT built through the iterative application of the vertex-pair contraction operation. A *vertex-pair contraction* applied to a pair of vertices v_1, v_2 of a simplicial complex Σ consists of merging v_1 and v_2, and updating all the simplexes in $st(v_1) \cup st(v_2)$ as a consequence. For simplicity, we will consider merging vertex v_2 into v_1 (in other words, we call v_1 the new vertex). *Vertex split* is the inverse operation with respect to vertex-pair contraction. It consists of splitting a vertex v_1 in a simplicial complex Σ into two vertices v_1 and v_2. There may or may not be an edge $e = \{v_1, v_2\}$. The k-simplexes in $st(v_1)$ either expand into $(k+1)$-simplexes forming $st(e)$, if e exists, or become incident at v_1 or v_2, or are duplicated. Figure 5.25 shows two examples of the operations. The vertex-pair contraction operation changes the mesh Σ on the left to the mesh Σ_R on the right. Its inverse operation change mesh Σ_R into mesh Σ.

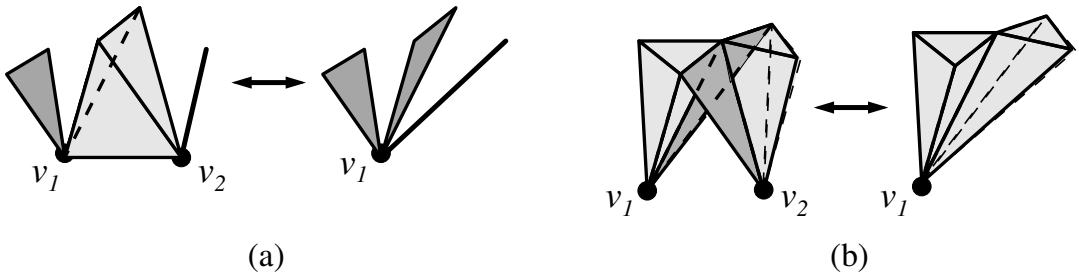


Figure 5.25: Two examples of contraction and its inverse, vertex split: vertices v_1 and v_2 are contracted into v_1

We call an NMT built through vertex-pair contraction a *Vertex-based Multi-Tessellation (VNMT).* An update in a VNMT, thus, corresponds to a contraction of a pair of vertices to a vertex and to its inverse, vertex split. The data structure encoding a VNMT consists of a procedural encoding of the updates (vertex contraction and vertex split) and a compact encoding of the partial order relation. For vertex-pair contraction, we just need to encode the vertices v_1 and v_2 which are merged, while for vertex split we need to specify the effect of the split on the simplexes in the star of v_1.

The *dependency relation* is encoded as a forest of binary trees of vertices by using a modification of the mechanism proposed in [33] for triangle meshes. The leaves of the forest correspond to the vertices of the reference mesh. Each internal node represents the vertex v_1 on which pair v_1 and v_2 is collapsed, and its two children represent vertices v_1 and v_2. Since a vertex-pair contraction does not generate a new vertex, we rename the surviving vertex v_1 and consider it as another vertex. If the pair v_1 and v_2 is collapsed to vertex v_1, then v_1' is a *renamed copy* of vertex v_1, and appears in the binary tree as the parent of vertices v_1 and v_2. In addi-

tion, we use a vertex enumeration mechanism. The n vertices of the full-resolution complex are labeled arbitrarily from 1 to n, the remaining vertices (vertex copies, in our case) are labeled with consecutive numbers as they are created during the LOD model generation through edge collapse. In this way, the label of parent v_1' will be greater than the labels of its two children v_1 and v_2. It can be shown that the information stored in the forest plus the enumeration mechanism are sufficient to retrieve the updates u' such that $u' \prec u$, when we are going to apply an update u on the currently extracted complex Σ.

The basis of any query on a multi-resolution model is *selective refinement*, which consists of extracting a complex, which satisfies some application-dependent requirements, such as approximating a spatial object with a certain accuracy which can be either uniform, or variable in space. The solution of a selective refinement query is the extracted complex Σ_S of minimum size associated with a closed set S of modifications applied to the base complex Σ_0. Selective refinement is performed by traversing the NMT and constructing a closed subset S of updates, and its associated mesh Σ_S either by recursive *top-down* refinement applied to the base complex or by an *incremental* fashion, which finds a solution to a new query by applying refinement and coarsening updates to the complex obtained as a solution to a previous query [31, 24, 38]. The incremental algorithm can be applied to the base mesh to extract a complex from scratch. So, it encompasses the top-down refinement algorithm as a special case.

In [17], we have encoded the extracted complex as a SIG (see Section 5.2). Vertex-pair contraction is the basic operation we use to generate the multi-resolution

model. Both vertex-pair contraction and vertex expansion are used when extracting variable-resolution representations from the vertex-based NMT. The algorithm for performing vertex-pair contraction on the SIG has been described in Section 5.2.5. In the following Section, we report some experimental results on the construction of an NMT using the SIG and the VPC operator.

5.5.1 Experimental Results

In this Section, we show the results of three case studies that consist of triangle meshes describing the boundary of man-made objects. The first case study is the model of a bicycle. It contains 22,000 vertices and 44,000 triangles at full resolution. The interesting characteristic of this data set is the presence of many thin long parts. The second case study is the model of a handgun. The full resolution model of it has 13,000 vertices and 22,000 triangles. This model has parts (such as the trigger) that are features of interest in different resolutions. The third case study is a model of a reductor. This model has 67,000 vertices and 133,500 triangles at full resolution. Both the bicycle and the handgun models are obtained from the Princeton Shape Depository. The reductor is from CAD data. These models are shown in Figure 5.26.

To evaluate the performance of the vertex-pair contraction algorithm, we perform a sequence of vertex-pair contraction operations on each case study to construct a series of uniform-resolution models. The experiments have been performed on a Linux PC with 1 million CPU clocks per second. The times taken in constructing

Bicycle
22,000 vertices
44,000 triangles

Handgun
13,000 vertices
22,000 triangles

Reductor
67,000 vertices
133,500 triangles

Figure 5.26: Three case studies: a bicycle, a handgun and a reductor, each shown at full resolution

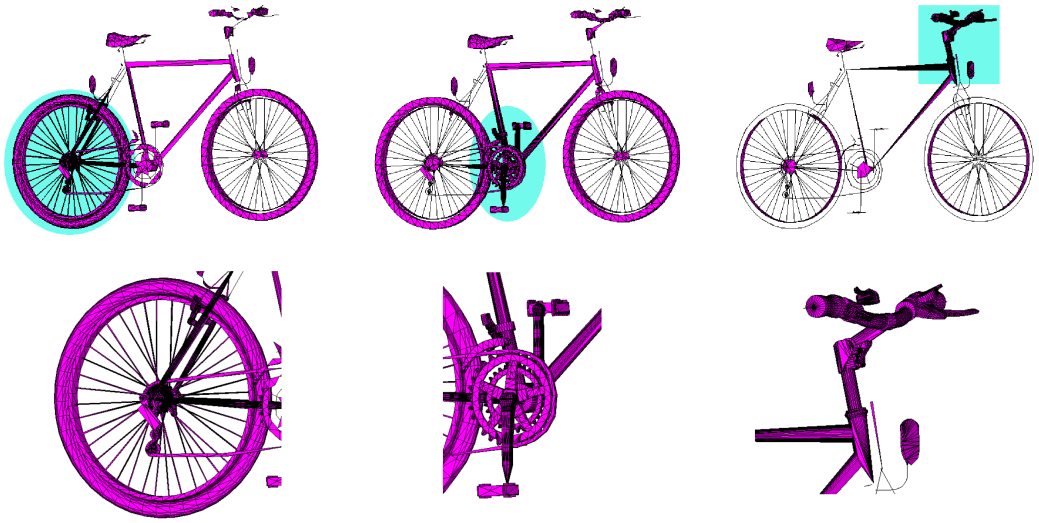
the handgun models are shown in Table 5.7.

Bicycle	Model size (number of triangles)	42000	24000	14000	2000
	Number of vertex-pair contractions	1000	10000	15000	21000
	Total time in sec	0.15	4.54	6.27	7.71
Handgun	Model size (number of triangles)	20,400	10,000	6,000	2,000
	Number of vertex-pair contractions	800	6,000	8,000	10,000
	Total time in sec	0.09	1.48	2.06	2.58
Reductor	Model size (number of triangles)	113,500	93,500	73,500	53,500
	Number of vertex-pair contractions	10,000	20,000	30,000	40,000
	Total time in sec	1.38	2.65	3.82	6.92

Table 5.7: CPU times used in performing vertex-pair contraction on the three models

We also show the results of various meshes extracted from the multi-resolution model of the bicycle model. Figure 5.27 show some variable-resolution meshes obtained. In each variable-resolution mesh, the part of the object that is of interest is shown in full resolution and is highlighted in the picture, while the remaining part of the object is at a user-designated resolution, which is uniform and is a fraction of the full resolution. This fraction is estimated by the ratio of the number of vertices

in the coarsened area of the mesh to the number of vertices of the same area in the full resolution mesh.



(a)	(b)	(c)
5,900 vertices	7,300 vertices	3,800 vertices
11,500 triangles	14,000 triangles	7,000 triangles
244 wire edges	320 wire edges	500 wire edges

Figure 5.27: The bicycle model described at variable resolution (a) wheel at full resolution, remaining parts at $\frac{1}{23}$ of full resolution; (b) paddle at full resolution, remaining parts at $\frac{1}{11}$ of full resolution; (c) handle at full resolution, remaining parts at $\frac{1}{26}$ of full resolution

5.6 Summary

In this Chapter, we studied the problem of modeling non-manifold simplicial shapes cost-effectively. We proposed two dimension-independent data structures, namely, the Simplified Incidence Graph (SIG) and the Incidence Simplicial (IS) data structure. These data structures cater to the need of applications which require the

explicit encoding of all simplicial entities. At the same time, these data structures optimize on the storage and navigation costs. Update operators have been developed for both data structures. We also case-studied the SIG in the application of Non-manifold Multi-Tesselation. Both data structures and the update operators on them have been implemented.

The work on SIG has been published in [12]. In conjunction with the modification operator (see Section 3.3.1, the SIG has been developed successfully into a multi-resolution modeling application, the non-manifold multi-tesselation, see [17]. The IS data structure has been discussed in [16] and is in the process for publication. It has been observed in the Solid Modeling community that there is a real demand for such cost-efficient data structures which encode all simplexes of a complex.

Chapter 6

Two Data Structures for Non-manifold Tetrahedral 3D Shapes

It is not uncommon that a model contains parts of mixed dimensions and with non-manifold connectivites. No compact topological data structure exists in the literature which describes 3D shapes with mixed dimensions and non-manifold connectivities. While dimension-independent data structures, such as those described in Chapter 5, are able to capture the topology of such a model with integrity, such data structures may not be optimized in storage cost.

Storage cost is strongly related to the amount of topological information encoded in a specific data structure. In general, the more the information encoded, the larger the size of the data structure. For applications on small- to medium-sized data, it is possible to trade off storage cost with the ease of access to the encoded information. However, for applications that work on models described by hundreds of millions of cells, storage cost is a key factor on the usability of a data structure.

In handling huge models, it is essential that the representation captures the intrigity of the shape while trading off the explicit description of some information

for less space consumption. One approach to optimization which is commonly used in data exchange is the "soup of top simplexes". The soup of top simplexes approach barely captures all the top simplexes that form the model and contains no topological information regarding the connectivities among the top simplexes. Thus such a representation has little use apart from data transmission. But it serves as the basis on which highly compact topological data structures may be built. In this Chapter, we propose two highly compact data structures specialized on 3D simplicial shapes of mixed dimensions and non-manifold connectivities.

The Indexed Data Structure with Adjacencies (NMIA) presented in Section 6.1 is a highly compact data structure for tetrahedral meshes with non-manifold singularities. The NMIA data structure is especially suitable for applications which perform shape modification on huge models. We also discuss update operation on the NMIA. The Double-Level Decomposition (DLD) Data structure presented in Section 6.2 is designed from the decomposition approach which breaks a non-manifold 3D simplicial shape into parts of uniform dimenion and simpler topology.

6.1 Non-Manifold Indexed Data Structure with Adjacencies (NMIA)[1]

We have designed the *Non-Manifold Indexed data structure with Adjacencies (NMIA)* [13] in order to capture all the 3D non-manifold singularities succintly. The NMIA data structure is a non-manifold extension of the EIA data structure. The ex-

[1]Originally published in [13] Copyright ©2003 Eurgraphics.

tension is performed by encoding the various connected components at non-manifold vertices and non-manifold edges. For the purpose of compactness, it encodes only the vertices, and all the top simplexes of a simplicial 3-complex. A minimal set of topological relations is encoded to ensure that all other topological relations are retrievable. The following are the topological relations encoded:

- For each tetrahedron t:
  - relation $R_{3,0}(t)$, which consists of all the vertices of t;
  - relation $R_{3,3}(t)$, which consists of all the tetrahedra sharing a face with t;
- For each dangling-face f (recall from Section 3.1 that a dangling-face is a top 2-simplex):
  - relation $R_{2,0}(f)$, which consists of all the vertices of f;
- For each wire-edge w (recall from Section 3.1 that a wire-edge is a top 1-simplex): relation $R_{1,0}(w)$, which consists of all the vertices of w;
- For each non-manifold edge e:
  - partial relation $R^*_{1,2}$, which consists of all the dangling faces incident at e;
  - partial relation $R^*_{1,3}$, which consists of one tetrahedron from each fan of tetrahedra (i.e., a linearly-sortable 2-connected set of tetrahedra) incident at e;

- For each vertex v:
  - partial relation $R^*_{0,1}(v)$, which consist of all the wire edges incident at v;
  - partial relation $R^*_{0,2}(v)$, which consists of one dangling-face from each 2D connected component (that is a component of dangling faces) in the star of v;
  - partial relation $R^*_{0,3}(v)$, which consists of one tetrahedron from each 3D connect component (that is a component of tetrahedra possibly mixed with dangling-faces) in the star of v.

Vertex-based $R^*_{0,q}$ (for $q = 1, 2, 3$) relations associate each vertex v with one q-dimensional connected component in $st(v)$. The edge-based relations $R^*_{1,2}$ and $R^*_{1,3}$ are encoded at the simplexes that are incident at the non-manifold edges in the following manner:

- at a non-manifold edge e of a dangling-face df, the top simplexes (dangling-faces or tetrahedra) following and preceding df around e are encoded. For example in Figure 6.1, tetrahedra t_1 and t_4 are encoded at edge e of dangling-face df;
- at a non-manifold edge e of a tetrahedron t, the top simplexes (dangling-faces or tetrahedra) following and preceding t around e and sharing only edge e with t are encoded (if these exist). For example in Figure 6.1, only tetrahedron t_4 is encoded at edge e of tetrahedron t_3.

This results in an implicit encoding of relation $R^*_{1,2}$ and $R^*_{1,3}$.

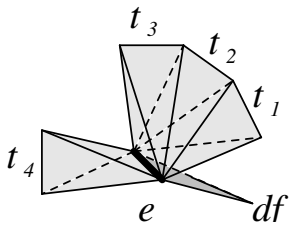


Figure 6.1: Encoding of relations at an implicitly encoded non-manifold edge e

6.1.1 Implementation and Storage Cost

In this Section, we describe the implementation details of the NMIA data structure and its storage cost in terms of the number of pieces of information encoded. The indices of the simplexes are integer. We assume that each memory slot may by interpreted as either an integer or a pointer, depending on the context. Flags are encoded by bits.

For each entity (vertex, wire-edge, dangling-face and tetrahedron), we store a *visit flag*. This flag is used by traversal algorithms to mark a simplex as having been visited. The flags of the visited entities have to be reset after each traversal is completed.

All boundary relations are implemented as fix-sized arrays.

Relation $R_{3,3}$ is implemented as a fix-sized array with one bit-flag for every element to indicate if the corresponding face-adjacent tetrahedron is present or not. When the adjacent tetrahedron is absent, the corresponding array position is used as a pointer to a fix-sized array which encodes $R^*_{1,2}$ and $R^*_{1,3}$ at each edge of the corresponding face f of t, encoding one element per edge. A 3-bit flag encodes

which edges of face f of t are non-manifold edges. When all three edges of f are manifold edges, the array position contains value -1.

For dangling-faces, partial relations $R^*_{1,2}$ and $R^*_{1,3}$ at each edge of dangling-face df is encoded by a variable-sized array with up to two elements for each edge of df. Three flags are used to indicate whether each edge of df is manifold.

For partial co-boundary relations at vertices, we encode an index I and a flag for each vertex v. The flag indicates whether v is manifold or not. If v is manifold, the index I points to one tetrahedron in the star of v. Otherwise, I points to an array of three pointers. Each pointer points to a variable array encoding $R^*_{0,1}, R^*_{0,2}, R^*_{0,3}$ respectively.

Let n_3, n^*_2, n^*_1, n_0 denote the number of tetrahedra, dangling-faces, wire-edges and vertices respectively. let us define C_e to be the total number of connected components summed over all the non-manifold edges, C_v to be the total number of connected components in all stars of vertices.

For this implementation, the storage cost can be estimated as follows:

- $R_{3,0}$: $4n_3$ integers
- $R_{3,3}$: $4n_3$ integers and $4n_3$ bits
- $R^*_{1,3}$ and $R^*_{1,2}$: $2C_e$ integers and $12n_3 + 3n^t_2$ bits
- $R_{2,0}$: $3n^t_2$ integers
- $R_{1,0}$: $2n^t_1$ integers
- $R^*_{0,1}, R^*_{0,2}, R^*_{0,3}$: C_v integers and n_0 bits

The total storage cost is $8n_3 + 3n_2^t + 2n_1^t + 2C_e + C_v$ integers and $16n_3 + 3n_2^t + n_0$ bits. For the case of encoding a regular object, $n_2^t = n_1^t = 0$ and the cost is reduced to $8n_3 + 2C_e + C_v$ integers and $16n_3 + n_0$ bits. Moreover, when the complex encoded has a 3-manifold domain, $n_2^t = n_1^t = C_e = 0$ and $C_v = n_0$. Thus the cost is $8n_3 + n_0$ integers and $16n_3 + n_0$ bits. Thus, the NMIA has a very high scalability. When it encodes a manifold simplicial 3-complex, its overhead is only $16n_3 + n_0$ bits compared with the EIA.

Non-manifold	n_0	n_3	n_1^t	n_2^t	C_e	C_v	Storage Cost
Teapot	4,658	5,666	2,944	3,930	144	10,617	73,911 int 107,104 bits
Balloon	1,108	856	64	1,632	0	1,268	13,140 int 19,700 bits
Regular	n_0	n_3	$n_1^t = 0$	$n_2^t = 0$	C_e	C_v	Storage Cost
Cubic	4,823	14,503	-	-	160	4,823	121,167 int 236,871 bits
Dough	3,097	8,509	-	-	256	3,097	71,681 int 139,241 bits
Manifold	n_0	n_3	$n_1^t = 0$	$n_2^t = 0$	$C_e = 0$	$C_v = n_0$	Storage Cost
Gargoyle	2.73k	10.0k	-	-	-	-	82.73 int 162.73 bits
Rings	2.52k	8.13k	-	-	-	-	67.56 int 132.6 bits

Table 6.1: An evaluation of the implementation-specific storage cost of the NMIA data structure on non-manifold, regular and manifold data sets

6.1.2 Retrieval of Topological Relations

In this Section, we discuss how topological relations can be retrieved in optimal, or almost optimal, time from the NMIA data structure. 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 $R_{3,2}(t)$ 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 $R_{2,1}(f)$, 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 $R_{2,1}(f)$ are again specified as pairs of vertices. Thus, relation $R_{2,1}(f)$ 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 $R_{1,2}(e)$ and $R_{1,3}(e)$ involves navigating around an edge e. Such navigation can be described by Algorithm 3. The simplexes of the desired dimensions can be extracted during such navigation.

We illustrate through the example in Figure 6.2 how to retrieve both $R_{1,2}(e)$ and $R_{1,3}(e)$ relations, i.e., how to navigate around an edge e in counter-clockwise direction. To this aim, we perform the following steps:

1. We start with e being an edge of t_3.

2. Using $R^*_{1,3}(t_3, e)$, we retrieve the left and right neighbors of t_3 with respect to

Algorithm 3 RetrieveSimplexesAtEdge(σ, e)

Require: σ be a tetrahedron or dangling-face, and e be an edge of σ

1: σ':=σ
2: done:=false
3: **repeat**
4: **if** $dim(\sigma') = 3$ **then**
5: Retrieve σ'', which follows σ' around edge e, from $R^*1,2(\sigma, e)$, $R^*1,3(\sigma, e)$ or $R_{3,3}(\sigma)$.
6: **else if** $dim(\sigma') = 2$ **then**
7: Retrieve σ'', which follows σ' around edge e, from $R^*1,2(\sigma, e)$ or $R^*1,3(\sigma, e)$
8: **end if**
9: σ':=σ''
10: **until** $\sigma = \sigma'$

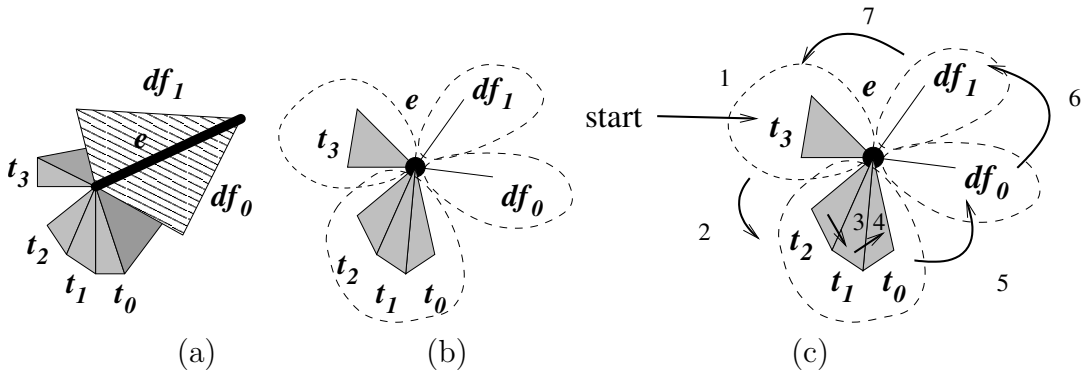


Figure 6.2: (a) shows a non-manifold edge e; (b) shows the same edge viewed perpendicularly (c) shows the steps of navigating through the entities incident at the non-manifold edge e

e. These are df_1 and t_2, respectively. Since we choose to go around e in the counter-clockwise direction, we move to t_2.

3. At t_2, relation $R_{3,3}(t_2)$ allows us to extract all the tetrahedra which are face-adjacent to t_2. By using $R_{3,0}(t_2)$, we can select t_1 as the only tetrahedron that is also incident at e. Then, we move to t_1.

4. At t_1, again we use the $R_{3,3}(t_1)$ to retrieve the tetrahedra which are face-adjacent to t_1 and $R_{3,0}(t_1)$ to select those incident at e. This gives t_0 and t_2,

and thus we move to t_0.

5. Tetrahedron t_0 has only one face-adjacent neighbour, namely t_1 which has been visited. The top simplex following t_0 around e is encoded as $R^*_{1,2}(t_0, e)$. From $R^*_{1,2}(t_0, e)$ we retrieve the right neighbor of t_0, which is df_0. Thus, we move to df_0.

6. The neighbours of df at e are encoded as $R^*_{1,2}(df_0, e)$. From $R^*_{1,2}(df_0, e)$ we retrieve the right neighbor of df_0, which is df_1. Thus, we move to df_1.

7. From $R^*_{1,2}(df_1, e)$ we retrieve the right neighbor of df_1, which is t_3. Thus, we are done, since we started with t_3.

Retrieving relation $R_{1,2}(e)$ takes a time linear in the number of faces incident at e, i.e., it can be performed in optimal time. Retrieving relation $R_{1,3}(e)$ takes $O(|R_{1,2}(e)|)$ time, where $|R_{1,2}(e)|$ denotes the number of faces incident at e, because of the possible presence of dangling-faces. Relation $R_{2,2}(f)$ is basically retrieved in the same way as relation $R_{1,2}(e)$, one for each edge of face f. Therefore, it takes $O(|R_{2,2}(f)|)$ time, which is optimal.

Top simplexes incident at a vertex v must be retrieved by traversing the star of v through relation $R^*_{0,q}$, for $q = 1, 2, 3$. Retrieving relations $R_{00}(v)$, $R_{01}(v)$, $R_{02}(v)$, and $R_{03}(v)$ requires a breadth-first search over all the connected components incident at vertex v. Within each component, entities that are incident at v are traversed by using R_{33}, $R_{3,0}$, $R^*_{1,3}$, and $R^*_{1,2}$ relations. Here we show the algorithm for retrieving $R_{0,3}$.

Algorithm 4 RetrieveTetrahedraAtVertex(v)

```
1: for each σ in R_{0,3}(σ) do
2:    visit(σ)
3:    enqueue(Q, σ)
4:    while not Empty (Q) do
5:       σ = dequeue(Q)
6:       if dim(σ') = 2 then
7:          retrieve all edges of σ through R_{2,0}(σ)
8:       else
9:          retrieve all edges of σ through R_{3,0}(σ)
10:      end if
11:      for each edge e retrieved that is incident at v do
12:         retrieve all top simplexes incident at e
13:         for each top simplex σ' incident at e do
14:            if not visited σ' then
15:               visit(σ')
16:               enqueue(Q, σ')
17:            end if
18:         end for
19:      end for
20:   end while
21: end for
```

6.1.3 Performing an Edge Collapse on the NMIA Data Structure[2]

In this Section, we address how the NMIA data structure can be useful for applications that need to perform updates on 3D models. Specifically, we want to study how the NMIA data structure can support the elementary mesh update operator *edge collapse*, and its reverse operation, *vertex split*. The general principles of the elementary edge collapse and vertex split on non-manifold 3D shapes have been discussed in Section 3.3.3. In this Section, we address how an edge collapse may

[2]Originally published in [14]. Reproduced with notice of ACM copyright: permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

be performed on the NMIA data structure so that the neighbourhood connectivity of the shape is captured in the data structure. Vertex split and its encoding are addressed in the next two Sections.

Performing the collapse of an edge $e = (v_1, v_2)$ into a vertex v in a complex Σ requires specifying just edge e and vertex v. $R_{k,0}$ relation is affected for all k-simplexes in $st(v_1) \cup st(v_2)$. $R_{3,3}$, $R^*_{1,3}$, $R^*_{1,2}$ and $R^*_{0,q}$ (for $q = 1, 2, 3$) relations are affected for all simplexes belonging to $st(v_1) \cup st(v_2)$, or adjacent to simplexes in $st(v_1) \cup st(v_2)$. In this Section, we discuss in detail the different situations which can arise and show how the entities and the relations in the NMIA data structure have to be updated. Based on this analysis, we present an algorithm for performing edge collapse.

In Section 3.3.3, we enumerated the three cases that may occur when edge e is collapsed. We summarize them as follows:

1. $\sigma \in st(e)$: in this case, σ can either be a 2-simplex or a 3-simplex.

2. $\sigma \in st(v_1)$, $\sigma \notin st(v_2)$ and there exists $\overline{\sigma} \in st(v_2)$ such that $\sigma \cap \overline{\sigma} \neq \emptyset$: in this case, σ is incident at v_1 but not in v_2, and there exists a simplex $\overline{\sigma}$ incident at v_2 which intersects σ. This case also includes the symmetric situation in which $\sigma \in st(v_2)$ and $\sigma \notin st(v_1)$ and there exists $\overline{\sigma} \in st(v_1)$ such that $\sigma \cap \overline{\sigma} \neq \emptyset$.

3. $\sigma \in st(v_1)$, $\sigma \notin st(v_2)$ and $\sigma \cap \overline{\sigma} = \emptyset$, for every $\overline{\sigma} \in st(v_2)$: in this case, σ is incident at v_1 and not in v_2 and it does not have any intersection with simplexes incident at v_2. This case also includes the symmetric situation in which $\sigma \in st(v_2)$ and $\sigma \notin st(v_1)$ and $\forall \overline{\sigma} \in st(v_1)$, $\sigma \cap \overline{\sigma} = \emptyset$.

Case 1, in which a k-simplex is reduced to a $(k-1)$-simplex and case 2, in which two k-simplexes may be merged into a single one, require more attention. We enumerate here the different situations which may arise in these two cases. This will also help us encoding the inverse of edge collapse, vertex split.

To this aim, we consider $L = link(v_1) \cap link(v_2)$. L is a collection of vertices and edges which can be ordered clockwise or counter-clockwise around edge $e = (v_1, v_2)$. If e is a manifold edge, then L is homeomorphic to a circle or to a portion of a circle. If e is a non-manifold edge, then L is composed of several connected components, one for each connected component incident at e. Each component may consist of an isolated vertex or of a chain of edges. Figure 6.3 gives an example of $L = link(v_1) \cap link(v_2)$. Edge $e = (v_1, v_2)$ is the edge to be collapsed. $st(v_1)$ is composed of tetrahedra t_1, t_2, t_4 and of all their faces. $link(v_1)$ consists of the simplexes in $st(v_1)$ that are not incident at v_1, namely faces f_1 and f_3 with all their boundaries. $st(v_2)$ is composed of tetrahedra t_1, t_3, t_5 and of all their faces. $link(v_2)$ is defined similarly to $link(v_1)$, and consists of faces f_2 and f_4 with all their boundaries. Thus L consists of one edge, namely, edge (u_1, u_2).

Let us consider $\omega_i \in L$: ω_i can be a vertex, u, or an edge (u_1, u_2). We denote with $l(\omega_i)$ the set of simplexes incident at ω_i, i.e. in $st(\omega_i)$, and incident at either v_1 or v_2 but not in both. In the example of Figure 6.3, L has only one component, namely edge (u_1, u_2), which we call ω_0. Thus, $l(\omega_0) = \{t_2, t_3\}$. We denote with $c(\omega_i)$ the set of simplexes incident at ω_i, i.e., in $st(\omega_i)$, and in both v_1 and v_2. In the example of Figure 6.3, $c(\omega_0) = \{t_1\}$.

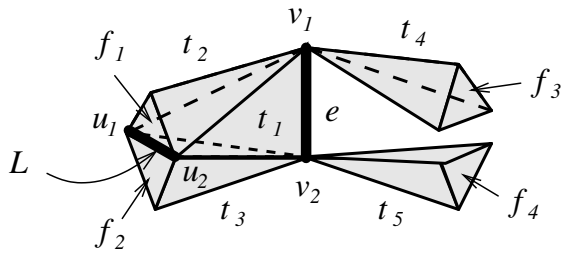


Figure 6.3: An illustration for $L = link(v_1) \cap link(v_2)$: $link(v_1)$ is composed of faces f_1, f_3 and of all their boundaries. Similarly, $link(v_2)$ is composed of faces f_2, f_4, and of their boundaries. Therefore, L is composed of edge (u_1, u_2). Let $\omega_0 = \{t_2, t_3\}$. Then, $l(\omega_0) = \{t_2, t_3\}$, and $c(\omega_0) = \{t_1\}$.

We consider a simplex $\sigma \in c(\omega_i)$, which can be either a tetrahedron, or a dangling-face, and two simplexes σ_1 and σ_2, incident at v_1 and v_2, respectively, and belonging to $l(\omega_i)$. Note that σ_1 and σ_2 can be tetrahedra, dangling-faces or wire-edges. This latter case is possible only when ω_i is a vertex.

The possible combinations of σ_1, σ and σ_2 are reported in Table 6.2. The first eight cases apply when ω_i is an edge. The other eight cases apply when ω_i is a vertex. In the columns corresponding to σ_1, σ and σ_2, the symbol T, F, E or $\emptyset$, means that the corresponding simplex is a tetrahedron, a dangling-face, a wire-edge or is non-existent, respectively. When ω_i is an edge, σ is always bounded by four vertices $\{v_1, v_2, \omega_i\}$ and can be either a tetrahedron or a hole. In both cases, it shares face $\{v_1, \omega_i\}$ with σ_1 and face $\{v_2, \omega_i\}$ with σ_2. If σ is a tetrahedron, it undergoes a dimension-reduction transformation into face $\{v, \omega_i\}$. In all cases, simplexes $\{v_1, \omega_i\}$ and $\{v_2, \omega_i\}$ are merged into simplex $\{v, \omega_i\}$. In the NMIA data structure, we are only interested in the case in which $\{v, \omega_i\}$ is a dangling-face, since we do not encode the faces bounding a tetrahedron explicitly.

When ω_i is a vertex, σ can be either a dangling-face or a hole, but, in both cases, it is bounded by three vertices $\{v_1, v_2, \omega_i\}$. Simplexes σ_1 and σ_2 share edges $\{v_1, \omega_i\}$ and $\{v_2, \omega_i\}$ with σ, respectively. Note that, in general, there are several σ_1 and σ_2 in $l(\omega_i)$, except when σ_1 or σ_2 are wire-edges: in this case, they are completely defined by $\{v_1, \omega_i\}$ or $\{v_2, \omega_i\}$. Moreover, either σ_1 or σ_2 can be a wire-edge only if σ is empty.

ω_i is an edge (u_1, u_2)				ω_i is a vertex u			
Case	σ_1	σ	σ_2	Case	σ_1	σ	σ_2
1	$\emptyset$	T	$\emptyset$	9	$\emptyset$	F	$\emptyset$
2	T	T	$\emptyset$	10	T/F	F	$\emptyset$
3	$\emptyset$	T	T	11	$\emptyset$	F	T/F
4	T	T	T	12	T/F	F	T/F
5	F	$\emptyset$	F	13	E	$\emptyset$	E
6	T	$\emptyset$	F	14	T/F	$\emptyset$	E
7	F	$\emptyset$	T	15	E	$\emptyset$	T/F
8	T	$\emptyset$	T	16	T/F	$\emptyset$	T/F

Table 6.2: Cases for simplex σ in $c(\omega_i)$ and simplexes σ_1 and σ_2 in $l(\omega_i)$. T = tetrahedron, F = dangling-face, E = wire-edge, $\emptyset$ stands for the absence of a simplex.

Examples of cases 1, 2, 4, 9, 10 and 12 in Table 6.2 are shown in Figures 3.13(a) to 3.13(f). The examples in Figures 3.14(a) to 3.14(f) are instances of cases 5, 6, 8, 13, 14 and 16, respectively. Cases 3, 7, 11 and 15 are symmetric to cases 2, 6, 10 and 14 respectively.

In what follows, we examine how the entities and the relations are affected in the sixteen cases shown in Table 6.2. We consider L as an ordered sequence of elements ω_i, where ω_i is either a vertex or an edge. We denote as ω_{i-1} and ω_{i+1},

respectively, the predecessor and the successor of ω_i along L. Note that ω_i does not need to be connected to either ω_{i-1} or ω_{i+1}. Also, ω_i can be an isolated vertex, or a vertex which is common to the two edges ω_{i-1} and ω_{i+1}, or an extreme vertex of either ω_{i-1} or ω_{i+1}. These two latter cases occur when σ is a dangling-face $\{u, v_1, v_2\}$ and $\{\omega_{i-1}, v_1, v_2\}$ or $\{\omega_{i+1}, v_1, v_2\}$, or both, define an empty tetrahedral hole. Thus, similarly to isolated vertices, we consider such a vertex as a separate element of L and not as an extreme vertex of an edge.

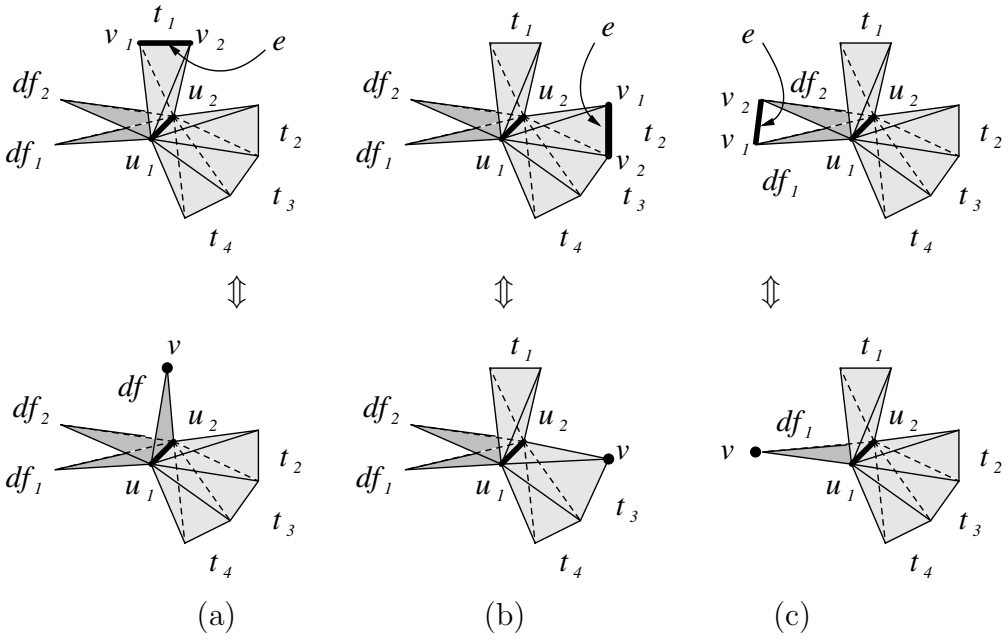


Figure 6.4: Examples of the updates needed at edge (u_1, u_2) after the collapse of edge $e = (v_1, v_2)$: In (a), tetrahedron t_1 becomes a dangling-face df. $R^*_{1,2}(df, e')$ at edge $e' = (u_1, u_2)$ consists of dangling-face df_2 and tetrahedron t_2. The adjacency relations of the original neighbors of t_1 at edge (u_1, u_2), namely df_2 and t_2, need to be updated. df replaces t_1 in $R^*_{1,2}(df_2, e')$ and $R^*_{1,3}(t_2, e')$. In (b), after edge e is collapsed, tetrahedron t_2 becomes a boundary face of tetrahedron t_3, and tetrahedra t_1 and t_3 become immediate neighbors of each other at edge (u_1, u_2). Therefore, t_2 is removed from $R_{3,3}(t_3)$ relation and t_1 is added to $R^*_{1,3}(t_3, e')$. t_3 replaces t_2 in $R^*_{1,3}(t_1, e')$. In (c), dangling-face df_2 is merged into dangling-face df_1. df_1 and tetrahedron t_4 become immediate neighbors of each other. So t_4 replaces df_2 in $R^*_{1,2}(df_1, e')$, and df_1 replaces df_2 in $R^*_{1,3}(t_4, e')$.

When $\omega_i = (u_1, u_2)$ is an edge, (which holds for cases 1 to 8 in Table 6.2,) the collapse of edge $e = (v_1, v_2)$ may cause the 1- and 2-adjacency (i.e., $R^*_{1,2}$, $R^*_{1,3}$ and $R_{3,3}$) relations to change for simplexes incident at edge (u_1, u_2). These relations need to be updated. The three examples in Figure 6.4 illustrate how the update is done at (u_1, u_2). In each example, the left part illustrates the situation before edge collapse, and the right part after. After edge collapse, adjacency relations ($R^*_{1,3}$ or $R^*_{1,2}$) at the new edges (u_1, v) and (u_2, v) are updated in a similar fashion. Special attention has to be given, however, when $\omega_i = (u_1, u_2)$ is connected to ω_{i-1} or to ω_{i+1} because neighboring simplexes may also be merged due to the collapse of edge e. Figure 6.5(a) illustrates a situation in which ω_i is not connected to its predessesor or successor, and Figure 6.5(b), a situation in which ω_i is connected to its predessesor. In Figure 6.5(a), tetrahedron t_1 is incident at the edge $e = (v_1, v_2)$ to be collapsed. Let ω_0 be edge (u_1, u_2). ω_0 is not connected to any other components of L since it is the only component of L. Dangling face df_1 is the immediate neighbor of t_1 at edge (u_1, v_1), and dangling-face df_2 is the immediate neighbor of t_1 at edge (u_1, v_2). After edge collapse, df_1 and df_2 become 1-adjacent at the new edge, (u_1, v). In Figure 6.5(b), the edge to be collapsed is $e = (v_1, v_2)$. L has two components, namely (u_3, u_1) and (u_1, u_2), which we call ω_0 and ω_1. Similar to the example of 6.5(a), before edge collapse, dangling-faces df_1 and df_2 are neighbors of t_1 at edges (u_1, v_1) and (u_1, v_2), respectively. After edge collapse, df_2 is merged to df_1.

When $\omega_i = u$ is a vertex, (cases 9 to 16 in Table 6.2,) the collapse of edge $e = (v_1, v_2)$ causes two 1-connect components to merge into one. $R^*_{0,q}$ $(q = 1, 2, 3)$

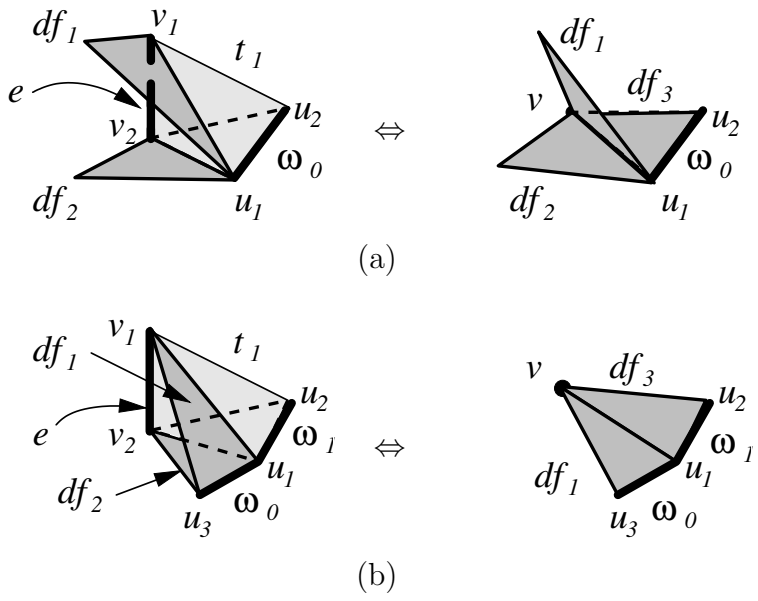


Figure 6.5: Two examples to show the update of the connected components at edge (u_1, v) after an edge collapse operation. In (a), tetrahedron t_1 becomes a dangling-face df_3. Dangling faces df_1 and df_2, which are neighbors of t_1 at edge (u_1, v_1) and edge (u_1, v_2), respectively, become neighbors of each other at the new edge $e' = (u_1, v)$ Therefore, $R^*_{1,2}(df_3, e')$ consists of df_1 and df_2. df_3 replaces t_1 in both $R^*_{1,2}(df_1, e')$ and $R^*_{1,2}(df_2, e')$. In (b), after edge collapse, tetrahedron t_1 becomes dangling-face df_3, and dangling-face df_1 is merged into dangling-face df_2. So $R^*_{1,2}(df_3, e')$ consists of only df_1 and $R^*_{1,2}(df_1, e')$ consists of only df_3.

at the new vertex v needs to be defined. We need to update $R^*_{1,3}$ or $R^*_{1,2}$ relations at edge (u, v). These updates are completely similar to those done for simplexes incident at edge (u_1, v) and (u_2, v) for the case where ω_i is an edge.

We summarize now the various cases in an edge collapse algorithm. Let Σ be the given complex. Let $e = (v_1, v_2)$ be the edge to be collapsed into a vertex v. Recall that $L = link(v_1) \cap link(v_2)$. We denote with Ω_{L1} the set of top simplexes (tetrahedra, dangling-faces and wire-edge) σ such that σ is incident at v_1 and in some entity of $link(v_1) - L$, and with Ω_{L2} the set of simplexes incident at v_2 and in

some entity of $link(v_2) - L$.

The *Edge Collapse algorithm* performs the following steps:

Step 1: Compute L, Ω_{L1}, Ω_{L2}, and, for each $\omega_i \in L$, $l(\omega_i)$ and $c(\omega_i)$ as follows:

1. Compute $st(v_1)$ and $st(v_2)$ by using $R^*_{0,q}(v_1)$ and $R^*_{0,q}(v_2)$ $(q = 1, 2, 3)$, respectively.
2. Compute $link(v_1)$ and $link(v_2)$ from the boundary relations of the entities in $st(v_1)$ and $st(v_2)$.
3. Compute $L = link(v_1) \cap link(v_2)$. If a face exists in L, the edge collapse operation is invalid.
4. For each $\omega_i \in L$, compute $l(\omega_i)$ and $c(\omega_i)$ by using $R^*_{0,q}$, $R^*_{1,2}$ and $R^*_{1,3}$ relations.
5. Compute Ω_{L1}, Ω_{L2} from L, $st(v_1)$ and $st(v_2)$.

Step 2: For each simplex $\omega_i \in L$:

If $\omega_i = (u_1, u_2)$ is an edge,

1. Perform validity check to ensure that the region $\{v_1, v_2, u_1, u_2\}$ is either a tetrahedron or is empty.
2. **Case 1:** mark σ as deleted, and create a new dangling-face σ' in Σ.

   Cases 2, 3 and 4: mark σ as deleted.

   Cases 5 and 6: mark σ_2 as deleted.

   Case 7: mark σ_1 as deleted.

If $\omega_i = u$ is a vertex,

1. Perform validity check to ensure that the region $\{v_1, v_2, u\}$ is either a dangling-face or is empty.

2. **Cases 9 to 12:** mark σ as deleted.

   Case 9: create new a wire-edge σ' in Σ,

   Cases 13 and 14: mark σ_2 as deleted.

   Case 15: mark σ_1 as deleted.

Step 3: For each simplex $\omega_i \in L$:

Update the relations affected at the neighborhood of ω_i, as described before.

Step 4: For each simplex $\sigma' \in \Omega_{L1}$, Update $R_{k,0}(\sigma')$ by replacing v_1 with v.

Step 5: For each simplex $\sigma' \in \Omega_{L2}$, Update $R_{k,0}(\sigma')$ by replacing v_2 with v.

Step 6: Update $R^*_{0,q}(v)$ $(q = 1, 2, 3)$. Delete all the marked simplexes.

The first step in the edge collapse algorithm involves the examination of the neighbourhood of the two extreme vertices, v_1 and v_2, of the collapsing edge e. The time complexity in the computation of the stars of v_1 and of v_2 is thus bounded by the size of $st(v_1) \cup st(v_2)$. The size of the links of v_1 and v_2 is also bounded by the size of the corresponding star of each vertex (a property of the relationship between the link and the star of a vertex in 3D space). The link of e is computable based on the links of v_1 and v_2. Thus the time complexity of this step is bounded by the size of $st(v_1) \cup st(v_2)$. Subsequent steps of the algorithm involve only the local neighbourhood within $st(v_1) \cup st(v_2)$. Thus the edge collapse operation has a time complexity that is bounded by $O(st(v_1) \cup st(v_2))$.

6.1.4 An Encoding Scheme for Vertex Split in NMIA

Edge collapse is often used in conjunction with its reverse operation *vertex split* (see Section 3.3.3). In this Section, we discuss an encoding of vertex split, which is computed at the time when an edge $e = (v_1, v_2)$ is collapsed into a new vertex v. The encoding of vertex split at v enables the expansion of v back into two vertices v_1 and v_2 sharing an edge e.

We describe a compact encoding scheme for a vertex split. The encoding scheme is composed of two parts: a labeling of the entities in the restricted star of v and an encoding of $L = link(v_1) \cap link(v_2)$ together with the cases discussed in Section 6.1.3. The labeling of the entities in the star of v allows us to modify all the simplexes which become incident at v_1 or v_2 to generate the new k-simplexes obtained by expanding $(k-1)$-simplexes and to duplicate simplexes. The encoding of L is necessary for encoding boundary faces and edges which are duplicated and expanded (since they are not described in the NMIA data structure), and for updating topological relations locally. For every k-simplex σ' in $st(v)$, we store a 2-bit code, $c_1(\sigma')$ for detecting whether σ' after the split becomes incident at v_1 ($c_1(\sigma') = 00$) or at v_2 ($c_1(\sigma') = 01$) or it is duplicated into two k-simplexes incident at v_1 and at v_2 respectively ($c_1(\sigma') = 10$), or it is expanded into a $(k+1)$-simplex incident at edge e ($c_1(\sigma') = 11$). A unique traversal of $st(v)$ is defined by following the order in which the representative simplexes are encoded in the $R^*_{0,q}$ $(q = 1, 2, 3)$ relation and a predefined traversal inside each connected component at the vertex.

For each element $\omega_i \in L$, we store:

- a 4-bit code $c_2(\omega_i)$ which encodes the sixteen cases shown in Table 6.2;
- 1-bit code $c_3(\omega_i)$ which indicates whether ω_i is connected to ω_{i-1} through vertex u. If ω_i is connected to ω_{i-1}, then vertex u is not encoded;
- one or two indexes of the vertices which define ω_i (only one vertex is encoded when ω_i is a vertex or is connected to ω_{i-1});
- other information which depend on the specific case according to Table 6.2:
  - **Case 1:** index of the dangling-face $\{u_1, u_2, v\}$, which becomes a tetrahedron $\{u_1, u_2, v_1, v_2\}$;
  - **Cases 2, 4, 6 and 8:** index of σ_1;
  - **Cases 3 and 7:** index of σ_2;
  - **Case 5:** index of the dangling-face $\{u_1, u_2, v\}$, which becomes two faces $\{u_1, u_2, v_1\}$ and $\{u_1, u_2, v_2\}$; a
  - **Case 9:** index of wire-edge (u, v), which becomes a dangling-face $\{u, v_1, v_2\}$;
  - **Cases 10 and 14:** index of σ_1;
  - **Cases 11 and 15:** index of σ_2;
  - **Cases 12 and 16:** index of σ_1 and σ_2.
  - **Case 13:** index of wire-edge (u, v), which becomes two wire-edges (u, v_1) and (u, v_2);

Figure 6.6 shows two examples of the encoding scheme. In the example of

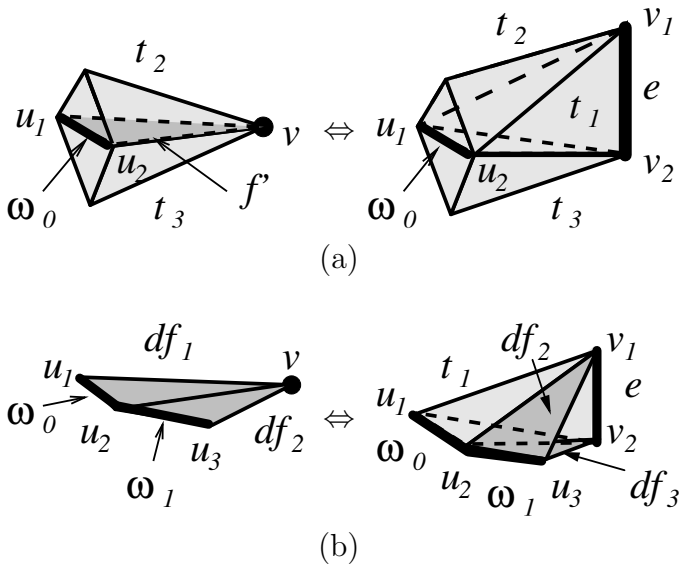


Figure 6.6: Two examples of encoding a vertex split: In (a), the encoding of L is *[4 0* u_1 u_2 t_2*]*. In (b), the encoding of L is *[1 0* u_1 u_2 df_1*; 5 1* u_3 df_2*]*.

Figure 6.6(a), L consists of just one element, namely (u_1, u_2), which we call ω_0. The encoding of L relative to ω_0 is *[4 0* u_1 u_2 t_2*]*. The first field is code c_2, meaning that we are in case 4. The second field indicates that the first vertex of ω_0 is not connected to the last vertex of ω_0. So, both vertices u_1 and u_2 are found in the next field, which is followed by tetrahedron t_2 incident at v_1. During vertex split, t_3 is retrieved from $R_{3,3}(t_2)$ relation, and, a new tetrahedron t_1 is created, which is incident at the new edge e.

In Figure 6.6(b), L consists of two elments, edges (u_1, u_2) and (u_2, u_3), that we call ω_0 and ω_1, respectively. The encoding of L is *[1 0* u_1 u_2 df_1*; 5 1* u_3 df_2*]*. ω_0 is in case 1. The value 0 in the next field indicates that the first vertex of ω_0 is not connected to the last vertex of ω_1. The simplex incident at ω_0 is df_1. During vertex split, df_1 is expanded into a tetrahedron incident at both v_1 and v_2. ω_1 is in

case 5. The value 1 in the next field means that first vertex of ω_1 is the same as the last vertex of ω_0. The second vertex of ω_1 is u_3, which is given in the third field. The last field gives the simplex, df_2, which is incident at ω_1. During vertex split, df_2 becomes incident at v_1, and a new dangling-face incident at v_2 is created.

6.1.5 Performing Vertex Split on the NMIA Data Structure

Vertex split is the reverse operation of edge collapse. The formulation of the vertex split operation has been addressed in Section 3.3. In this Section, we discuss how this operator is implemented on the NMIA data structure in conjunction with the encoding of it described in Section 6.1.4.

The algorithm for performing vertex split uses the encoding of the vertex split operation described in Section 6.1.4 and updates the simplicial complex Σ on which the vertex split is applied through the following steps:

Step 1: Compute $st(v)$ by retrieving $R_{01}(v)$ for wire-edges incident at v, $R_{02}(v)$ for dangling-faces incident at v and $R_{03}(v)$ relations from the partial relations $R^*_{0,k}(v)$.

Step 2: For each $\sigma' \in st(v)$,

- **If** $c_1(\sigma') = 00$ (σ' becomes incident at v_1),

  then update $R_{k,0}(\sigma'), k = 2$ or 3 by replacing v with v_1.

- **If** $c_1(\sigma') = 01$ (σ' becomes incident at v_2),

  then update $R_{k,0}(\sigma'), k = 2$ or 3 by replacing v with v_2.

- **If** $c_1(\sigma') = 10$ (σ' is duplicated into two simplexes),

  then replace k-simplex $\sigma', k = 1$ or 2, with two new k-simplexes σ_a and σ_b such that $R_{k,0}(\sigma_a)$ is obtained from $R_{k,0}(\sigma')$ by replacing v with v_1 and $R_{k,0}(\sigma_b)$ is obtained from $R_{k,0}(\sigma')$ by replacing v with v_2.

- **If** $c_1(\sigma') = 11$ (σ' is expanded into a $(k+1)$-simplex),

  then replace k-simplex $\sigma', k = 1$ or 2, with a new $(k+1)$-simplex σ such that $R_{k,0}(\sigma)$ is obtained from $R_{k,0}(\sigma')$ by replacing v with (v_1, v_2).

Step 3: For each $\omega_i \in L$:

Case 2: (σ_1 is already incident at v_1) A new tetrahedron $\sigma = \{u_1, u_2, v_1, v_2\}$ is created which shares face $\{u_1, u_2, v_1\}$ with σ_1.

Case 3: (σ_2 is already incident at v_2) A new tetrahedron $\sigma = \{u_1, u_2, v_1, v_2\}$ is created which shares face $\{u_1, u_2, v_2\}$ with σ_2.

Case 4: (σ_1 is already incident at v_1 and σ_2 is already incident at v_1) A new tetrahedron σ is created which shares faces $\{u_1, u_2, v_1\}$ and $\{u_1, u_2, v_2\}$ with σ_1 and σ_2, respectively.

Case 6: (σ_1 is already incident at v_1) A new dangling-face $\sigma_2 = \{u_1, u_2, v_2\}$ is created.

Case 7: (σ_2 is already incident at v_2) A new dangling-face $\sigma_1 = \{u_1, u_2, v_1\}$ is created.

Cases 10, 11 and 12: A new dangling-face $\sigma = \{u, v_1, v_2\}$ is created.

Case 14: (σ_1 is already incident at v_1) A new wire-edge $\sigma_2 = (u, v_2)$ is

created.

Case 15: (σ_2 is already incident at v_2) A new wire-edge $\sigma_1 = (u, v_1)$ is created.

All other cases: nothing to be done.

In all cases, define $R_{k,0}$ for each newly created k-simplex.

Step 4: Define adjacency and incidence relations for each newly created entity. Update relations for each simplex σ' incident at an element $\omega_i \in L$ and at v_1 or at v_2, and update relations for each neighbor of σ'. This step reverses the modifications to the adjacency and incidence relations encoded in the NMIA data structure performed in edge collapse (see Section 6.1.3).

Step 5: Compute the partial relations $R^*_{0,k}(v_1)$ and $R^*_{0,k}(v_2)$ for $k = 2, 3$.

The vertex split algorithm acts on the neighbourhood of the vertex v that is being splitted. The total number of simplexes visited or introduced is bounded by the size of the two stars of the resultant vertices v_1 and v_2.

6.2 Double-Level Decomposition (DLD) data structure[3]

In this Section, we approach the problem of non-manifold shape modeling along the decomposition approach, which considers a non-manifold shape as a collection of simpler parts, called *Initial Quasi-Manifolds (IQM)*, connected at non-manifold joints. The Initial Quasi-Manifold Decomposition has been proposed in

[3]Originally published in [47] Copyright ©2006 Eurgraphics.

[26]. We reviewed the IQM decomposition approach for general d-dimensional simplicial complexes in Section 4.2.2. In this Section, we consider the specialization of this approach to a Euclidean 3D simplicial complex. We propose an optimized data structure, that we called the Double-Level Decomposition (DLD) data structure, for 3D simplicial shapes based on this decomposition. A special feature of the DLD data structure is that it captures high-level information of the shape in terms of its non-manifold structure. As a result of this, the DLD is not suitable for shape modification based on local updates, unlike the NMIA data structure we proposed in Section 6.1.

6.2.1 Decomposition of a 3D Simplicial Complex

In this Section, we present an algorithm for computing the IQM decomposition of a 3D simplicial complex. In a 3D simplicial complex, non-manifold singularities may occur at edges and vertices. The IQM decomposition can be obtained by cutting the complex along all non-manifold vertices and non-manifold edges. For an efficient computation of such decomposition, we need: adjacency relations $R_{3,3}$ for all tetrahedra, adjacency relations $R_{2,2}$ for all dangling-faces, and the stars of all the vertices. The decomposition algorithm performs the following five steps, that are detailed in the rest of this Section:

1. Compute adjacency relation $R_{3,3}$ for all tetrahedra.

2. Compute adjacency relation $R_{2,2}$ for all dangling-faces.

3. Compute the stars of all vertices, where each star is described as the set of all

top simplexes incident at that vertex.

4. Identify non-manifold edges and non-manifold vertices through a traversal of the star of each vertex.

5. Decompose the complex at non-manifold simplexes and identify IQM components.

Step 1: Compute adjacency relation $R_{3,3}$ for all tetrahedra. An efficient way to compute it is to sort the tetrahedra by their four faces. It can be done as follows:

1. The faces of the tetrahedra are not explicit in the input. Each such face can be described through a 4-tuple (u_1, u_2, u_3, t), where u_1, u_2, u_3 are three vertices that describe one face of t, and are sorted in the increasing order of their indices. Each 4-tuple not only identifies a unique face, but also associates the face with a tetrahedron bounded by it. For each tetrahedron t four 4-tuples are created.

2. After sorting all the 4-tuples in lexicographical order, adjacent 4-tuples of the form (u_1, u_2, u_3, t_1) and (u_1, u_2, u_3, t_2) indicate that tetrahedra t_1 and t_2 are face-adjacent.

The time complexity for this step is $O(m_3 log(m_3))$, where m_3 denotes the number of tetrahedra in the complex.

Step 2: Compute adjacency relation $R_{2,2}$ for all dangling-faces. The technique is the same as the computation of relation $R_{3,3}$ for tetrahedra described above.

The complexity of this step is, thus, $O(d_2 log(d_2))$, where d_2 denotes the number of dangling-faces in the complex.

Step 3: Compute the stars of all vertices. This is performed as follows:

1. For each vertex v and each h, create empty sets, $b(v, h)$, which we call *buckets*, for collecting all the top simplexes of dimension h incident at v.

2. For each top h-simplex σ described by vertices $\{v_1, \cdots, v_{h+1}\}$, add σ to buckets $b(v_i, h)$, for $i = 1, \cdots, h + 1$.

This step is performed in time linear with respect to the number of vertices and the number of top simplexes, i.e., $O(v_0 + w_1 + d_2 + m_3)$, where v_0 is the number of vertices, w_1 the number of wire-edges, d_2 the number of dangling-faces and m_3 the number of tetrahedra.

Step 4: Identify non-manifold edges and non-manifold vertices. Non-manifold vertices and edges are identified through a traversal of the star of each vertex. This traversal is done by using the information stored in the buckets $b(v, h)$, plus the relations $R_{3,3}$ for tetrahedra, and $R_{2,2}$ for dangling-faces. During the traversal, the top simplexes in the star of v are grouped into densely $(h-1)$-connected components. Each component found is assigned a unique label, which we call the *component index*. All vertices (except for v) in a component C are labeled with the index of C. These labels are used for identifying non-manifold edges in the star of v. If a vertex u in the link of v has more than one label, then edge (u, v) is a non-manifold edge. If the star of v consists of more than one component, then v is a non-manifold vertex; it is a manifold vertex otherwise. Algorithm 5 provides a

pseudo-code description of the traversal strategy.

Algorithm 5 FindComponentsInStar(v, b)

```
1: j ← 1
2: for h from 3 downto 1 do
3:    while b(v, h) is not empty do
4:       Remove the unvisited top simplex σ from b(v, h)
5:       Create new component C_j for v
6:       Enqueue(Q, σ)
7:       while not empty(Q) do
8:          σ ← Dequeue(Q)
9:          C_j ← C_j ∪ σ
10:         for each γ in R_{h,h}(σ) do
11:            if the (h−1)-face between σ and γ is manifold and visited(γ)=0 then
12:               visited(γ) ← 1
13:               Enqueue(Q, γ)
14:            end if
15:         end for
16:      end while
17:      for each σ in C_j do
18:         Add label j to all vertices of σ (except v)
19:      end for
20:   end while
21: end for
```

We illustrate the labeling of the star of v through the example in Figure 6.7. In this example, the four tetrahedra form two densely 2-connected components and the three dangling-faces three densely 1-connected components in the star of vertex v. The vertices in the link of v are labeled according to the component(s) to which they belong, thus exposing the non-manifold edges in the star of v.

The traversal of the star of each vertex is a linear process with respect to the number of top simplexes in that star. The time complexity for Step 4 is thus $O(\sum_{v\in\Sigma}|st(v)|)$, where $|st(v)|$ denotes the size of the star of vertex v in Σ. Since each h-simplex belongs to the stars of exactly $h+1$ vertices, the time complexity

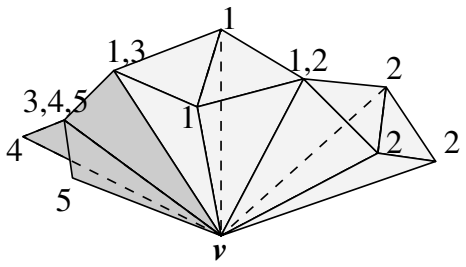


Figure 6.7: Labeling densely connected components in the star of vertex v

results to be linear in the number of top simplexes in the complex.

Step 5: Decompose non-manifold simplexes and identify IQM components. To complete the decomposition, the complex is cut at the non-manifold simplexes. For each non-manifold vertex v, one vertex copy v_i is created for each IQM component in the star of v. After the cutting, the whole complex is traversed once, but the traversal does not pass through non-manifold edges and non-manifold vertices. All tetrahedra that are 2-connected belong to the same IQM component. All the dangling-faces that are 1-connected form a separate manifold component. Likewise for all wire-edges that are 0-connected.

The star of each non-manifold vertex is partitioned when copies are created for the non-manifold vertex. The subsequent traversal of the whole complex takes linear time with respect to the size of the complex. Thus this step takes $O(\sum_{v_s \in \Sigma} |st(v_s)|) + O(v_0 + w_1 + d_2 + m_3 + k_0)$, where $|st(v_s)|$ denotes the size of the star of non-manifold vertex v_s in Σ, k_0 is the number of IQM components at all non-manifold vertices, and v_0, w_1, d_2, m_3 denote the number of vertices, wire-edges, dangling-faces and tetrahedra respectively.

Both Steps 1 and 2 involve sorting, while all the other steps perform operations that are linear in terms of the total number of top simplexes in the complex. For a typical 3-complex that is mostly 3-manifold with few dangling-faces and wire-edges, the time consumption of the decomposition is dominated by Step 1.

6.2.2 A Decomposition-based Data Structure for Simplicial 3-complexes

In this Section, we present the *Double-Level Decomposition (DLD) data structure*, which is based on the IQM decomposition and is generated through the algorithm described in Section 6.2.1. The DLD data structure is a two-layer representation in which the upper level describes the connectivity of the IQM components through their non-manifold simplexes, while the lower level describes the entities, their connectivity and adjacency relation inside the IQM components $C_1, \cdots, C_k$. This is similar in concept to the representation proposed in [26] for decomposed abstract simplicial complexes which is still a two-level data structure, but the description of the single IQM component is more complex since it may not necessarily be pseudo-manifold.

The connectivity of the components in the decomposition is represented as a hypergraph $G =< N, A >$, where N is a set of nodes representing the IQM components $C_1, \cdots, C_k$, and A is a set of hyperarcs. There are two kinds of hyperarcs: *hyperarcs of type vertex*, which represent non-manifold vertices, and *hyperarcs of type edge* which represent non-manifold edges. A hyperarc representing a non-manifold vertex v connects all components which contain copies of vertex v. Similarly, a hy-

perarc representing a non-manifold edge e connects all components which contain copies of edge e.

Figures 6.8(a)-(c) give an example of the IQM decomposition of a simple 3-complex and the hypergraph that represents the decomposition. The 3-complex shown in Figure 6.8(a) consists of two tetrahedra that share the non-manifold edge e which is incident at non-manifold vertices u and v, and two wire-edges that are incident at vertex v. The decomposition of this complex consists of four components: C_1 and C_2 are the two tetrahedra, C_3 and C_4 are the wire-edges. Figure 6.8(b) shows all the components of the decomposition. The non-manifold edge e and non-manifold vertices u and v are copied for each component. Figure 6.8(c) is a full description of the decomposition graph G. The nodes are $C_1, \cdots, C_4$ and the hyperarcs are e, u and v. In the hypergraph, the solid lines connecting C_i and the hyperarcs are the copies of the non-manifold joints. The dashed lines between u, v and e indicate their incidence.

All non-manifold singularities are thus explicitly represented only in the upper level, which encodes hypergraph G. The following information are encoded:

- For each node representing IQM component C_i:
  - dimension of the component;
  - and a pointer to one top simplex in this component.
- For each hyperarc representing non-manifold edge e: (We consider hyperarc e in Figure 6.8(c) to illustrate the following)

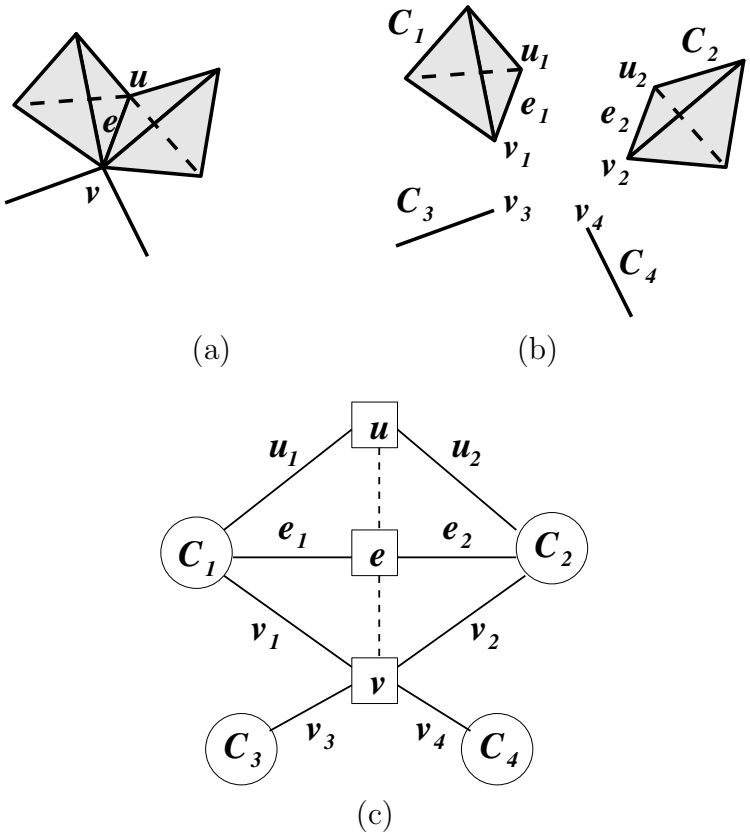


Figure 6.8: (a) a 3D complex; (b) its IQM decomposition; and (c) the hypergraph describing the decomposition

  - a pointer each to its extreme vertices, which are hyperarcs in G (in our illustration, they are hyperarcs u and v for e);
  - two lists of pointers: each pointer references a representative top simplex for each 2D or 3D IQM component in the star of e. One list collects the 2D representatives and the other the 3D representatives (in our illustration, the 2D list of e is empty while its 3D list consists of the two tetrahedra in Figure 6.8(b)).

- For each hyperarc representing non-manifold vertex v: (see hyperarc v in Figure 6.8(c) as an example)

- A list of pointers, each to the vertex copy of v in each IQM component in the star of v, (for the example, the list of v consists of copies $v_1, \cdots, v_4$);
- a list of pointers, one for each non-manifold edge in graph G, that is incident at v (for the same example, the list of v contains hyperarc e).

The lower level describes the IQM components. For any h-dimensional IQM component C_i, we use the Indexed data structure with Adjacencies which encodes all the h-simplexes, the vertices and the following relations:

- For each vertex v in the component, relation $R^*_{0,h}(v)$ which consists of one h-simplex in the star of v;
- For each h-simplex σ of C_i, relation $R_{h,0}(\sigma)$ and relation $R_{h,h}(\sigma)$

The low level data structure is implemented through the following constructs:

- For each vertex v in component C_i:
  - A 1-bit flag to indicate whether v is manifold;
  - One pointer for relation $R^*_{0,h}(v)$;
- For each wire-edge σ of C_i: a pointer array of size 2 for relation $R_{1,0}(\sigma)$
- For each top h-simplex σ of C_i, $h > 1$:
  - A pointer array of size $(h+1)$ for relation $R_{h,0}(\sigma)$
  - A pointer array of size $(h+1)$ for relation $R_{h,h}(\sigma)$
  - A bit flag of size $\binom{h+1}{1}$ to indicate whether each edge of σ is manifold;

- A hash table H_v that stores the pointers from the vertex copies to the node that corresponds to v in graph G.

The storage cost required by the DLD data structure can be evaluated in terms of the following quantities:

m_0 : number of manifold vertices;

n_0 : number of non-manifold vertices;

k_0 : total number of IQM components at all non-manifold vertices;

n_1 : number of non-manifold edges;

w_1 : number of wire-edges;

k_1 : total number of IQM components at all non-manifold edges;

d_2 : number of dangling-faces;

m_3 : number of tetrahedra;

C : total number of IQM components in the whole complex.

In the lower level data structure, the total number of vertices (including all manifold vertices and copies of non-manifold vertices) is $m_0 + k_0$. Assuming that the hash tables are 10% full in order to support constant access time, the size of the hash table H_v is $20n_0$ pointers. The storage cost of the DLD data structure for various domains is shown below:

- For general non-manifold complexes:

  - lower level: $m_0 + k_0 + 2w_1 + 4d_2 + 8m_3 + 20n_0 + 30n_1$ pointers and $m_0 + k_0 + 3d_2 + 6m_3$ bits,
  - upper level: $2C + 6n_1 + 2n_0 + k_1 + k_0$ pointers,
  - hash table: $20n_0$ pointers.

- For manifold complexes, $d_2 = w_1 = n_0 = n_1 = k_0 = k_1 = 0$ and $C = 1$
  - lower level: $m_0 + 8m_3$ pointers and $m_0 + 6m_3$ bits,
  - upper level: 2 pointers,
  - hash table: 0 pointers.

A comparison with the extended Indexed data structure with Adjacencies (IA)[64] for manifolds gives us a measure of the scalability of the DLD data structure. When encoding manifolds, the DLD data structure is reduced to the EIA with just some additional bit flags. Thus, the overhead of the DLD data structure in encoding manifold is $m_0 + 6m_3$ bits and 2 pointers.

6.2.3 Navigation in the DLD Data Structure

In this Section, we discuss how to retrieve topological relations from the DLD data structure. These algorithms are the basic building blocks for any algorithm which navigates or updates the complex.

Boundary relation can be retrieved both for top simplexes and for faces of top simplexes. For any top p-simplex, σ $(p = 2, 3)$, the set of q-faces of σ are described as $\binom{p+1}{q+1}$ combinations of $(q + 1)$ vertices of σ. Thus, to retrieve boundary relation

$R_{p,q}(\sigma)$, relation $R_{p,0}(\sigma)$ is retrieved, and the combinations describing the q-faces are generated.

We retrieve co-boundary relation $R_{p,q}(\sigma)$ of a p-simplex σ through a traversal of the star of σ, $(p = 0, 1, 2)$. In the case in which σ is a manifold simplex, all q-simplexes incident at σ belong to the same IQM component. Thus, the traversal of the star of σ is performed within the lower level data structure. When σ is non-manifold, the star of σ is distributed among several components. Therefore, it is necessary to access the upper level data structure to retrieve all the components incident at σ.

Relation $R_{0,h}(v)$, $h = 2, 3$, in an h-dimensional IQM component C is retrieved by traversing the star of v in C through relations $R^*_{0,h}$, $R_{h,h}$ and $R_{h,0}$. The traversal is performed by starting with h-simplex $\sigma = R^*_{0,h}(v)$. The h-simplexes $(h-1)$-adjacent to σ are found through $R_{h,h}(\sigma)$, and those which are incident at v are found by considering $R_{h,0}$ for such h-simplexes. The pseudocode is described in Algorithm 6.

This process is linear in the number of h-simplexes incident at v. If we want to retrieve $R_{0,q}(v)$ in an h-dimensional IQM component with $q < h$, we perform the same traversal described above, but we collect as result the q-faces of the h-simplexes found in the retrieval. The time complexity is still linear in the number of q-simplexes incident at v, since the number of h-simplexes in the star of v is linear in the number of q-simplexes incident at v because of Euler' formula.

Next, we consider how to retrieve relation $R_{0,q}(v)$, $0 < q \leq h$, for a non-manifold vertex u. The star of u is the union of the stars of all its vertex copies. The vertex copies of u are retrieved from the upper level data structure. Relation

Algorithm 6 Co-boundary$_{0,h}(C,\ v)$

Require: v is a vertex in the h-dimensional component C

```
1: S ← ∅
2: σ ← R*_{0,h}(v)
3: Mark σ as visited
4: S ← S ∪ {σ}
5: Enqueue(Q, σ)
6: while not Empty(Q) do
7:    σ ← Dequeue(Q)
8:    for each γ ∈ R_{h,h}(σ) do
9:       if v ∈ R_{h,0}(γ) and γ is not visited then
10:         Mark γ as visited
11:         S ← S ∪ {γ}
12:         Enqueue(Q, γ)
13:      end if
14:   end for
15: end while
```

$R_{0,q}$ for each vertex copy is retrieved from the lower level data structure as though the vertex copy u was a manifold vertex. Given an arbitrary vertex v in a given component C of dimension h, the bit-flag indicates whether v is a vertex copy. If it is, the reference to the hyperarc representing the non-manifold vertex is retrieved from the hash table H_v. From the hyperarc, we retrieve all the other copies of the same vertex, and then the q-simplexes incident at each such copy. Thus, all $R_{0,q}(v)$ relations, where $0 < q \leq h$, can be retrieved in time linear in the number of q-simplexes in the star of v.

We consider how to retrieve co-boundary relation of type $R_{1,q}(e)$, $0 \leq q \leq h$, for an edge e. If e is a manifold edge, we consider a tetrahedron or triangle σ containing it. (Note that since the edges are not encoded in the DLD data structure for the IQM component, we consider all edges to be specified through a top simplex containing it.) For a 3-component, we retrieve all tetrahedra, or triangles, depending

on whether $q = 2$ or 3, incident at e, by traversing the star of e starting from σ, and retrieving all the other tetrahedra or triangles, by using $R_{h,h}$ and $R_{h,0}$ relations. For a 2-component, $R_{1,2}(e)$ is retrieved by simply considering $R_{2,2}$ of triangle σ.

If e is a non-manifold edge, we get access from the hyperarc describing it to a top simplex in each component containing it. For each component, we repeat the process discussed above for the manifold edge. The time complexity of this algorithm is linear in the number of q-simplexes incident at e.

Co-boundary relation $R_{2,3}(f)$ for a triangle f is retrieved through the $R_{3,3}$ relation of a tetrahedron that shares f. Adjacency relations $R_{p,p}$ $(p = 0, 1, 2)$ are retrieved as a combination of boundary and co-boundary relations, and are not elaborated here. Their time complexity is linear in the number of p-simplexes produced as result of retrieval. Thus, all topological relations can be extracted in optimal time from the DLD data structure.

6.3 Evaluation, Comparison and Discussion

In this Section we compare the NMIA and the DLD data structures for the representation of 3D simplicial shapes. This comparison is made in terms of: the types of entities and topological relations encoded, the storage cost and the navigation efficiency of these data structures.

6.3.1 Comparison between NMIA and DLD

In terms of the entities and relations encoded, the NMIA and the DLD data structures are comparable. Both encode only vertices and top h-simplexes for any $h \leq 3$. The relations encoded by both data structures are of the following types: adjacency relations $R_{h,h}$ among top h-simplexes, incidence relations $R_{h,0}$ of top h-simplexes, partial co-boundary $R^*_{0,h}$ relations which capture selected top h-simplexes in the star of a vertex, and partial co-boundary relations $R^*_{0,h}$ which capture selected top h-simplexes incident at a non-manifold edge.

In terms of scalability, both the NMIA and the DLD data structures are comparable to the EIA when the domain is manifold.

Also, both data structures support an efficient retrieval of topological relations. The retrieval algorithms for relations $R_{p,3}$, for $p = 0, 1$, are sub-optimal for the NMIA data structure, but still linear in the number of top simplexes in the star of a vertex for $p = 0$, or edge for $p = 1$. A summary of the navigation efficiency is shown in Table 6.3.

Relations retrieved	Boundary $R_{p,q}(\sigma)$	Co-boundary $R_{p,q}(\sigma)$	Adjacency $R_{p,p}(\sigma)$
NMIA	Optimal	$R_{1,3}$ and $R_{1,2}$: linear in top simplexes at σ Others: Optimal	$R_{1,1}$: linear in top simplexes at 2 vertices of σ Others: Optimal
DLD	Optimal	Optimal	Optimal

Table 6.3: Navigation efficiency of the NMIA and the DLD data structures

The primary difference between the two data structures is that the DLD data structure encodes the complex as an IQM decomposition, thus allowing the non-manifold singularities to be explicitly addressable, while the NMIA encodes the complex as a single piece with non-manifold singularities distributed inside the complex. The DLD belongs to the category of decomposition-based data structures.

6.3.2 Comparison of the NMIA with 3D Instances of IG and IS for Non-manifold Simplicial 3-complexes

While the DLD and the NMIA data structures are comparable in their storage costs, navigation efficiency and scalability to manifold domain, they are designed from two different approaches. Therefore, as we compare them with existing data structures, we need to compare them based on the approaches taken. Instantiated from the IQM data structure (see Section 4.2.2) for abstract complexes to Euclidean 3-complexes, the DLD data structure is one of a kind in the decomposition approach.

Outside of the decomposition approach, the only data structures that can describe non-manifold simplicial 3-complexes are the NMIA, the Incidence Graph (IG), the Simplified Incidence Graph (SIG) (see Section 5.2) and the IS data structure (see Section 5.3). In this Section, we compare the NMIA data structure with the IG and the IS data structure which is an improvement over the SIG.

Consider a simplicial 3-complex. The NMIA data structure encodes only the top simplexes and the vertices of the complex. This is in contrast with the dimension independent data structures, which encode all the simplexes in the complex. The

relations encoded by the NMIA data structure and the dimension independent ones are compared in Table 6.4

Relations encoded	Boundary	Co-boundary	Adjacency
NMIA	$R_{h,0}(\sigma)$: σ: top simplex	$R^*_{0,h}(v)$ $(h = 1, 2, 3)$, $R^*_{1,3}(e)$ and $R^*_{1,2}(e)$	$R_{h,h}(\sigma)$ σ: top simplex
IG	$R_{p,p-1}$: $0 < p \leq d$	$R_{p,p+1}$: $0 \leq p < d$	-
IS	$R_{p,p-1}$: $0 < p \leq d$	$R^*_{p,p+1}$: $0 \leq p < d$	-

Table 6.4: A summary of the topological encoded by the NMIA, the IG and the IS data structures

Table 6.5(a) shows five non-manifold data sets with mixed tetrahedra, dangling-faces and wire-edges. The storage cost of the NMIA data structure and of 3D instances of the IG and of the IS data structure are compared for these 3-complexes. The NMIA is the most compact as it encodes only top simplexes and vertices explicitly. The IG is at least three times as much as the NMIA because it encodes all simplexes and a large number of incidence relations. IS is more compact than the IG because it only encodes a subset of the incidence relations encoded by the IG.

In terms of navigation efficiency, the IG support the retrieval of all topological relations at optimal time for 3-complexes (See Section 5.4.2). The IS and the NMIA are comparable in their support of the retrieval of topological relations. The IS is able to support optimal retrieval of all relations given an optimized implementation.

We also evaluate the storage costs of the various existing data structures for manifold simplicial 3-complexes, namely, the CHF (see Section 4.1.3.3), the

Data set	n_0	n_1	n_2	n_3	n_1^t	n_2^t	C_e	$C_v - n_0$
Bucket	53	167	160	48	6	32	32	14
Wheel	402	2093	2728	1148	96	32	56	256
Balloon	1108	3913	3616	856	64	1632	0	160
Flasks	1301	6307	8465	3455	0	460	104	0
Teapot	4658	17.9k	17.0k	5666	2944	3930	144	5959

(a)

Data set	NMIA	IG	IS
Bucket	591	2012	1105
Wheel	10.2k	33.9k	17.7k
Balloon	13.1k	44.2k	23.4k
Flasks	30.4k	104k	53.2k
Teapot	73.8k	219k	120k

(b)

Table 6.5: (a) Five non-manifold 3D simplicial data sets; (b) Storage cost of three non-manifold data structures for the data sets in (a)

FE (see Section 4.1.3.1), and the specialization of the dimension-independent data structures, namely, the EIA (see Section 4.1.1.3) and the IG, along with our proposed dimension-independent IS data structure. In the case of manifold simplicial 3-complexes, the numbers of elements encoded in the data structures are evaluated to be:

- NMIA: $8n_3 + n_0$
- EIA: $8n_3 + n_0$
- IG : $8n_3 + 6n_2 + 4n_1$
- IS : $4n_3 + 5n_2 + 3n_1 + n_0$
- CHF: $8n_3 + n_2 + n_1 + n_0$

- FE: $4n_3 + 12n_2 + 2n_1$

In Table 6.6, we report an experimental comparison of the storage costs of these data structures on the five data sets of manifold tetrahedral meshes shown in Table 4.11. It can be observed that the NMIA and the EIA data structures are the most compact representation, followed by the CHF. The IS data structure is in the middle range in terms of storage cost. The data structure that encodes the most amount of topological information is the FE.

Data set	n_0	n_1	n_2	n_3
Rings	2.52k	13.2k	18.8k	8.13k
Basket	1.21k	6.43k	9.22k	4.00k
Cylinder	1.31k	7.79k	11.6k	5.16k
Gargoyle	2.73k	14.7k	22.0k	10.0k
Torus	2.29k	15.4k	24.0k	10.9k

(a)

Data set	EIA/NMIA	IG	IS	CHF	FE
Rings	67.6k	231k	169k	99.6k	285k
Basket	33.2k	113k	82.6k	48.9k	139k
Cylinder	42.6k	142k	104k	62.1k	176k
Gargoyle	82.7k	271k	197k	119k	333k
Torus	89.2k	293k	212k	129k	362k

(b)

Table 6.6: (a) Five manifold 3D simplicial data sets; (b) Storage cost of six data structures for the data sets in (a)

6.4 Summary

In this Chapter, we addressed the problem of the representation of 3D non-manifold simplicial shapes. We specifically focused on cost-efficient data structures

for huge data sets. We have considered two different approaches. In the first approach, we consider models that are pre-domininantly 3D and manifold, with small portion of lower dimensional parts and non-manifold connectivities, which are treated as singularities. The Non-manifold Indexed Data Structure with Adjacencies (NMIA) is an optimized data structure that follows this first approach. After the publication of our work on the NMIA [13, 14], it has been observed in the Solid Modeling community that there is a keen interest on compact representations of non-manifold 3D simplicial shapes. The development on such representations has been followed up by a research group in computer graphics.

In the second approach, we consider a model as a collection of parts that are uniformly dimensional and nearly manifold, connected at non-manifold joints. The Double-Level Decomposition Data Structure (DLD) is a new paradigm of this approach. The two representations proposed for these two approaches are comparable in terms of their storage cost and navigation efficiency. They are also highly compact because, in both data structures, the primary entities encoded are the vertices and the top simplexes, and the topological relations encoded are those among these encoded simplexes. This work has been published in [47].

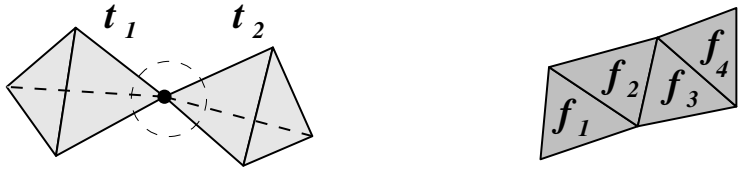
The concept of representing a shape at two levels may also be used in conjunction with other data structures such as the IS at the lower level. This direction can be taken on further to build a hierarchical representation at two levels of abstraction. This is addressed in the next chapter.

Chapter 7

Understanding Non-manifold Shapes by Decomposition

In this Chapter, we discuss the development from shape representation of shape understanding. The generic approach to non-manifold shape representation is based on the assumption that a 3D shape has a predominant dimension and is predominantly manifold. In fact, most data structures designed for non-manifold 2-complexes work under this assumption. Departing from this assumption, the decomposition approach considers a 3D shape as a collection of *topologically simpler parts* connected together at *non-manifold joints*. This approach opens up a some new issues to be addressed. There is the pressing issue on what topological properties have practical significance in semantics. The semantics of a shape associates domain-specific knowledge to the parts of the shape, and is thus context-dependent. Topology and semantics are not issues at the same level. To achieve both topological significance and semantical significance, we proposed a two-level decomposition. The lower level decomposition is purely topological and is based on the property of manifold-connectedness. The upper level decomposition integrates topological fea-

tures with semantics. The components of the semantics-oriented decomposition are uniformly dimensional nearly-manifold parts. We also propose a decomposition for describing the connectivities among components in these two decompositions. Then we discuss two collaborations we have on the application of non-manifold shape decomposition to form feature identification and shape understanding. Lastly we describe a shape analysis tool that has been built.



(a) (b)

Figure 7.1: Examples on the degree of connectivity: (1) Two tetrahedra that are 0-connected; (2) Four triangles that are 1-connected.

The topological complexity of a non-manifold shape can be characterized by *dimensionality* and *connectivity*. A shape of uniform dimension (i.e., a regular shape) is "dimensionally simpler" than a shape with mixed dimensions. We know that in any regular h-complex, top simplexes are necessarily manifold. In arbitrary non-manifold h-dimensional shapes, non-manifold situations only occur at simplexes of dimensions strictly lower than h. Thus, the decomposition of a shape based on dimensions yields a collection of regular parts that are simpler to understand. The connectivity within a shape is measurable both quantitatively and qualitatively. A quantitative measurement of a non-manifold shape is its *degree of connectivity*. It gives a direct measurement to the overall "thickness" of the object. If an object is

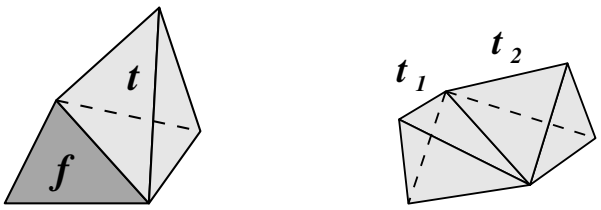
h-connected, it means that the path connecting any two simplexes in the object is at least h-dimensional. Figures 7.1(a) and (b) give two examples of the degree of connectivity. The degree of connectivity of a mixed-dimensional shape is constrained by the lowest dimensionality of the top simplexes in the shape.

Property 1 *A manifold d-complex Σ is $(d-1)$-connected. Σ is also a d-pseudo-manifold embedded in E^h.*

Property 2 *A regular h-complex Σ is at most $(h-1)$-connected.*

Property 3 *If the lowest-dimensional top simplex in a complex Σ is h where $h \leq d$, the complex is at most $(h-1)$-connected.*

These properties are illustrated in Figures 7.2(a)-(b). In Figure 7.2(a), the lowest dimensional top simplex is a dangling face. Therefore, the 3-complex is at most 1-connected. Figure 7.2(b) shows a regular complex that is 1-connected.



(a) (b)

Figure 7.2: Basic non-manifold properties: (a) A 3-complex Σ that consists of tetrahedron t sharing an edge with dangling-face f, which is the lowest-dimensional top simplex in Σ. Since f is of dimension 2, Σ is at most 1-connected; (b) A regular 3-complex that consists of two tetrahedra t_1 and t_2 and is only 1-connected.

Qualitatively, the "smoothness" of the shape is measurable by how close the shape is to manifold. As pointed out in [26], manifoldness is not a sufficient basis for decomposition of any non-manifold complex of dimension above 2. In the following, we define the notion of manifold-connectedness as a qualitative topological description of smoothness for simplicial objects up to dimension 3.

7.1 Manifold Connectedness

In this Section, we introduce the notion of manifold-connectedness, and then examine some properties of manifold-connected complexes embedded in the Euclidean 3D space E^3. Manifold-connectedness is close to the notion of manifoldness.

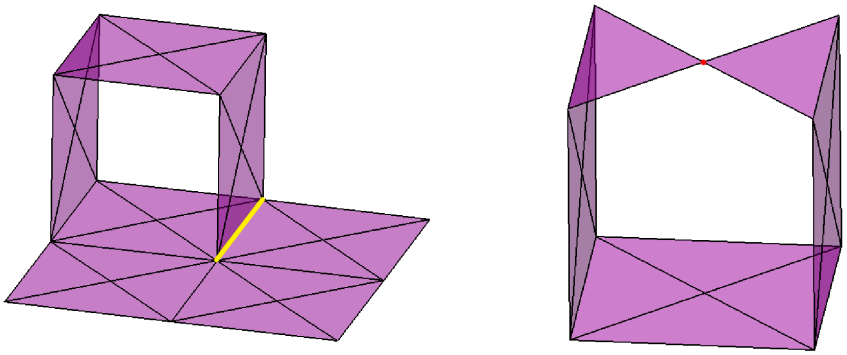
Recall that a regular simplicial d-complex is one in which every top simplex is d-dimensional. We consider a regular simplicial d-complex Σ embedded in the 3D Euclidean space, where $d = 1, 2, 3$. In such a complex, a $(d-1)$-simplex σ is a *manifold* simplex if and only if there are at most two d-simplexes in Σ incident in σ. We thus introduce the following definitions.

Definition 1 *An $(h-1)$-path (i.e., a path formed by alternating h- and $(h-1)$-simplexes) such that every $(h-1)$-simplex in the path is a manifold simplex is called a* manifold (h-1)-path.

Definition 2 *Two d-simplexes in a d-complex Σ are said to be* manifold-connected *if and only if there exists a manifold $(d-1)$-path connecting them.*

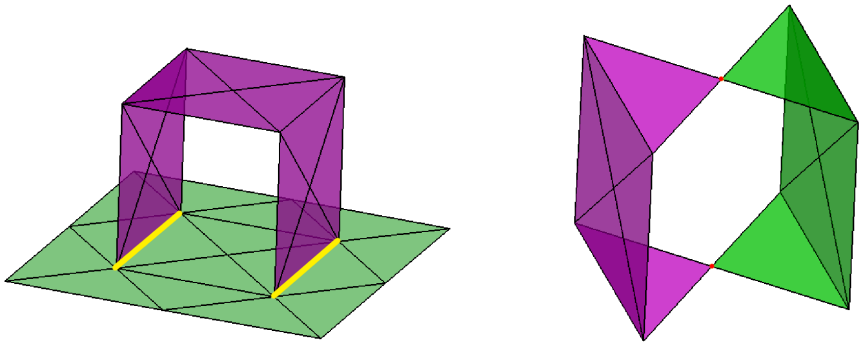
Definition 3 *A regular simplicial d-complex Σ is a* manifold-connected *complex if and only if every pair of d-simplexes in Σ is manifold-connected.*

Figures 7.3(a) and (b) give two examples of manifold-connected 2-complexes. Figures 7.5(a) and (b) give two examples of manifold-connected 3-complexes. Both examples in Figures 7.4(a) and (b) are not manifold-connected. In the complex in (a), the upper part is connected to the lower part only through non-manifold edges. In (b), the left and right parts are connected only through non-manifold vertices.



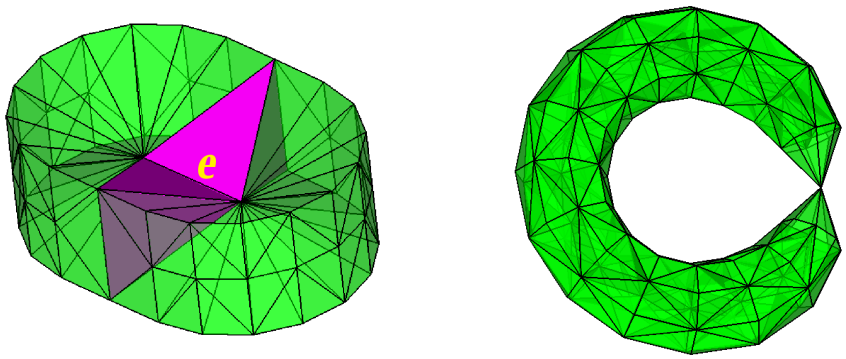
(a) (b)

Figure 7.3: Examples of simplicial manifold-connected 2-complexes with (a) a non-manifold edge and (b) a non-manifold vertex.



(a) (b)

Figure 7.4: Examples of simplicial 2-complexes that are not manifold connected.



(a) (b)

Figure 7.5: Examples of simplicial manifold-connected 3-complexes with (a) a non-manifold edge and (b) a non-manifold vertex.

A 1-dimensional manifold-connected shape consists of a linear or circular chain of wire-edges. 2-dimensional manifold-connected shapes are composed of dangling triangles (i.e., top 2-simplexes). There are two kinds of non-manifold situations in a manifold-connected 2-complex, namely, that at an edge and that at a vertex. The non-manifold situation at an edge occurs in the form of having more than two dangling triangles sharing that edge. An example of a non-manifold edge in a manifold-connected 2-complex is shown in Figure 7.3(a). Here also, the non-manifold situation at a vertex v consists a combination of the following two cases: the link of v (which is a 1-complex without isolated vertices) is disjoint, or a non-manifold vertex exists in the link of v.

3-dimensional manifold-connected shapes consist of tetrahedra. Since the 3-complex is embedded in E^3, all the triangles are shared by at most two tetrahedra. The non-manifold situations occur at the vertices and at the edges. The non-manifold edge is characterized by having more than one fan of tetrahedra in

its star. The link of a non-manifold edge is a set of disconnected chains of edges. An example of a non-manifold edge created by pinching a duster at its center is shown in Figures 7.5(a). The tetrahedra incident at non-manifold edge, e, are highlighted. The star of a non-manifold vertex consists of tetrahedra only. The link of a non-manifold vertex is a regular 2-complex consisting of triangles. An example of a non-manifold vertex on a pinched torus is shown in Figure 7.5(b).

In determining what is a meaningful decomposition, the issues to be considered include theoretical significance and practical cost efficiency. A decomposition that yields parts of homogenous dimension and with nearly manifold properties is an optimal choice. The class of manifold-connected h-complexes is of special interest to us because:

- The star of each non-manifold simplex in Σ is formed by a finite number of manifold-connected components. When we consider manifold-connected complexes embedded in E^3, the only difference between a manifold edge and a non-manifold edge lies in the number of components in their star, while the only difference between a manifold vertex and a non-manifold vertex lies in both the number of components and the presence of non-manifold edges in their stars.

- Manifold-connectedness ensures that the whole complex is traversable through visiting all top h-simplexes and $(h-1)$-simplexes once. Since the number of $(h-1)$-simplexes is linearly proportional to the number of h-simplexes, the traversal is a linear procedure with respect to the number of top h-simplexes

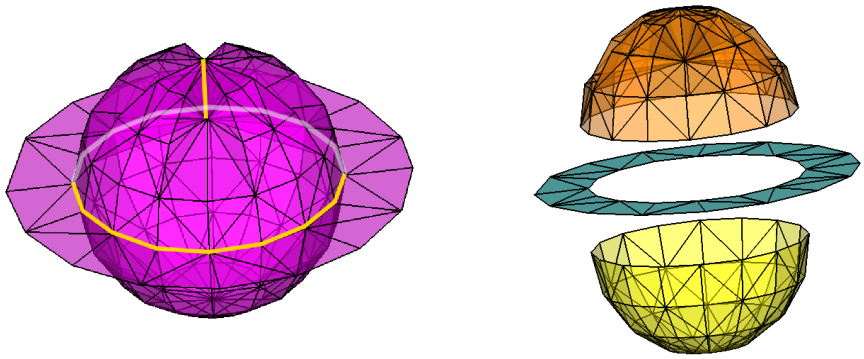
in the complex.

As a result, manifold-connectedness enables the design of compact, efficient and conceptually simple representations.

7.2 The MC-Decomposition[1]

A simplicial 3-complex Σ embedded in the three-dimensional Euclidean space can be decomposed into manifold-connected one-, two- and three dimensional manifold-connected complexes, called *MC-components*. A decomposition Δ of Σ is a collection of sub-complexes of Σ, such that the union of the components in Δ is Σ, and any two components Σ_1 and Σ_2 in Δ, if they intersect, intersect at a collection of non-manifold vertices and edges. An *MC-decomposition* is constructively defined by cutting complex Σ only and at all its non-manifold vertices and edges, and forming components that satisfy the following property: two k-dimensional top simplexes σ_1 and σ_2 belong to the same component in the MC-decomposition if and only if there exists a manifold $(k-1)$-path that connects σ_1 and σ_2 in Σ (see [44] for more details).

[1]Originally published in [45]. Reproduced with notice of ACM copyright: permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.



(a) (b)

Figure 7.6: An example of the MC-decomposition. (a) A hollow ball that is pinched at the top and has a circular wing, the visible non-manifold edges are highlighted; (b) Its MC-decomposition into three manifold-connected components.

7.2.1 An Algorithm for Computing an MC-decomposition

Our algorithm for computing the MC-decomposition of a simplicial 3-complex Σ consists of extracting first the k-dimensional regular sub-complexes of Σ for $k = 1, 2, 3$, and then computing the MC-decomposition of each such k-dimensional regular sub-complex. To compute the MC-decomposition of a k-dimensional regular complex, we use the property that any pair of manifold simplexes belonging to the same k-dimensional manifold-connected component (for $k = 1, 2, 3$) must be connected through a manifold $(k-1)$-path. This means that every such component can be traversed by following the manifold $(k-1)$-paths connecting the k-simplexes in the component. The algorithm for computing the MC-decomposition of the simplicial 3-complex Σ thus consists of the following steps:

1. Identify all non-manifold edges and vertices in Σ to ensure that these simplexes will not be visited in any traversal;

2. For $k=1,2,3$, extract the top k-simplexes from Σ, which form the k-dimensional regular sub-complexes of Σ;

3. For each k-dimensional regular sub-complex Σ_k, perform the following traversal that starts with some unvisited top k-simplex $\sigma \in \Sigma_k$: find all other unvisited top k-simplexes that are reachable from σ by alternately visiting manifold $(k-1)$-simplexes and their incident top k-simplexes. All visited top k-simplexes belong to the same k-dimensional manifold-connected component. The traversal of Σ_k is complete when all top k-simplexes have been visited.

4. Mark each non-manifold singularity, that is shared by more than one manifold-connected component, as a joint.

In our implementation of the algorithm above, we have used the *Incidence Simplicial (IS) data structure* to encode complex Σ (see Section 5.3). The IS data structure is a good choice for the MC-decomposition because of its ease of use as well as compactness. Algorithm 7 describes how MC-decomposition is computed from the IS representation of a simplicial complex Σ.

The manifold test of simplex γ in line 12 is based on the properties of non-manifold singularities in 3D complexes as follows. If γ is a triangle, it is always manifold. If γ is an edge, it is manifold if its partial co-boundary relation $R^*_{1,2}$ consists of either just one triangle, or two triangles that are top simplexes. If γ is a vertex, it is detected either as extreme vertices of non-manifold edges, or as vertices whose partial co-boundary relations $R^*_{0,1}$ consist of more than one edge. By using the IS data structure, the computation of the MC-decomposition requires a

Algorithm 7 MC-decomposition(Σ)

```
1: comp ← 0
2: Initialize empty queue Q
3: for p = dim(Σ) down to 1 do
4:    for each simplex σ of dimension p do
5:       if σ is top simplex and σ is not labeled then
6:          comp ← comp + 1
7:          label(σ) ← comp
8:          Enqueue(Q, σ)
9:          while not Empty(Q) do
10:            τ = Dequeue(Q)
11:            for each (p − 1)-simplex γ in R_{p,p−1}(τ) do
12:               if γ is manifold then
13:                  for each p-simplex μ in R*_{p−1,p}(γ) do
14:                     if μ is top simplex and μ is not labeled then
15:                        label(μ) ← comp
16:                        Enqueue(Q, μ)
17:                     end if
18:                  end for
19:               end if
20:            end for
21:         end while
22:      end if
23:   end for
24: end for
```

time complexity that is linear in terms of the number of simplexes in the simplicial complex Σ.

By examining the top h-simplexes incident at the $(h-1)$-faces of each top h-simplex σ, all top h-simplexes that are manifold-connected are visited. Two top h-simplexes of Σ carry the same label if and only if they are manifold-connected. Therefore, the MC-decomposition computing by the traversal algorithm is unique. The time complexity of this algorithm is linear with respect to the number of top simplexes in the complex.

Figure 7.7(a) shows the MC-decomposition of the object in Figure 7.6(a). Observe that the non-manifold edge at the top of the shape is preserved in the MC-decomposition. The example shown in Figure 7.7(b) is under-decomposed. The example in Figure 7.7(c) is over-decomposed. Figure 7.7(d) shows the overly decomposed part in the decomposition of Figure 7.7(c).

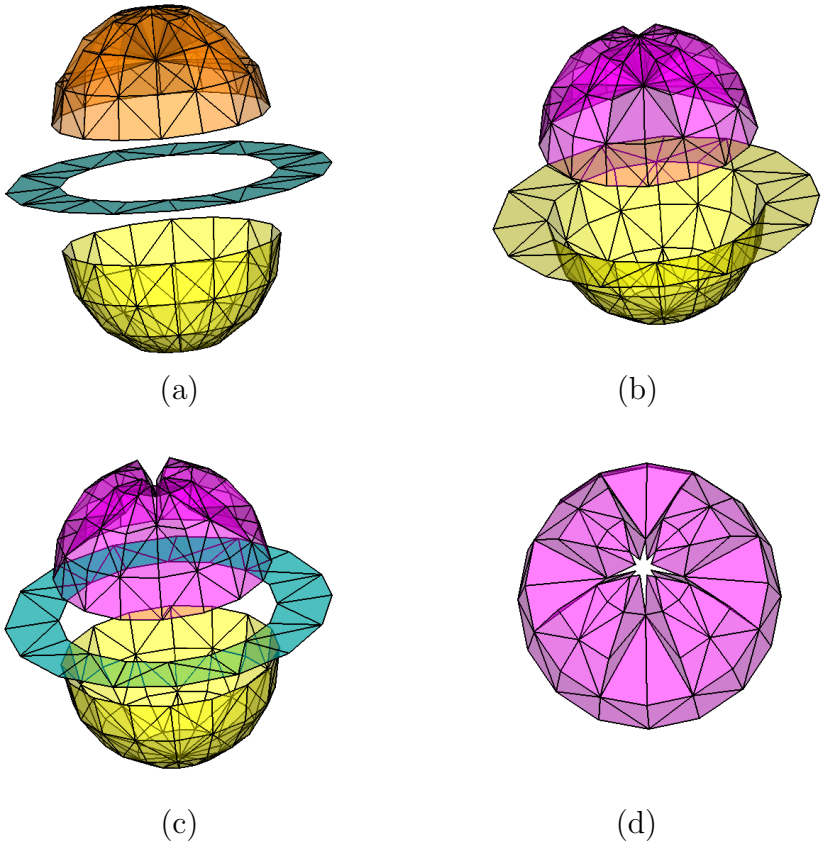


Figure 7.7: (a) MC-decomposition of 7.6(a); Both (b) and (c) are not MC-decompositions of 7.6(a); (b) is under-decomposed while (c) is over-decomposed; (d) a top-down view of the uppermost component of (c), showing that the connectivity of the triangles is broken at the non-manifold edge at the top.

7.3 The Semantics-Oriented Decomposition[2]

In this Section, we discuss a decomposition of a non-manifold 3D shape embedded in the Euclidean 3D space into composite topological shapes which are of interest because of their richer semantics in several application scenarios. We limit first the investigation of this decomposition to 2-complexes in which all triangles are 1-connected, and to 3-complexes in which tetrahedral parts have one connected boundary. (Thus, we exclude solids with holes in their interior.) In Section 7.3.1, we introduce these composite shapes in a 2-complex and discuss how to compute the semantics-oriented decomposition based on these shapes. Then, in Section 7.3.3, we show that 3-complexes embedded in the 3D Euclidean space are no harder to decompose than the 2-complexes.

7.3.1 Semantics-Oriented Decomposition of a Simplicial 2-Complex

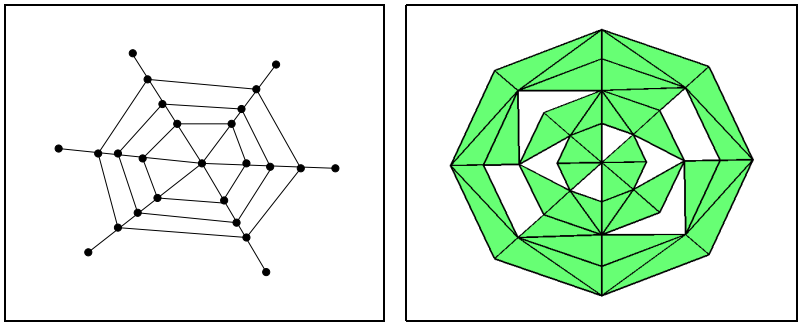
We consider here 1-connected simplicial 2-complexes. A semantics-oriented decomposition is a decomposition of a simplicial 2-complex embedded in the three-dimensional Euclidean space into components informally defined as:

- *Wire-webs*, which are maximal connected components formed only by top 1-simplexes. An example is given in Figure 7.8(a).

[2]Originally published in [45]. Reproduced with notice of ACM copyright: permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

- *shells*: a shell C is any manifold-connected 2-complex without boundary, such that the three-dimensional region in the space enclosed by C is connected, i.e., any two points in the region can be joined by a curve which does not intersect any simplex of Σ. In Figure 7.6(a), the part that encloses a hollow volume is an example of a shell.

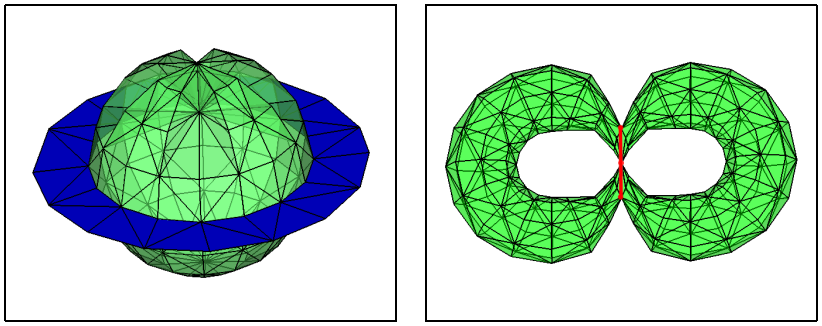
- *Sheets*, which are maximal manifold-connected 2-complexes with boundary that do not enclose any 3D region. Figure 7.8(b) shows an example of a sheet.



(a) (b)

Figure 7.8: Examples of (a) a wire-web and (b) a sheet

Figure 7.9(a) shows the semantics-oriented decomposition of the example shown in Figure 7.6(b). The hollow pinched ball in Figure 7.6(b) is a shell and the circular wing is a sheet. In the example of Figure 7.9(b), we have two shells formed by two hollow tori.



(a) (b)

Figure 7.9: Examples of the semantics-oriented decomposition of: (a) a hollow pinched ball with a circular wing; (b) two tori that are connected at two edges.

We observe that a shell in a simplicial 2-complex is the union of 2-dimensional manifold-connected components. In the example of Figure 7.6(b), the shell is formed by the upper component and the lower component of its MC-decomposition, as

shown in Figure 7.6(c).

7.3.2 Computing the Semantics-Oriented Decomposition for a 2-complex

In this Section, we show how to compute the semantics-oriented decomposition of a simplicial 2-complex Σ from its MC-decomposition (discussed in Section 7.2). We highlight the computation of shells. It is known that there is a unique way to assign orientations consistently to all the shells in a simplicial 2-complex Σ so that, if a triangle appears in two shells, its orientations in the two shells are opposite [28]. Thus, given the MC-decomposition of a simplicial 2-complex, we compute the shells from the manifold-connected 2-dimensional components by duplicating each component and assigning opposite orientations to them. Then, we sew together the parts whose oriented boundaries match each other. Thus, the algorithm for finding the wire-webs, sheets, and shells consists of the following steps:

1. Compute a wire-web as the union of all manifold-connected one-dimensional components which share one or more vertices.

2. Classify any two-dimensional manifold-connected component in the MC-decomposition that contains boundary edges (a boundary edge is one that belongs to exactly one triangle) in Σ as a sheet (an example is the circular wing in the Figure 7.6(b)).

3. For the remaining 2-dimensional components in the MC-decomposition, each component is duplicated and opposite orientations are assigned to each component and its duplicate. Based on the orientation of the boundaries, pieces

belonging to the same shell are matched.

We elaborate step 3 here. After the identification of the sheets, the remaining 2-dimensional components are either without boundaries or bounded by edges that are non-manifold. In the former case, each such component completely encloses a volume. In the latter case, the component may be part of an orientable surface that encloses a volume. Therefore, orientation is used to stitch the components together. To this end, each component is duplicated so that the original and its duplicate are assigned opposite orientations, since each component bounds two volumes, one on each side. By the left-hand rule, the orientation of each copy of the component induces an order to edges on its boundary (see Figure 7.10(a) and (b)).

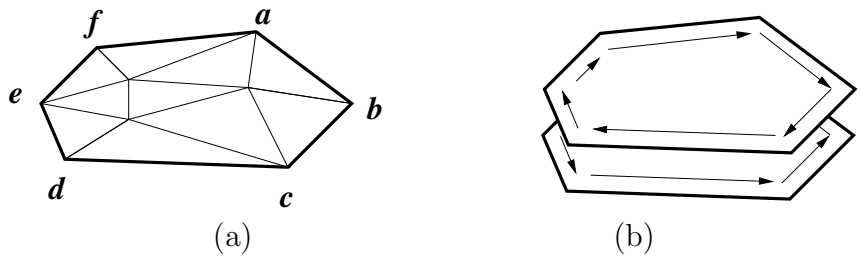


Figure 7.10: (a) A 2D component obtained from the MC-decomposition; (b) Two orderings of the boundary of the component shown in (a), corresponding to opposite orientations

Moreover, at any non-manifold edge e that is shared by orientable surfaces bounding 3D volumes, the surfaces with their assigned orientations may be ordered radially at e (see Figure 7.11(a) and (b)). Two neighbouring surfaces that are adjacent in this ordering can be stitched together at e. For example the neighbouring surfaces $\vec{A}$ and $\vec{B}'$ are adjacent in the order around e and may thus be stitched

together. After stitching, the resultant surface has the same orientation but it has a new boundary (see Figure 7.12(a) and (b)). Stitching terminates when no more surfaces share overlapping boundaries, at which point, those surfaces that have no boundary completely enclose volumes. The outermost surfaces which form the exterior of the complex are identifiable geometrically and are discarded. The remainder is the set of shells in the complex.

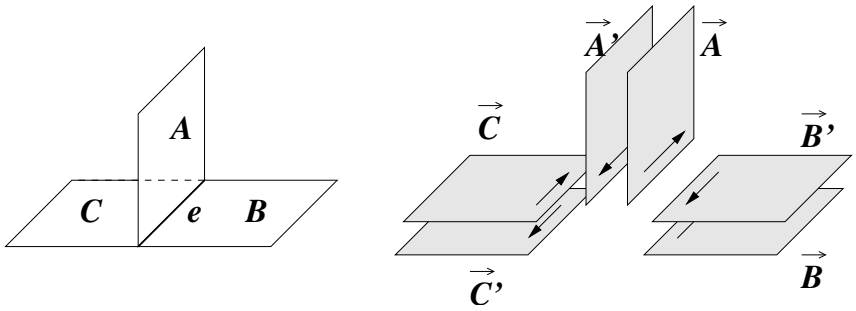


Figure 7.11: (a) Three 2D components, A, B and C, sharing a non-manifold boundary e; (b) The six surfaces with orientation, $\vec{A}, \vec{A'}, \vec{B}, \vec{B'}, \vec{C}$ and $\vec{C'}$, corresponding to A, B and C in (c) are sortable around the non-manifold boundary e

Consider the example of two overlapping spheres sharing a common surface as shown in Figure 7.13(a). The MC-decomposition of this shape yields three pieces as shown in Figure 7.13(b). The components are duplicated and assigned unique orientations. Each such surface with an orientation is shown in a different colour in Figure 7.13(c). Stitching is performed on the surfaces and yield three sets of surfaces without boundaries, of which, the outermost one is discarded. Note that the time complexity of step 3 of this algorithm is linearly proportional to the total number

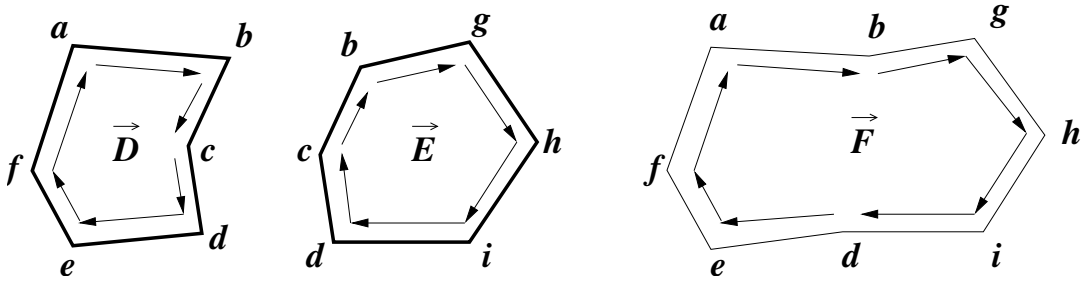


Figure 7.12: (a) Two surfaces with orientation, D and E, whose boundaries may be stitched together along their common edges; (b) The resultant surface F of the stitch has the same orientation as D and E but it has a new boundary

of triangles at the non-manifold edges of the component. This is because step 3 is performed by tracing the boundaries of the components.

7.3.3 Semantics-Oriented Decomposition of a Simplicial 3-Complex

The semantics-oriented decomposition can be also extracted for any simplicial 3-complex embedded in the 3D Euclidean space. In this Section, we define the components in this decomposition for 3-complexes in which each tetrahedral component has one connected boundary. The components of the semantics-oriented decomposition for the domain of a simplicial 3-complex are:

- *Wire-webs*, which are maximal connected components formed by top 1-simplexes.
- *shells*: a shell C is any manifold-connected triangles without boundary, such that the three-dimensional region in the space enclosed by C is connected, i.e., any two points in the region can be joined by a curve which does not intersect

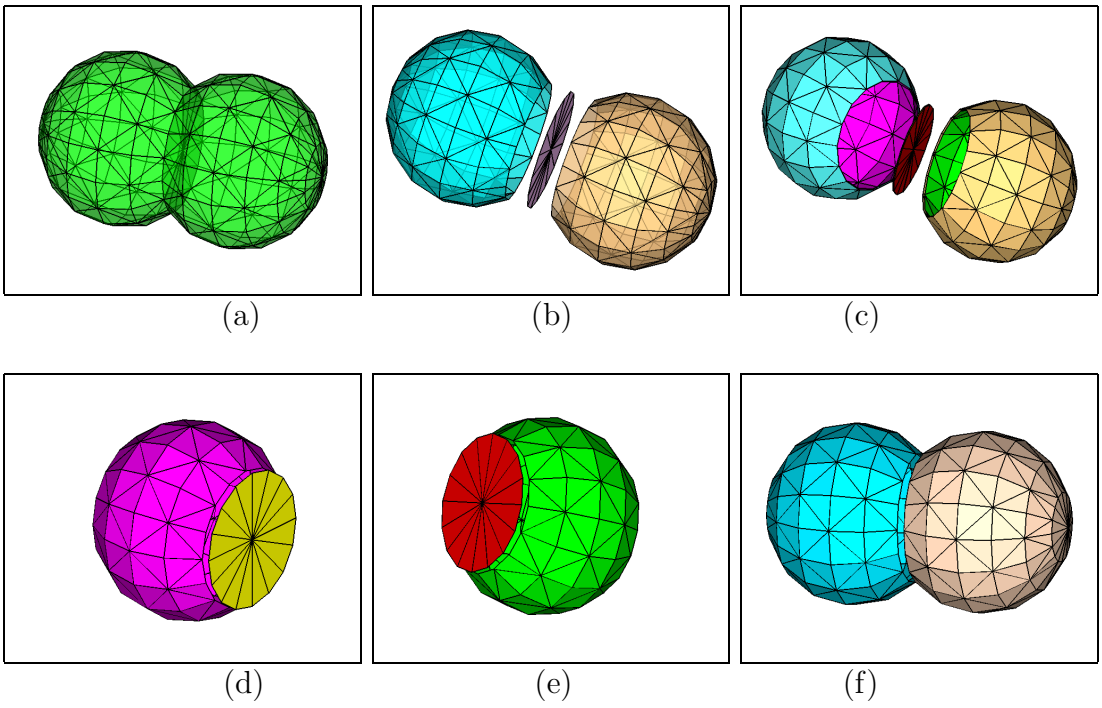


Figure 7.13: (a) A 2-complex describing two spheres that overlap and share a common surface; (b) The MC-decomposition of the complex consisting of three 2D components; (c) The 2D components are duplicated so that the originals and the duplicates have opposite orientations; (d)-(f) three resultant surfaces that completely bound volumes, among which (d) and (e) are shells, and (f) is the exterior of the shape

any simplex of Σ. The triangles forming a shell may be top simplexes, or faces of tetrahedra.

- *Sheets*, which are maximal manifold-connected two-dimensional components with boundary that do not enclose any region.

- *Solids*, which are maximal manifold-connected subcomplexes formed by tetrahedra.

Figure 7.14(a) shows a half-filled sphere. Its semantics-oriented decomposition yields a solid that is the lower tetrahedralized hemisphere, and a shell that is composed of the dangling triangles and part of the surface of the solid, that together enclose the upper hollow hemisphere.

7.3.4 Computing the Semantics-Oriented Decomposition for a 3-complex

For each simplicial 3-complex Σ, there is a corresponding 2-complex Σ' which consist of the wire-edges and the dangling triangles of Σ, plus the set of triangles that are on the 2D boundary of the tetrahedral components of Σ.

The complex Σ' can be obtained from Σ by replacing each tetrahedral component t in Σ with its 2D boundary δt. We call Σ' the *reduced complex*. For the example of the half-filled sphere shown in 7.14(a), the reduced complex consists of a sphere with an internal surface partitioning it into halves. Figure 7.14(b) shows the boundary of the half-filled sphere which is added to the remainder of the MC-decomposition (c) to form the reduced complex shown in Figure 7.14(d). The

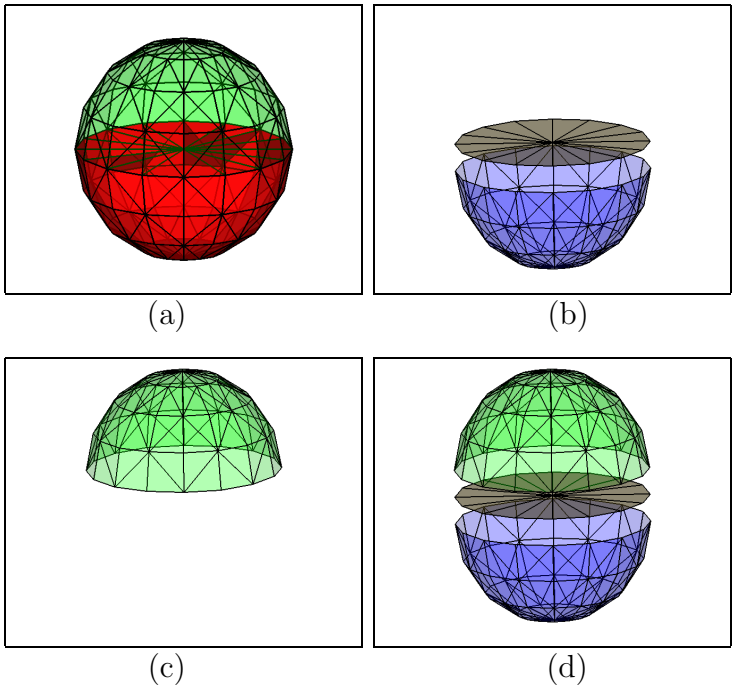


Figure 7.14: Example of components in the semantics-oriented decomposition of a 3-complex: (a) A complex Σ with empty upper hemisphere and solid lower hemisphere; (b) the solid part of Σ is replaced by its 2D boundary, and it is added to the remaining component shown in (c); (d) All the parts of the reduced complex Σ' of Σ.

semantics-oriented decomposition of the 3-complex Σ may be computed in three steps:

1. Identify the set $T = \{t_1, ...t_k\}$ of k tetrahedral components of Σ and extract the set $S = \{\delta t_1, ..., \delta t_k\}$, where δt_i is the 2D boundary of component t_k in T.

2. Construct the complex Σ' as the union of the set S, and the set of wire-edges and the dangling triangles of Σ.

3. Compute the semantics-oriented decomposition of the 2-complex Σ',

4. For each shell s in Σ', if s corresponds to the boundary of some tetrahedral

component t in Σ, replace s by t.

The computation of the 2D boundary of each tetrahedral component in Σ takes time that is linear with respect to the number of tetrahedra in the component. Note that the number of triangles on the 2D boundary of each tetrahedral component t is of the same order as the number of tetrahedra in t itself. Thus, this algorithm has the same complexity as the computation of the semantics-oriented decomposition of Σ'.

7.4 The Decomposition Graph[3]

In this Section, we introduce a graph-based representation for both the MC- and the semantics-oriented decompositions discussed, respectively, in Sections 7.2 and 7.3. This representation captures the complexity of the connectivity among the components and supports the extraction of interesting global topological features of the decomposed complex.

Both the MC-decomposition and the semantics-oriented decomposition can be described as a hypergraph $H =< N, A >$, that we call the *decomposition graph*, in which the nodes in N correspond to the components in the decomposition, while the hyperarcs in A capture the structure of the connectivity both among and within the components. Hyperarcs that are self-cycles represent the non-manifold structure of

[3]Originally published in [45]. Reproduced with notice of ACM copyright: permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honoured. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

a single component. The hyperarcs that connect distinct components are defined as follows. Any k components $C_1, C_2, \cdots, C_k$ in the decomposition with $k > 1$ such that the intersection J of all such components is a not empty, and J is common only to the k components, defines one or more hyperarcs with extreme nodes in $C_1, C_2, \cdots, C_k$. The intersection of components $C_1, C_2, \cdots, C_k$ consists of isolated non-manifold vertices, maximal connected 1-complexes formed by non-manifold edges, or, only when we consider a semantics-oriented decomposition, maximal 1-connected 2-complexes formed by triangles. A hyperarc is a connected component of such intersection. Thus, we classify hyperarcs as follows:

- 0-hyperarcs, which consist only of one non-manifold vertex;
- 1-hyperarcs, which are maximal connected 1-complexes formed by non-manifold edges;
- 2-hyperarcs, which are maximal 1-connected 2-complexes formed by triangles.

A 2-hyperarc exists only in the graph describing the semantics-oriented decomposition and it may connect only two nodes, because a connected set of triangles may belong to at most two shells. Note that there may exist one or more hyperarcs connecting the same set of nodes. All hyperarcs which connect the same components $C_1, C_2, \cdots, C_k$ can be grouped into a *macro-hyperarc*, which thus defines the structure of the intersection of the k intersecting components.

An example of a decomposition graph is shown in Figure 7.15. The semantics-oriented decomposition of the object, shown in Figure 7.15(a), consists of two shells, C_1 and C_2. The connectivity between C_1 and C_2 is shown in Figures 7.15(b) and

(c). C_1 and C_2 share vertex v (shown in Figure 7.15(b)) and the four edges e_1 to e_4 (shown in Figure 7.15(c)). Thus, there are two hyperarcs: a 0-hyperarc defined by the standalone vertex v and a 1-hyperarc defined by the sequence of edges $(e_1, \cdots, e_4)$. Figure 7.15(d) shows the hypergraph with the macro-hyperarc (labeled a) connecting components C_1 and C_2. Note that the number of hyperarcs between the two components is related to the number 1-cycles in the object. In this case, the two components form a 1-cycle.

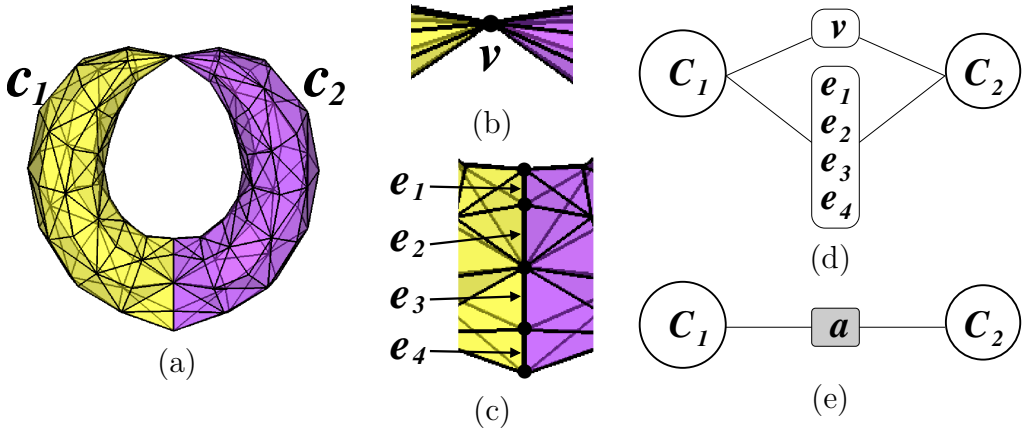


Figure 7.15: An example showing the connectivity between two shells: (a) the semantics-oriented decomposition consists of two shells C_1 and C_2; (b) connection at vertex v; (c) connection through a chain of four edges e_1 to e_4; (d) the hypergraph showing two hyperarcs which describe vertex v and the chain of edges e_1, e_2, e_3, e_4 connecting components C_1 and C_2; (e) the hypergraph at a higher level, showing the macro-hyperarc a, composed of the two hyperarcs connecting C_1 and C_2.

Since a component C may contain non-manifold singularities, we represent C in the decomposition graph with a node and with self-cycles corresponding to the non-manifold vertices and non-manifold edges. A 0-hyperarc corresponds to a non-manifold vertex belonging to C, while a 1-hyperarc to a maximal connected

1-complex formed by non-manifold edges all belonging to C. Figure 7.16(a) shows an example of a non-manifold vertex within a pinched torus. The pinched torus is a manifold-connected 2-complex. The graph describing the non-manifold connectivity of the shape is shown in Figure 7.16(b). Figure 7.16(c) shows an example of a non-manifold edge within a pinched duster. The pinched duster is a manifold-connected 2-complex.

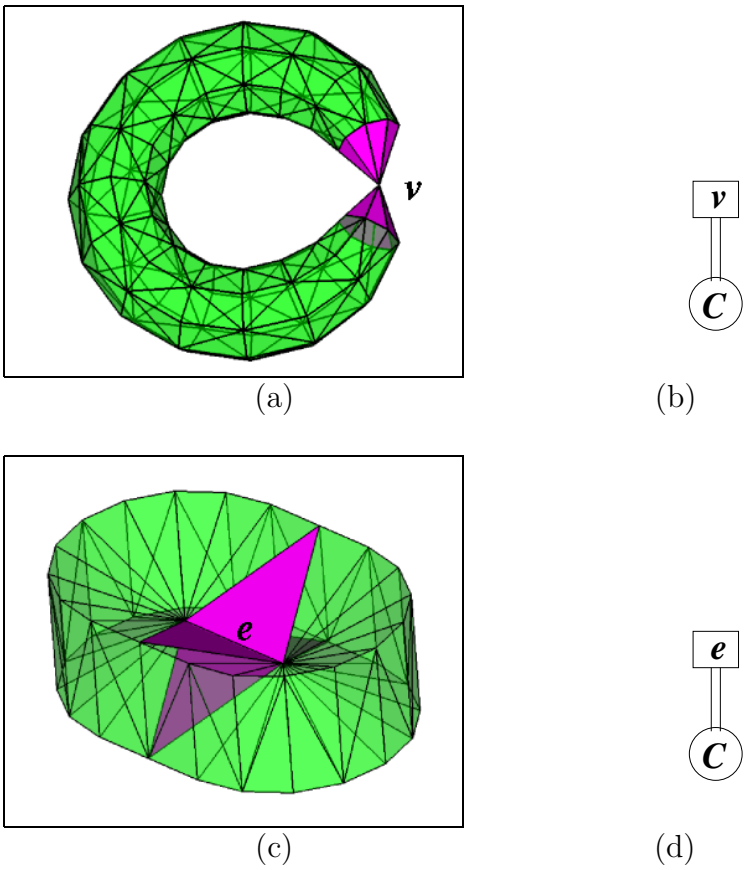


Figure 7.16: (a) Example of a manifold-connected 2-complex with a non-manifold vertex. The neighborhood of the non-manifold vertex v is highlighted; (b) The graph describing the non-manifold vertex internal to the complex; (c) An example of a manifold-connected 2-complex with a non-manifold edge. The four triangles incident at the non-manifold edge e are highlighted; (d) The graph describing the non-manifold vertex internal to the complex.

7.5 Form Feature Identification

The design of a product is constrained to satisfy several requirements specified by the customer. In the design process, it is necessary to conduct several analysis from multiple perspectives in order to ensure that all the requirements are met. Qualitative attributes are generally attached to the geometric framework to provide such perspectives. On top of it, it is also instrumental to develop a feature-based description of the model which offers a channel to associate functional information to geometric data [10]. An example of such an application is shown in Figures 7.17(a)-(c).

It is common for the models to be described as simplicial 2-complexes. Furthermore, in the process of abstraction, mixed dimensionality and non-manifold connectivities may occur in the simplified model. In collaboration with IMATI (Genova), we investigated the possibility of using the decomposition of non-manifold objects into parts connected at non-manifold joints as a first step to feature identification. The parts in the decomposition can then be classified into a predefined set of features. One such category of feature-based description is the *form features* in Finite Element model. The components of a manifold shape can be classified according to a set of form features documented in the STEP (Standard for Exchange of Product Data, ISO 10303). The work [10] proposes a taxonomy of form features for non-manifold objects, extending the STEP classification. These features fall into two categories: added parts and subtracted parts. The features, that add parts to a

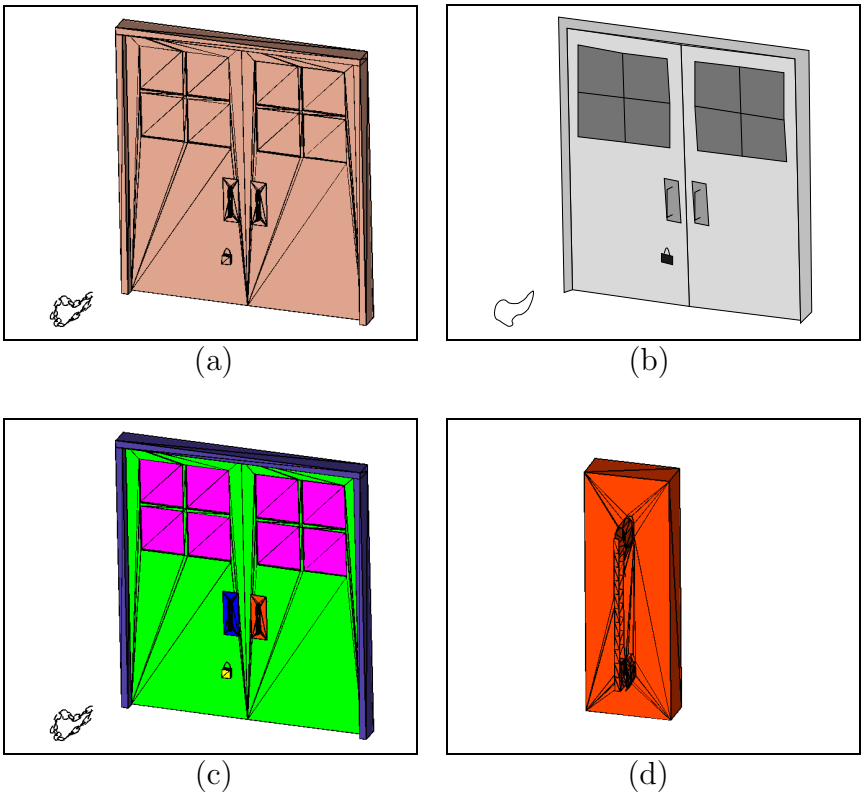


Figure 7.17: (a) A detailed geometric model of the door; (b) An idealized model of the door with identifiable parts which may be associated with features describing functional information; (c) An association between the functional information and the geometric description of the parts of the shape; (d) A part of the model that is associated with the function of a door handle

shape, include *protrusions*, *connectors*, *handles* and *standalones*. The features, that subtract parts from a shape, include *cavities* and *through holes*. Form features that substract parts from the object may be considered as the negation of those that add parts to it. In the non-manifold taxonomy, a part is a 1-dimensional, 2-dimensional or 3-dimensional connected and compact subset of the object that has a meaningful semantics in the application context. These parts are defined according to [10] as follows:

Connector: is a k-dimensional part of object that, if removed, splits the object into

two or more connected m-dimensional parts, where $m \geq k$;

Handle: is a k-dimensional part of object that intersects only one other m-dimensional part, with $m \geq k$, in two or more connected portions of their common boundary, and it is external to this other part;

Connector-handle: is a k-dimensional part of an object which intersects at least two other m-dimensional parts, where $m \geq k$, and whose removal does not break the object into disconnected elements;

Through hole: is a k-dimensional part of object that intersects only one other m-dimensional part, with $m \geq k$, in two or more connected portions of their boundary, and is internal to this other part;

Protrusion: is a k-dimensional part of object that intersects only one other m-dimensional part, with $m \geq k$, in one connected portion of the boundary, and is external to this other part;

Cavity: is a k-dimensional part of object that intersects only one other m-dimensional part in one connected portion of the boundary, and is internal to this other part, with $m \geq k$;

Standalone: part of object that is not a feature of any other part and can be considered as independent.

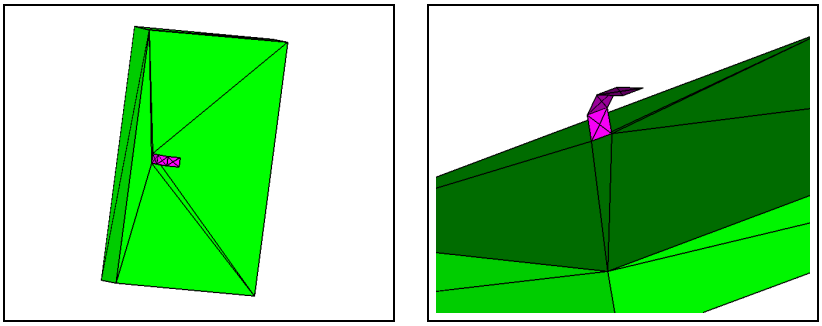


Figure 7.18: Non-manifold objects with identifiable form features (part 1): Two views of a door with a lamina door-handle. The door-handle is considered as a protrusion to the door, which is a standalone object

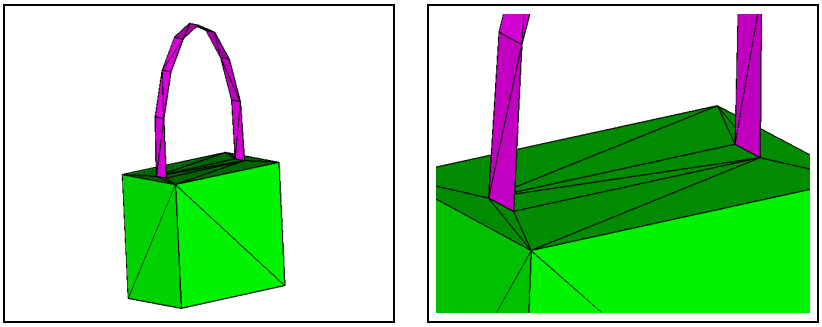
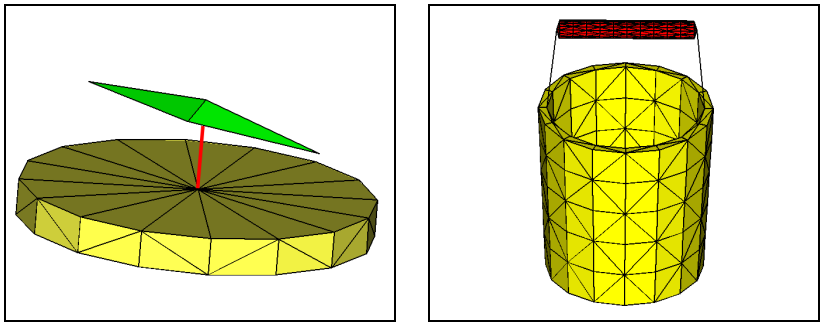


Figure 7.19: Non-manifold objects with identifiable form features (part 2): Two views of a lock; The upper part of the lock forms a handle on the lower part.

Figure 7.18, Figure 7.19, and Figure 7.20 give examples to illustrate each non-manifold form feature. Figure 7.18 shows a door that is composed of two parts: the door and the door-handle. The door-handle is a protrusion. The door is a standalone component in the model. The lock modeled by a lamina attached to a block shown in Figure 7.19 illustrates the handle feature. Figure 7.20(a) shows a compass that consists of three parts: the base, the thin wire supporting the magnetized needle and the magnetized needle itself. The thin wire is a connector. In Figure 7.20(b), the bucket consists of four parts. The wires of bucket are connector-handles. For



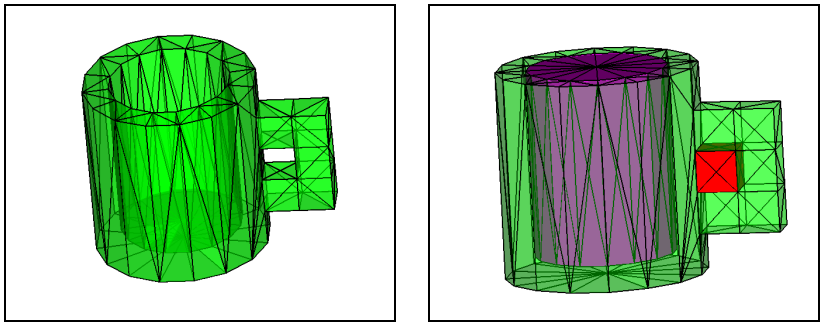
(a) (b)

Figure 7.20: Non-manifold objects with identifiable form features (part 3): (a) A compass in which the pivot supporting the needle is a connector between the magnetized needle and the base of the compass; (b) A bucket that is formed by four mixed-dimensional components.

visual clarity, we use manifold objects in Figures 7.21(a)-(b) to illustrate the features of through holes and cavities. Figure 7.21(a) illustrate a mug with a cavity and a through hole. Figure 7.21 (b) shows the volumes that are subtracted to form the cavity and the through hole of the mug.

We consider here only the form features formed by addition. The identification of form features requires the availability of multiple sources of information on the shape. Geometric information include the size and relative position of the parts. Topological information includes dimension of the parts, and the connectivity among them. Protrusions, connectors and handles are local features that can be determined based on the connectivity between adjacent parts, while connector-handles are global features whose identification require an examination of the whole model. We investigate how the semantics-oriented decomposition (see Section 7.3) can be used as a starting point to extract the topological information.

In the next two Sections, we discuss the use of the semantics-oriented de-



(a) (b)

Figure 7.21: Manifold objects with identifiable local form features: (a) A mug with a cavity and a through hole; (b) The volumes that are substracted from the mug to form the cavity and the through hole shown in (a)

composition for identifying local form features; then we discuss certain interesting properties of the decomposition graph that open up the door to further research on the global structure of the non-manifold shape.

7.5.1 An Interpretation of Semantics-Oriented Decomposition

In the semantics-oriented decomposition of a simplicial 2-complex, we can consider the shells as implicit representations of solids (even though in some cases they really only represent holes) and consider them as *three-dimensional components*. Sheets are considered as *two-dimensional compoents* while wire-webs are *one-dimensional*. These components are directly obtainable from the semantics-oriented decomposition. Based on the component dimension, a hierarchy may be created among the components such that the component with the lower simplicial dimension is potentially considered to be a feature of the one with the same or a higher dimension. In the case where two components are of the same dimension,

there is a tie that cannot be resolved without geometric information such as the size of the two.

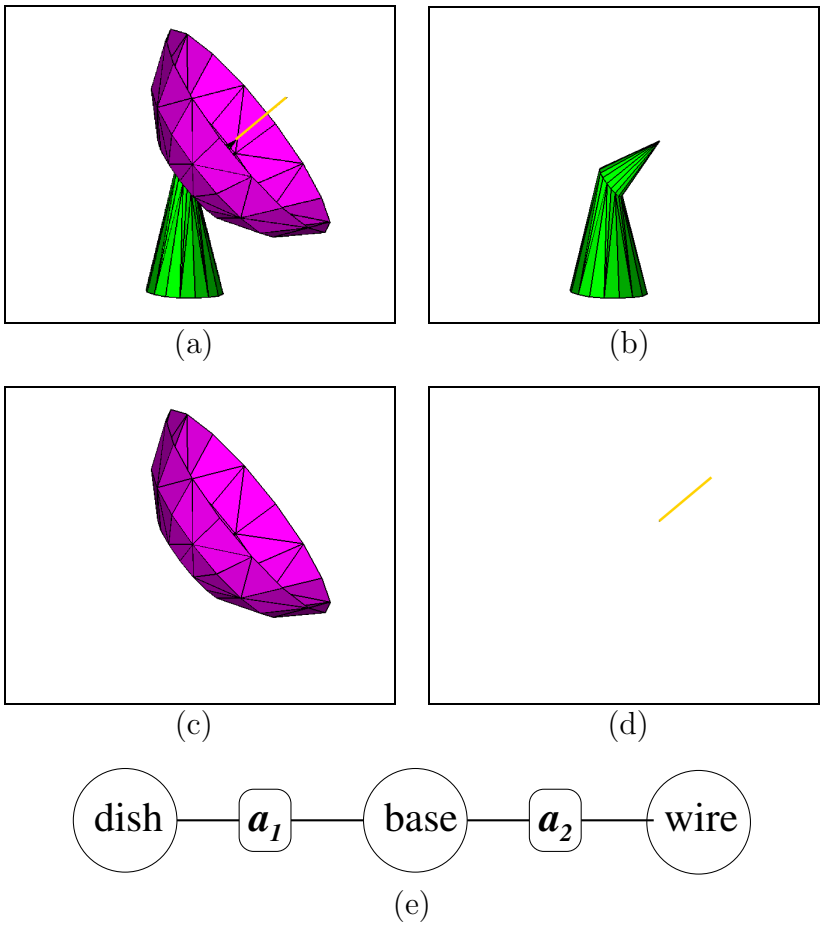


Figure 7.22: (a) An antenna model in which the base is a shell, the dish is a sheet and the antenna is a wire-web; (b) The base; (c) The dish; (d) The wire; (e) The graph describing the connectivity among the three parts: arc a_1 represents the set of non-manifold edges connecting the dish and the base, while arc a_1 represents the non-manifold vertex shared by the base and the wire.

In addition to dimension, the graph describing the semantics-oriented decomposition offers information about connectivity among the components. See for example the antenna model shown in Figures 7.22(a)-(d). The graph of the decomposition

of the antenna describes the connectivity among its three parts (see Figure 7.22(e)).

Combining the properties in the decomposition graph with information on the dimensions of the components, we are able to identify properties that have correspondence with form features. Let p and q be nodes in the decomposition graph. We know that:

1. if p is an end node and it has lower dimension than its neighbouring node, then p is a *protrusion*;

2. if p is an articulation node which does not belong to any cycle and has a lower dimension than one of its neighbour, then p is a *connector*;

3. if two nodes, p and q form a simple cycle, and q has a higher dimension than q, then p is a *handle*.

For example, the wire component in the graph of Figure 7.22(e) is classified as a protrusion to the dish component. Figures 7.23(a)-(b) describe a compass and its decomposition graph. Component B (the wire) is a connector between components A and B because it is an articulation node, not part of any cycle, and its dimension is lower than those of its neighbours. Figures 7.24(a)-(c) describe the decomposition graph of the lock model shown in Figure 7.19. Component A (the lamina) is a handle to component B (the block) because the two components form a simple cycle and A has a lower dimension than B.

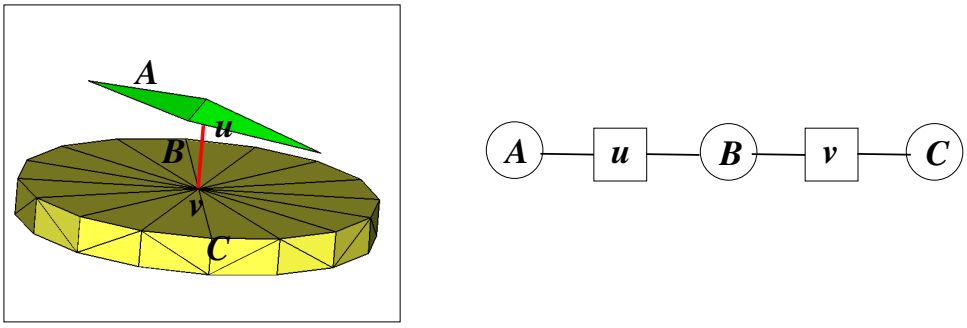


Figure 7.23: (a) A compass described by component A (a magnetized needle) connected to component B (a wire) at vertex u, and component B is set on component C (the base) at vertex v; (b) The graph describing the connectivity of the compass

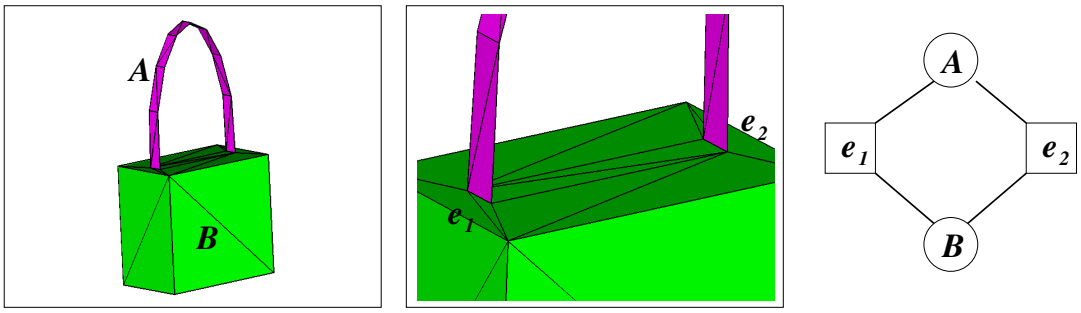


Figure 7.24: (a) A lock described by component A (a lamina) connected to component B (a block); (b) The two non-manifold edges e_1 and e_2 connecting components A and B; (c) The graph describing the connectivity of the lock

7.5.2 Further Observations on the Decomposition Graph

In this Section, we make further observations on the decomposition graph which are pointers to future research. The cycles in the graph are formed by a circular list of alternating nodes and arcs. They describe both local and global features of the shape. There are three types of cycles that reflect interesting structural properties in the shape: *cycles of overlapping joints*, *self-cycles* and *multi-component simple cycles*, which we elaborate below.

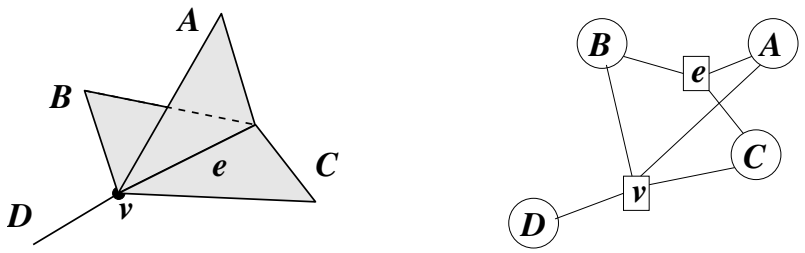


Figure 7.25: Example of a cycle of overlapping joints in the decomposition graph: (a) Non-manifold vertex v which is a hyperarc shared by components A, B, C and D, is part of the non-manifold edge e shared by A, B and C; (b) Non-manifold edge e shared by components A, B and C is part of the surface f shared by A and B.

Cycles of overlapping joints When multiple components are connected with overlapping joints, this is reflected in the decomposition graph as simple cycles of the form $< C_1, a_1, C_2, a_2, C_1 >$, where C_1 and C_2 are components and the hyperarc a_1 represents a joint that is a subset of that of a_2. An example is shown in Figures 7.25a)-(b). In this example, the non-manifold edge, e is shared by components A, B and C while one of its extreme vertices, v, is shared by components A, B, C and D. The simple cycles such as $< A, v, B, e, A >$, where v is a subset of e, reflect the local feature at the vicinity of non-manifold edge e and non-manifold vertex v.

Self-cycles Self-cycles are cycles that consist of just one component and one arc. They correspond to non-manifold folding within a component, which can be classified into the internal case and the external case. Internal folding applies only to shells. In the internal case, the 3D space that is considered the interior of the shell is continous in the neighbourhood of the non-manifold singularity. The folding is

external otherwise. External folding contributes to the formation of handles inside the component. Two examples of folding is shown in Figures 7.26(a) and (b). Figures 7.26(c) and (d) shows the 3D space in the neighbourhood of the non-manifold vertices in the examples of (a) and (b).

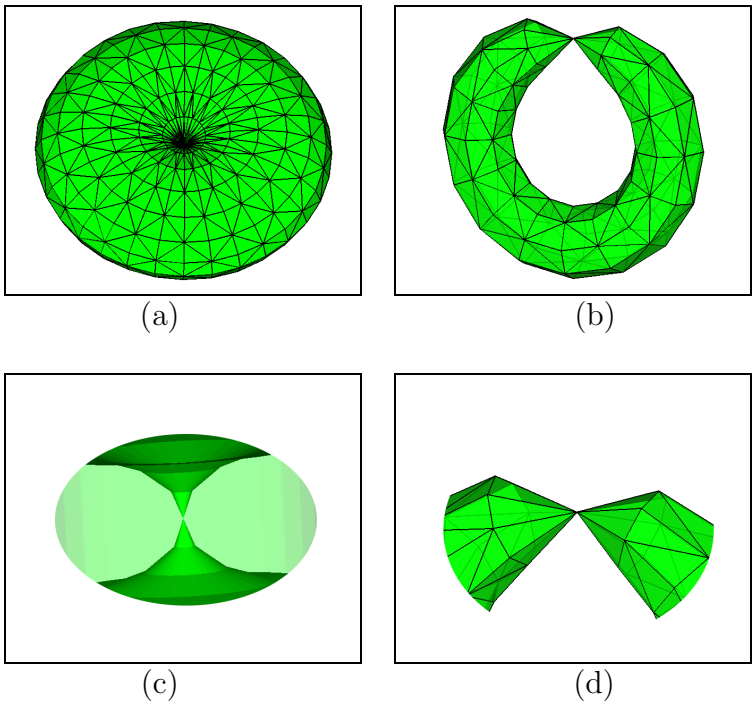
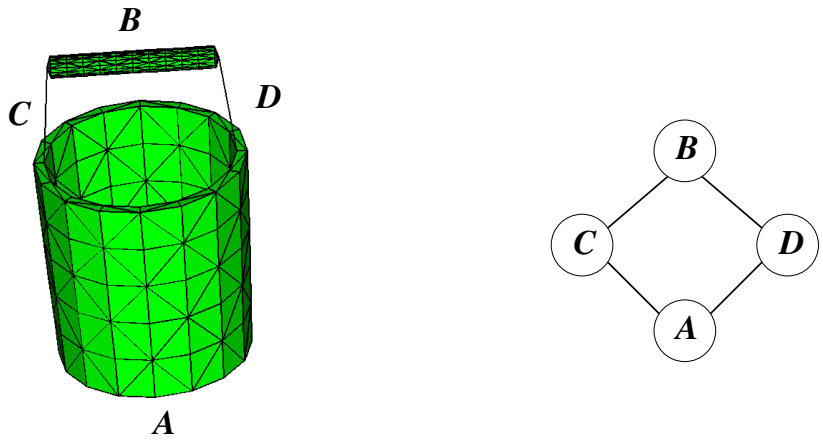


Figure 7.26: Non-manifold folding: (a) A pinched ball (internal folding); (b) A pinched torus (external folding); (c) the continuous 3D space in the neighbourhood of the non-manifold vertex in (a) indicating an internal folding; (d) two disjoint 3D spaces in the neighbourhood of the non-manifold vertex in (b) indicating an external folding

Multi-component simple cycles Certain simple cycles in the graph, that do not correspond to overlapping joints and self-cycles, correspond to some loops in the object. An example is shown in Figures 7.27(a)-(b), in which the decomposition graph (b) of the bucket model (a) describes a loop in the model. This is, by far, the

most interesting property of the decomposition graph which makes it a possible non-manifold correspondence to the reeb graph representation for the manifold shape. It remains an open question at this stage how all such cycles can be identified.



(a) (b)

Figure 7.27: An example of multi-component cycle: (a) A bucket with four components; (b) The graph of the semantics-oriented decomposition of (a).

7.6 Web-based shape retrieval using topological characteristics as the ontology

In recent years, large number of multimedia data are generated from industrial, research and personal sources. This creates the need for these data to be organized in an intelligent way, that enables the retrieval of these data using different types of information descriptors. To create an intelligent database requires strong integration of knowledge management technologies. A first step to this integration involves an ontology of shapes, that represents a collection of concepts describing the shapes

and the relationships among these concepts.

The understanding of a shape has been a challenging problem for a while. Previous approaches have generally taken on the directions of machine learning based on statistical analysis of geometric data. Some approaches also overlap with pyschological studies. The common ground shared by many researchers is that a shape understanding is multifacetal. A shape can be described at three levels: geometric, structual and conceptual. Geometrically, a shape is described as a collection of elementary cells, such as triangles, edges and vertices, which captures the boundaries of the volume encapsuled by the shape. A structural representation is a more concise description of a shape in which geometric details are abstracted and only important features remain. Thus, it is a suitable basis for semantic annotation and reasoning. Examples of structural representations are skeleton-based descriptions, or part-based decompositions.

The availability of geometric and topological information is instrumental to the construction of the conceptual model of a shape. In particular, a structural representations guided by topological information (such as the semantics-oriented decomposition proposed in Section 7.3) is essential for inferring semantic properties. To this aim, the Common Shape Ontology has been proposed [3] as the first step to shape understanding. In collaboration with DISI-Genova, we proposed a system based on this ontology, that is capable of exploring, organizing and understanding digital representations [21]. The development of this work has been published in [22, 66] In the following, we describe this proposed BeSmart system. We also report a tool that we have built for the topological component of this system.

7.6.1 be-SMART (BEyond Shape Modeling for understAnding Real world representations)

Be-SMART (BEyond Shape Modeling for understAnding Real world representations) [22, 66] is a Java-based system, designed for geometric-topological inspection and semantic annotation and structuring of 3D shapes. The two purposes of be-SMART are to extract quantitatively replicable information about features and regions of interest, and and to provide an intuitive interface for reasoning on digital models, which thereby enables the generation of ontology-driven metadata about an object. This is achieved by (i) extracting (automatically) geometric and topological information from the model and by maintaining them using ontology-driven metadata; by (ii) segmenting the model (both manually and/or automatically) using editing technologies and context-dependent segmentation techniques and by (iii) structuring and idealizing (automatically) the shape in order to create a structural multi-level representation of the model guided by the associated semantic. The system can be coupled with a semantic web portal, which provides metadata management and interaction functionalities. Be-SMART consists of the following five modules:

1. *Geometry and Topology Analyzer (GTA)*: it analyses the input shape model and extracts geometrical/topological information which is maintained in the enriched shape model and as instance values of a given ontology.

2. *Topological Decomposer (TD)*: starting from the information extracted by the GTA module, this module produces a graph-based representation (decompo-

sition graph) of the shape model into nearly manifold components.

3. *Manual Segmentation module (MS)*: This module offers both simple and advanced editing functionalities allowing a user to select portions of the model. The segmentation is maintained in the decomposition graph.

4. *Automatic Segmentation module (AS)*: This module offers the possibility of applying automatic segmentation algorithms for decomposing the manifold components into meaningful parts (according to context-dependent criteria). The segmentation is maintained in the decomposition graph.

5. *Semantic Annotator (SA)*: This module offers the possibility of associating specific metadata values to specific portions of the decomposed model according to pre-loaded ontologies. Basically, it associates metadata with nodes of the decomposition graph which describe the decomposed model.

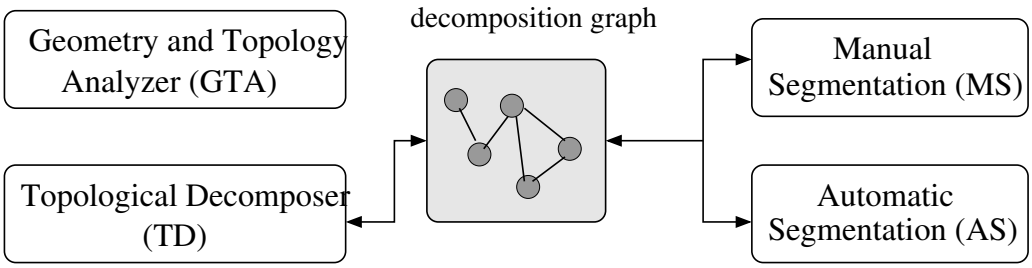


Figure 7.28: The general architecture of be-SMART, with its five constituent modules. The GTA module exchanges information with all the other modules. Both the GTA and the TD modules are application-independent. The other modules are context-dependent

The interactions among the modules in the be-SMART architecture are shown in Figure 7.28. We contribute to the be-SMART system primarily in the definition and the construction of the Geometry and Topology Analyzer (GTA) module. The Geometry and Topology Analyzer (GTA) addresses the problem of extracting topological characteristics from non-manifold 3D shapes described as simplicial 2-complexes containing parts of different dimensions.

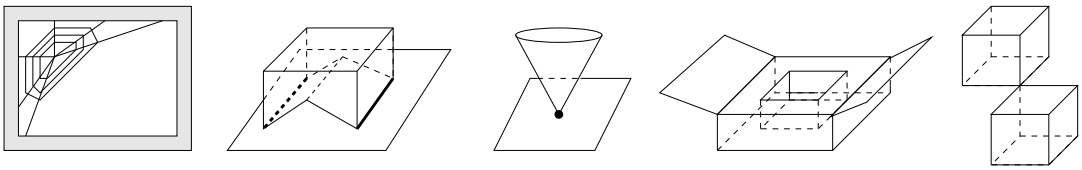


Figure 7.29: Non-manifold features handled by the GTA: (a) A spider-web on a window; (b) a block that touches a plane at two straight-lines; (c) a cone touching a plane at a single point; (d) an object that encloses one void, that is it has a shell, and contains two 1-cycles, which define the handle; (e) an object with two shells, each of which is the interior of a cube.

As we have discussed in Chapter 3, the basic characteristics of a non-manifold shape that are relevant to topological analysis are: non-manifold singularities, which can be non-manifold isolated points, or non-manifold curves (see the two examples in Figure 7.29(b) and (c)). Additionally, as mentioned in Chapter 7, a non-manifold shape may be described as parts with certain dimension and degree of connectivity. Such parts include: wire-webs (see Figure 7.29(a)); connected components of the shape; and maximal connected components which do not contain non-manifold isolated points (such as the objects in Figure 7.29(d) and (e)). Note that the object

in Figure 7.29(c) is formed by two of such parts (the cone and the planar surface). These characteristics can be expresses as the following quantitative features:

1. non-manifold isolated vertices and non-manifold edges, which correspond to the non-manifold singularities of the shape;
2. wire-edges;
3. connected components;
4. wire-webs (see their definition in Section 7.3.1) which describe the 1-dimensional parts of a shape (see the web in Fig. 2(a))
5. 1-connected components. Note that a 1-connected component is a component in which, for every pair of triangles, there exists a path composed of triangles and edges such that any edge in the path belongs to the boundary of the two triangles preceding and following it in the path.

In addition, a topological signature for a non-manifold shape can be defined on the numbers of its non-manifold singularities (isolated points and curves), the numbers of different components listed above, and on its *Betti numbers*, β_0, β_1 and β_2, which are the number of connected components, the number of 1-cycles and shells in the shape, respectively [1]. The Betti numbers are related through the Euler-Poincare's formula: $V - E + F = \beta_0 - \beta_1 + \beta_2$, where V, E and F denote the number of the vertices, edges and triangles, respectively. The list of properties to be extracted by the GTA module and the corresponding metadata described in the Common Shape Ontology is shown in Table 7.1.

Information extracted by the GTA	NonManifoldMesh metadata
# triangles	hasNumberOfCell
# vertices	hasNumberOfVertice
# edges	hasNumberOfEdge
# wire-edges	hasNumberOfWireEdges
# non-manifold vertices	hasNumberOfNonManifoldIsolatedVertex
# non-manifold edges	hasNumberOfNonManifoldEdge
# connected components made of wire-edges	hasNumberOfConnectedSimplexes-OfDim1
# connected components made of triangles	hasNumberOfConnectedSimplexes-OfDim2
β_0 (number of connected components)	hasNumberOfConnectedComponents
# 1-connected components	hasNumberOf1ConnectedComponents
β_1	hasNumberOf1Cycles
β_2 (number of shells)	hasNumberOf2Cycles

Table 7.1: Correspondence between properties extracted by the GTA module and the metadata described in the Common Shape Ontology for the concept of non-manifold mesh *(NonManifoldMesh)*

To fulfill the objective of the GTA module we developed a topological analysis tool, that we called *TopMesh*, for the extraction of topological properties. We describe TopMesh in the following Section.

7.7 TopMesh: A Tool for Topological Analysis of Non-manifold Shapes

Simplicial meshes are the most common representation for 3D digital shapes in a variety of application domains. A 3D shape is most commonly described through a discretization of its boundary into a simplicial mesh consisting of triangles, bounded by edges and vertices, and by wire-edges, which are edges that do not bound any triangle. Here, in collaboration with May Huang, we address the problem of extracting topological characteristics from non-manifold 3D shapes containing parts of different dimensions and discretized as simplicial meshes. While there exist tools to extract geometric and topological information from manifold shapes, much less work exists on extracting such information from non-manifold ones. An example of a tool for extracting such information from manifold shapes is provided by the *TriMeshInfo* tool currently used in the Shape repository of AIM@SHAPE to extract shape metadata only for manifold shapes. Thus, we have developed algorithms for extracting non-manifold singularities, parts of a 3D shape with different degrees of connectivity, and with different dimensions from a discretization of the shape as a simplicial 2-complex embedded in the three-dimensional Euclidean space.

All the algorithms have been implemented into a tool for topological analysis of non-manifold meshes, called *TopMesh*, that we are applying to generate metadata for the shapes in the AIM@SHAPE shape repository [2]. *TopMesh* is based on a representation of the underlying simplicial mesh as a Triangle-Segment (TS) data structure (see Section 4.1.2.2). Recall that the TS data structure is a topological

data structure that encodes triangles, wire-edges and vertices, along with the following set of topological relations: boundary $R_{2,0}$ relation for triangles, boundary $R_{1,0}$ relation for wire-edges, partial co-boundary $R^*_{0,1}$ and $R^*_{0,2}$ relations for vertices, partial adjacency $R^*_{2,2}$ relation for triangles. The encoding of this set of topological information is sufficient to enable efficient navigation in the complex. *TopMesh* extracts the topological characteristics of the shape, in the form of metadata specified in the Common Shape Ontology (see Table 7.1 in Section 7.6.1), and computes the semantics-oriented decomposition of the shape (see Section 7.3). Such information obtained for the shape provides a topological signature for the shape useful for shape retrieval.

In the following, we discuss how to extract the metadata enlisted in Table 7.1 from a simplicial 2-complex encoded in the TS data structure. The computation of the numbers of triangles, vertices, edges is trivial and is not described. The identification of non-manifold isolated vertices, non-manifold edges and wire-edges is described in Section 7.7.1. The computations of component-based metadata, namely, the connected components made of wire-edges (i.e., wire-webs), connected components made of triangles, 1-connected components and the Betti numbers β_0, β_1 and β_2 are described in Section 7.7.2. The computation of the semantics-oriented decomposition of a simplicial 2-complex has been described in Section 7.3 and is not repeated here. A detailed description of the TS data structure is found in Section 4.1.2.2.

7.7.1 Computing Non-manifold Singularities, Wire-Edges and Betti Numbers

Wire-edges are explicitly encoded in the TS data structure. A non-manifold isolated vertex v is a vertex such that its link has more than one connected component. This can be identified in the TS data structure by considering the relations $R^*_{0,2}$ and $R^*_{0,0}$ relations at each vertex v. Thus, vertex v is not considered to be non-manifold isolated if only one of these two relations is empty, and

- if the $R^*_{0,2}(v)$ relation is not empty, and it consists only one triangle;
- if the $R^*_{0,0}(v)$ relation is not empty, and it consists of either one or two wire-edges

A non-manifold edge e can be detected by considering a triangle t incident at e and checking whether t has a predecessor and successor in the $R^*_{2,2}$ relation of t at e, which are different.

TopMesh extracts shells and sheets and thus computes the semantics-oriented decomposition of the shape (see algorithm in Section 7.3.2). There is a one-to-one correspondence between a shell and a 2-cycle in a simplicial 2-complex. The value of β_2 is thus equal to the number of shells found in the semantics-oriented decomposition. β_0 is the number of connected components (see Section 7.7.2). β_1 is the number of 1-cycles, and β_1 can be computed through Euler-Poincare's formula given also the numbers of triangles, vertices and edges in the simplicial complex.

7.7.2 Extracting Components of Various Connectivity

TopMesh computes: connected components, connected components made of triangles, connected components made of wire-edges, 1-connected components. The TopMesh also computes the semantics-oriented decomposition of the shape, and thereby computes the Betti numbers of the shape. Here, we first describe how the Betti numbers can be computed. Then we describe how various components are found.

While non-manifold vertices and edges are detected locally by examining their stars through the $R^*_{0,2}$, $R^*_{0,0}$ and $R^*_{2,2}$ relations, detecting parts with certain connectivity properties require a traversal of the simplicial complex. We describe below how we perform this task as a breadth-first traversal in the TS data structure by using a queue as an auxiliary data structure.

7.7.2.1 Extracting Connected Components

The traversal of each connected components starts at an arbitrary unvisited vertex v in the complex, and visits all triangles, or wire-edge incident in v. These are all triangles in the $R^*_{0,2}$ relation at v and all the wire-edges in the $R^*_{0,0}$ relation at v. Then, we consider all the vertices which are bounding such triangles and wire-edges, and if not visited, we insert them in a queue. The vertices bounding a triangle t are obtained through the $R_{2,0}(t)$ relation, while those bounding a wire-edge are retrieved through the $R_{1,0}$ relation. The traversal continues by extracting the first vertex in the queue and considering it as current vertex v. A connected component

is completely traversed when the queue is empty. By repeating this traversal process till no unvisited vertex left in the complex, we retrieve all the connected components.

7.7.2.2 Extracting Connected Components of Uniform Dimensions

Wire-webs are still connected components but formed only of wire-edges. To detect such components, we just perform a similar traversal as the one described above for the connected components, but at each vertex v, we consider only the wire-edges incident in it, defined by the $R^*_{0,0}$ relation at v. Similarly, connected components made solely of triangles can be traversed in the same fashion with the exclusion of wire-edges.

7.7.2.3 Extracting 1-connected Components

The 1-connected components can be computed by considering the regular complex Σ' obtained from the original simplicial 2-complex Σ by eliminating the wire-webs.

The 1-connected components can be extracted from the TS data structure as follows. We consider each connected component of Σ separately. We start from an arbitrary triangle t of a given connected component of Σ. For each e of t, we extract all the triangles incident at edge e from the TS data structure as follows. Let edge $e = (u, v)$, and t_1 be the first triangle after t in the counter-clockwise direction by the right-hand-rule with the thumb pointing in the direction of $v \rightarrow u$. We retrieve t_1 from relation $R^*_{2,2}$ of (t, e) and examine the order of the vertices as

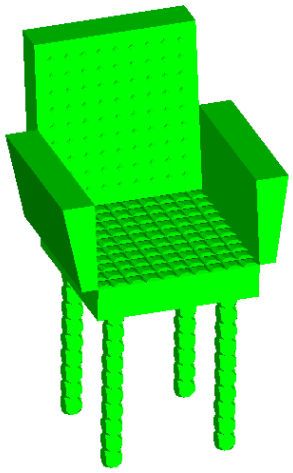
they are encoded in relation $R_{2,0}$ of t_1. If the order of vertices u and v is $v \rightarrow u$ then, we retrieve the first triangle in the counter-clockwise direction in the relation $R^*_{2,2}$ of (t_1, e). Otherwise, we retrieve the first triangle in the clockwise direction. The process is repeated for every triangle retrieved at edge e until we hit t again. We visit all the triangles incident at e and we insert them in a queue. Then, we extract the first triangle from the queue and we repeat the traversal from such triangle. A 1-connected component is completely traversed when the queue is empty. By repeating this traversal process till no unvisited triangle is left in the complex, we retrieve all the 1-connected components.

7.7.3 TopMesh as a Web Service

It has been mentioned in Section 7.6.1 that shape reasoning relies on the availability of geometric and topological information about a shape in the form of metadata. TopMesh has been, in collaboration with May Huang, into a C library. It is available as a web service for AIM@Shape Network of Excellence [2], for the extraction of metadata described in the Common Shape Ontology shown in Table 7.1. Figure 7.30 shows the metadata extracted on a non-manifold data set.

7.8 Summary

In this Chapter, we discussed non-manifold shape decomposition as the first step to the understanding of non-manifold shapes. We proposed a two-level decom-



(a)

NonManifoldMesh metadata
hasNumberOfCell = 10616
hasNumberOfVertice = 5269
hasNumberOfEdge = 15876
hasNumberOfWireEdges = 0
hasNumberOfNonManifoldIsolatedVertex = 0
hasNumberOfNonManifoldEdge = 56
hasNumberOfConnectedSimplexes-OfDim1 = 0
hasNumberOfConnectedSimplexes-OfDim2 = 1
hasNumberOfConnectedComponents = 1
hasNumberOf1ConnectedComponents = 0
hasNumberOf1Cycles = 0
hasNumberOf2Cycles = 8

(b)

Figure 7.30: (a) An armchair model; (b) Metadata reported by TopMesh on the data set

position, in which we first decompose a non-manifold shape based on its manifold-connectedness, then we compute the semantics-oriented decomposition which yields parts of the form: wire-webs, sheets, shells and solids. The semantics-oriented decomposition has been considered as a first step to non-manifold shape analysis. This decomposition approach has been proposed in [45, 18]. Two collaborations started based on this work. They are the identification of non-manifold form features, which is at its preliminary stage of development, and the development of the BeSmart system, as discussed in [21, 22, 66]. In conjunction with May Huang, we developed a non-manifold shape analysis tool, *TopMesh*, which is now available as a web service at the AIM@SHAPE project.

The decomposition-based graph representation of non-manifold shape opens door to research on shape understanding, and has received interest from both Shape Modeling and CAD Engineering communities.

Chapter 8

Conclusion

In this work, we have considered the problem of representing and understanding 3D shapes. We have focused on the simplicial representations of such shapes. We have surveyed the field and identified the areas that are lacking and are in demand. These areas are in the modeling of shapes with parts of mixed dimensions and non-manifold connectivites.

The lack of representations for non-manifold 3D simplicial shapes is due to the lack of understanding on the nature of non-manifoldness in such shapes. Therefore, in this work, we address this lack by providing a characterization of non-manifold properties. Through this characterization, it is possible to design new data structures and to design operators that support shape modification on such data structures.

We proposed four data structures for two different types of applicational needs. The Non-manifold Indexed data structure with Adjacencies (NMIA) is a highly compact data structure for 3D non-manifold shapes. It is suitable for solid modeling applications that handle huge volume of data. Research on this direction has been picked up by the Solid Modeling community since the proposal of the NMIA, which

highlighted the technique of capturing non-manifold singularities in a 3D simplicial complex.

It has been observed that many modeling tools require data structures that are more compact than the existing ones, but have the capacity to associate attributes to all simplexes in the complex. The Simplified Incidence Graph (SIG) and the Incidence Simplicial (IS) data structure are designed for such applicational needs. Both data structures are for simplicial 3-complexes with mixed dimensional parts. These two data structures have smaller storage cost compared with the only existing data structure, the Incidence Graph (IG), for such shapes. These two data structure efficient support for topological navigation and shape modification. We have successfully employed the SIG with the elementary shape modification operator, vertex-pair contraction, in the construction of a multi-resolution model called the Non-manifold Multi-Tesselation.

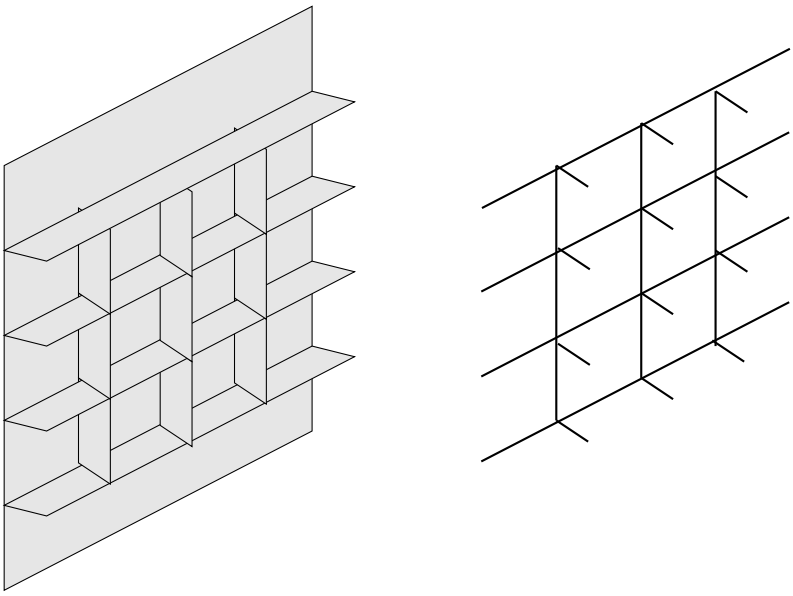
The design of the Double-Level Decomposition (DLD) data structure is a bridge from shape representation to shape analysis through the technique of decomposition. The DLD data structure represents a shape both at the resolution of cells and the resolution of nearly manifold components. The DLD data structure provides a structural description of a non-manifold shape, which opens a door to shape analysis.

We proposed a two-level decomposition of non-manifold shape. The lower level decomposition, that we called the MC-decomposition, breaks a shape uniquely into manifold-connected components. The upper level decomposition, that we called the semantics-oriented decomposition, breaks a shape into wire-webs, sheets, shells

(and pseudomanifold solid in the case of 3-complexes). The semantics-oriented decomposition is computable uniquely from the MC-decomposition. We proposed a hyper-graph that captures the connectivities among the various components of the decomposition of a non-manifold shape. This hyper-graph representation opened up a new frontier to non-manifold shape understanding. We collaborated with two Solid Modeling communities to develop techniques and tools for non-manifold shape analysis.

8.1 Pointers for Future Research

We have shown that shape decomposition is a basic tool for shape reasoning. An example of its use is provided by reasoning on CAD models based on the identification of structural features. One of the characteristics of such models is that they often describe shapes with a complex non-manifold topology. This is especially true when a CAD-generated shape undergoes a so-called idealization process to prepare it for finite element simulation. The non-manifold connectivities in the model can be employed as the signature of the model. The ability to identify and capture non-manifold properties in a shape is thus instrumental to shape characterization. One example is given in Figures 8.1(a)-(c). Figure 8.1(a) shows a surface with a complex supporting structure, and Figure 8.1(b) shows the framework of non-manifold connections in the model of (a), which can be considered as the shape signature of Figure 8.1(a).



(a) (b)

Figure 8.1: Use of non-manifold features: (a) An idealized representation of a surface with supporting structure; (b) the non-manifold edges describing the framework of the supporting structure

Moreover, we foresee that non-manifold shape analysis can be integrated with manifold shape analysis techniques such as the reeb graph to provide a complete description of shapes in general. An example is shown in Figures 8.2(a)-(e). Figure 8.2(a) shows an object that has three tori. The top and the middle tori are connected along a cycle of edges, and thus the two tori form one manifold part (shown in Figure 8.2(b)). The middle torus touches the bottom torus at vertex v (shown in Figure 8.2(c)). The analysis of this shape may first be done first based on the non-manifold properties, thus giving the structural description shown in Figure 8.2(d). Then each component is then analyzed individually using the reeb graph and the overall structural description of the shape is shown in Figure 8.2(e).

Global topological characteristics, such as the betti numbers β_0, β_1 and β_2 in

Euler's Formula have geometric interpretations for manifold shapes. Investigation is needed on how these numbers are geometrically interpretable when applied to non-manifold shapes.

Shape modification is often a related problem to shape representation because most applications need not only a static representation of shapes, but also the ability to operation on them. With the development of shape analysis, it will be necessary to address how to update shapes at the structural level. The literature on this area is lacking. It is particularly interesting to understand how a local modification in shape may affect the global structure of the shape.

Modification on a simplicial shape can be built on elementary operators such as vertex pair contraction if a representation captures only the topology at the level of simplexes. However, when a representation also captures the topology at the level of components (such as the DLD data structure), shape modification involves not only local updates, but also structural updates that results from local change. Structural update is a global operation and is thus costly. One possible solution to keep track of global structure while performing local changes is to assume that global changes occur less frequently than the local ones. Based on such an assumption, the structural representation is recomputed periodically after a sequence of local updates. An example is represent a shape using the NMIA as the underlying data structure for local update, and to construct the DLD data structure when it is necessary to access the structural information of the shape.

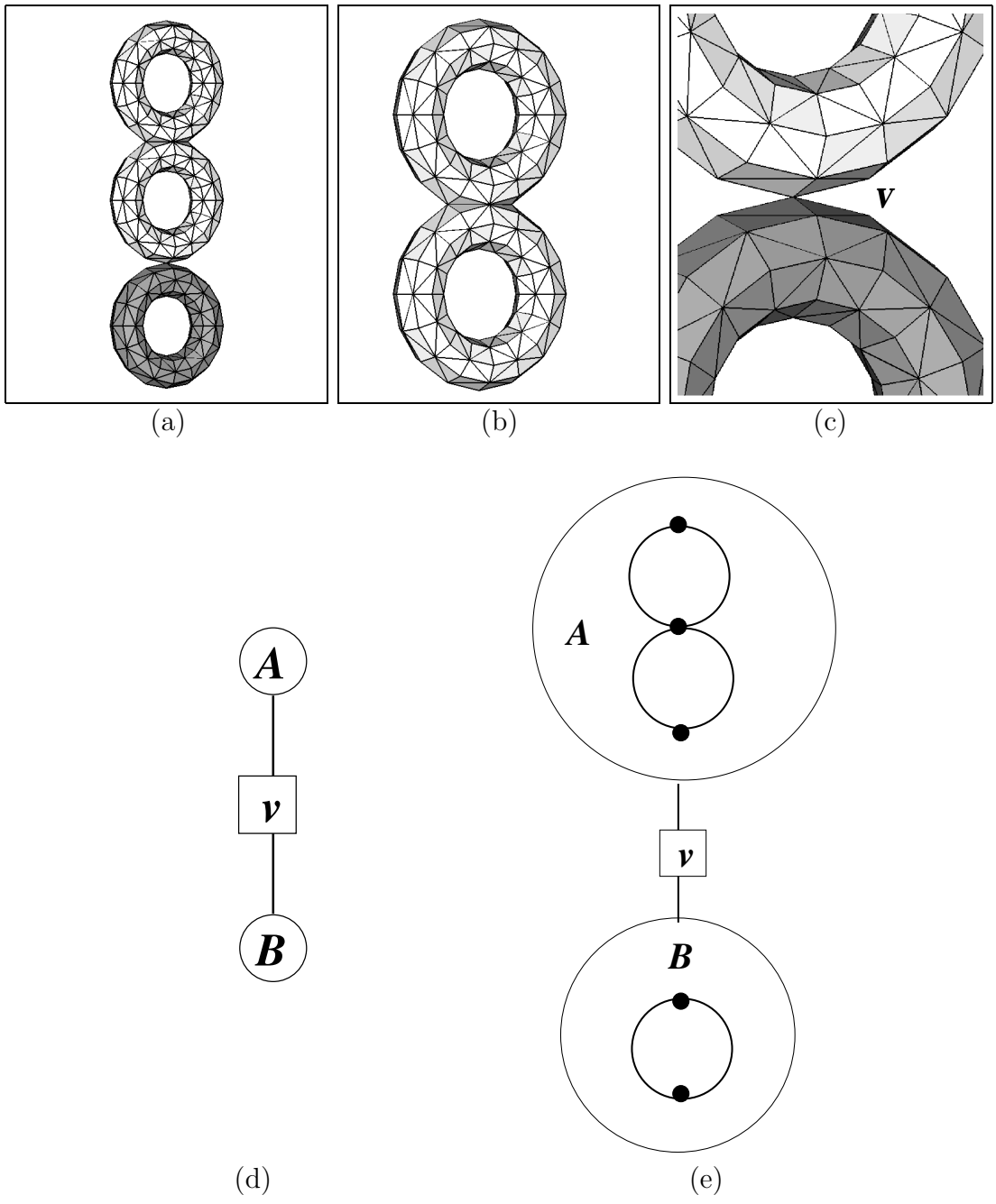


Figure 8.2: The integration of manifold and non-manifold technique on the structural description of a shape

Bibliography

[1] M. Agoston. *Computer Graphics and Geometric Modelling*. Springer, 2005.

[2] NoE AIM@SHAPE. Network of excellence grant n.506766 by the European commission - www.aimatshape.net.

[3] M. Attene, L. Moccozet, T. Hassner, J. C. Leon, R. Sayegh, A. Tal, L. Papaleo, F. Robbiano, M. Gutierrez, O. Andersenn, S. Marini, S. Biasotti, C. Catalano, V. Cheutet, R. Albertoni, A. Belayev, S. Hammann, P. Alliez, P. Cignoni, and M. Pitikakis. Metadata for digital shape models, July 2005. IST-NoE 506766 AIM@SHAPE.

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

[5] B. G. Baumgart. A polyhedron representation for computer vision. In *Proceedings AFIPS National Computer Conference*, volume 44, pages 589–596, 1975.

[6] E. Brisson. Representing geometric structures in D dimensions: topology and order. In *Proceedings 5th ACM Symposium on Computational Geometry*, pages 218–227. ACM Press, 1989.

[7] S. Campagna, L. Kobbelt, and H.-P. Seidel. Directed edges - a scalable representation for triangle meshes. *Journal of Graphics Tools*, 3(4):1–12, 1998.

[8] P. Chopra and J. Meyer. TetFusion: an algorithm for rapid tetrahedral mesh simplification. In *Proceedings IEEE Visualization 2002*, pages 133–140. IEEE Computer Society, October 2002.

[9] P. Cignoni, D. Costanza, C. Montani, C. Rocchini, and R. Scopigno. Simplification of tetrahedral volume data with accurate error evaluation. In *Proceedings IEEE Visualization 2000*, pages 85–92. IEEE Computer Society, 2000.

[10] C. Crovetto, L. De Floriani, and F. Giannini. A first classification of form features in non-manifold shapes. Technical report, University of Genova, Italy, November 2006.

[11] B. Cutler, J. Dorsey, and L. McMillan. Simplification and improvement of tetrahedral models for simulation. In *Proceedings Second ACM/Eurographics Symposium on Geometry Processing*, Nice, France, July 2004.

[12] L. De Floriani, D. Greenfieldboyce, and A. Hui. A data structure for non-manifold simplicial d-complexes. In L. Kobbelt, P. Schroder, and H. Hoppe, editors, *Proceedings of the 2nd ACM/Eurographics Symposium on Geometry Processing*, pages 83–92, Nice (France), 8–10 July 2004.

[13] L. De Floriani and A. Hui. A scalable data structure for three-dimensional non-manifold objects. In L. Kobbelt, P. Schroder, and H. Hoppe, editors, *Proceedings of the 1st ACM/Eurographics Symposium on Geometry Processing*, pages 72–82, Aachen (Germany), 23–25 June 2003.

[14] L. De Floriani and A. Hui. Update operations on 3D simplicial decompositions of non-manifold objects. In D. Fellner, editor, *Proceedings of the 9th ACM Symposium on Solid Modeling and Applications*, pages 169–180, Genova (Italy), 9–11 June 2004.

[15] L. De Floriani and A. Hui. Data structures for simplicial complexes: an analysis and a comparison. In M. Desbrun and H. Pottmann, editors, *Proceedings of the 3rd Eurographics Symposium on Geometry Processing*, pages 119–128, Vienna (Austria), 4–6 July 2005.

[16] L. De Floriani and A. Hui. Representing non-manifold shapes in arbitrary dimensions. In *Proceedings of the 6th Israel-Korea Bi-National Conference on New Technologies and Visualization Methods for Product Development on Design and Reverse Engineering*, pages 11–15, Haifa (Israel), 7–9 November 2005.

[17] L. De Floriani and A. Hui. A dimension-independent representation for multi-resolution non-manifold meshes. *Journal of Computing and Information Science in Engineering*, 7(1):397–404, March 2007.

[18] L. De Floriani and A. Hui. A semantic-oriented decomposition for non-manifold shapes. In *Proceedings of the Israel-Italy Bi-National Conference on Shape Modeling and Reasoning for Industrial and Biomedical Applications*, Haifa (Israel), 7-9 May 2007.

[19] L. De Floriani and A. Hui. Shape representations based on simplicial and cell complexes. In *The Annual Conference of the European Association for Computer Graphics, Eurographics*, 2007.

[20] L. De Floriani and A. Hui. Shape representations based on simplicial and cell complexes. *Computer Graphics Forum, The European Association for Computer Graphics*, 2008. Submitted.

[21] L. De Floriani, A. Hui, and L. Papaleo. Topology-based reasoning on non-manifold shapes. In *Proceedings of the 1st International Symposium on Shapes and Semantics*, pages 23–30, Matsushima, Japan, June 2006. Aim At Shape.

[22] L. De Floriani, A. Hui, L. Papaleo, M. Huang, and J. Hendler. A semantic web environment for digital shapes understanding. In *2nd international conference on Semantics And digital Media Technologies (SAMT)*, Genova, December 2007.

[23] L. De Floriani and P. Magillo. Multi-resolution mesh representation: models and data structures. In M. Floater, A. Iske, and E. Quak, editors, *Principles*

of Multi-resolution Geometric Modeling, Lecture Notes in Mathematics, pages 364–418, Berlin, 2002. Springer Verlag.

[24] L. De Floriani, P. Magillo, F. Morando, and E. Puppo. Dynamic view-dependent multi-resolution on a client-server architecture. *Computer Aided Design*, 32(13):805–823, 2000.

[25] L. De Floriani, P. Magillo, E. Puppo, and D. Sobrero. A multi-resolution topological representation for non-manifold meshes. *Computer-Aided Design Journal*, 36(2):141–159, February 2004.

[26] L. De Floriani, M. M. Mesmoudi, F. Morando, and E. Puppo. Decomposing non-manifold objects in arbitrary dimension. *CVGIP: Graphical Models*, 65(1/3):2–22, January 2003.

[27] L. De Floriani, E. Puppo, and P. Magillo. A formal approach to multi-resolution modeling. In W. Strasser, R. Klein, and R. Rau, editors, *Geometric Modeling: Theory and Practice*, pages 302–323. Springer-Verlag, 1997.

[28] C-J. A. Delfinado and H. Edelsbrunner. An incremental algorithm for betti numbers of simplicial complexes on the 3-sphere. *Computer Aided Geometric Design*, 12(7):771–784, 1995.

[29] T. Dey, H. Edelsbrunner, S. Guha, and D. Nekhayev. Topology preserving edge contraction. *Publications de l'Institut Mathematique (Beograd)*, 60(80), 1999.

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

[31] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. ROAMing terrain: real-time optimally adapting meshes. In R. Yagel and H. Hagen, editors, *Proceedings IEEE Visualization'97*, pages 81–88, Phoenix, AZ, October 1997. IEEE Computer Society.

[32] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Verlag, Berlin, 1987.

[33] J. El-Sana and A. Varshney. Generalized view-dependent simplification. *Computer Graphics Forum*, 18(3):C83–C94, 1999.

[34] B. Falcidieno and O. Ratto. Two-manifold cell-decomposition of R-sets. In A. Kilgour and L. Kjelldahl, editors, *Proceedings Computer Graphics Forum*, volume 11, pages 391–404, September 1992.

[35] L. Fine, L. Remondini, and J.-C. Léon. Automated generation of FEA models through idealization operators. *International Journal for Numerical Methods in Engineering*, 49:83–108, 2000.

[36] P.-M. Gandoin and O. Devillers. Progressive lossless compression of arbitrary simplicial complexes. In *Proceedings 29th annual conference on Computer graphics and interactive techniques SIGGRAPH 2002*, pages 372–379, San Antonio, TX, 2002. ACM Press.

[37] M. Garland. Multi-resolution modeling: survey and future opportunities. In *Eurographics '99 – State of the Art Reports*, pages 111–131. Eurographics Association, 1999.

[38] B. Gregorski, M. Duchaineau, P. Lindstrom, V. Pascucci, and K. Joy. Interactive view-dependent rendering of large isosurfaces. In *Proceedings IEEE Visualization 2002*, San Diego, CA, October 2002. IEEE Computer Society.

[39] M. H. Gross and O. G. Staadt. Progressive tetrahedralizations. In *Proceedings IEEE Visualization'98*, pages 397–402, Research Triangle Park, NC, 1998. IEEE Computer Society.

[40] A. Gueziec, G. Taubin, F. Lazarus, and W. Horn. Converting sets of polygons to manifold surfaces by cutting and stitching. In *Conference abstracts and applications: SIGGRAPH 98*, Computer Graphics, pages 245–245. ACM Press, 1998.

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

[42] 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.

[43] B. Hamann and J. L. Chen. Data point selection for piecewise trilinear approximation. *Computer-Aided Geometric Design*, 11:477–489, 1994.

[44] A. Hui and L. De Floriani. Topological decompositions for 3d non-manifold simplicial shapes. Technical Report CS-TR-4855/UMIACS-TR-2007-10, University of Maryland, College Park, March 2007.

[45] A. Hui and L. De Floriani. A two-level topological decomposition for non-manifold simplicial shapes. In *Proceedings of Solid and Physical Modeling Symposium*, Beijing (China), June 2007.

[46] A. Hui and L. De Floriani. Notes on compact data structrures for finite elements applications. Technical report, Department of Computer Science, University of Maryland, College Park, September 2004.

[47] A. Hui, L Vaczlavik, and L. De Floriani. A decomposition-based representation for 3d simplicial complexes. In *Proceedings of the 4th Eurographics Symposium on Geometry Processing*, pages 101–110, Cagliari (Italy), June 2006.

[48] K. I. Joy, J. Legakis, and MacCracken. Data structures for multi-resolution representation of unstructured meshes. In G. Farin, H. Hagen, and B. Hamann, editors, *Hierarchical Approximation and Geometric Methods for Scientific Visualization*. Springer Verlag, Heidelberg, 2002.

[49] M. Kallmann and D. Thalmann. Star vertices: a compact representation for planar meshes with adjacency information. *Journal of Graphics Tools*, 6(1):7–18, 2001.

[50] M. Lage, T. Lewiner, H. Lopes, and L. Velho. CHF: a scalable topological data structure for tetrahedral teshes. In *18th Brazilian Symposium on Computer Graphics and Image Processing (Sibgrapi 2005)*, pages 349–356, Oct 2005.

[51] S. H. Lee and K. Lee. Partial-entity structure: a fast and compact non-manifold boundary representation based on partial topological entities. In *Proceedings Sixth ACM Symposium on Solid Modeling and Applications*, pages 159–170, Ann Arbor, Michigan, June 2001. ACM Press.

[52] J-C. Leon and L. Fine. A new approach to the preparation of models for FE analyses. *International Journal of Computer Applications in Technology*, 23(2/3/4):166 – 184, 2005.

[53] P. Lienhardt. N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3):275–324, 1994.

[54] H. Lopes and G. Tavares. Structural operators for modeling 3-manifolds. In *Proceedings Fourth ACM Symposium on Solid Modeling and Applications*, pages 10–18. ACM Press, May 1997.

[55] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan-Kaufmann, San Francisco, 2002.

[56] Y. Luo and G. Lukács. A boundary representation of form features and non-manifold solid objects. In *Solid Modeling Foundations and CAD/CAM Applications*, Austin, TX, June 1991. ACM Press.

[57] M. Mantyla. *An Introduction to Solid Modeling*. Computer Science Press, 1987.

[58] 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*, pages 171–182, 2001.

[59] F. Morando. *Decomposition and Modeling in the Non-Manifold domain*. PhD thesis, Department of Computer and Information Science, University of Genova (Italy), February 2003.

[60] E. Mucke. *Shapes and Implementations in Three-Dimensional Geometry*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1993.

[61] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7:217–236, 1978.

[62] A. Nabutovsky. Geometry of the space of triangulations of a compact manifold. *Commun. Math. Phys.*, 181:303–330, 1996.

[63] V. Natarajan and H. Edelsbrunner. Simplification of three-dimensional density maps. *IEEE Transactions on Visualization and Computer Graphics*, 10(5):587–597, 2004.

[64] G. M. Nielson. Tools for triangulations and tetrahedralizations and constructing functions defined over them. In G. M. Nielson, H. Hagen, and H. Müller, editors, *Scientific Visualization: overviews, Methodologies and Techniques*, chapter 20, pages 429–525. IEEE Computer Society, Silver Spring, MD, 1997.

[65] 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.

[66] L. Papaleo, L. De Floriani, J. Hendler, and A. Hui. Towards a semantic web system for understanding real world representations. In *3IA International Conference on Computer Graphics and Artificial Intelligence*, Athens (Greece), 30-31 May 2007.

[67] S. Pesco, G. Tavares, and H. Lopes. A stratification approach for modeling two-dimensional cell complexes. *Computers and Graphics*, 28:235–247, 2004.

[68] J. Popovic and H. Hoppe. Progressive simplicial complexes. In *ACM Computer Graphics Proceedings Annual Conference Series, (SIGGRAPH '97)*, pages 217–224. ACM Press, 1997.

[69] K. J. Renze and J. H. Oliver. Generalized unstructured decimation. *IEEE Computational Geometry and Applications*, 16(6):24–32, 1996.

[70] J. Rossignac and D. Cardoze. Matchmaker: manifold BReps for non-manifold R-sets. In W. F. Bronsvoort and D. C. Anderson, editors, *Proceedings Fifth Symposium on Solid Modeling and Applications*, pages 31–41, New York (USA), 9–11 June 1999. ACM Press.

[71] J. Rossignac, A. Safonova, and A. Szymczak. 3D compression made simple: Edge-Breaker on a Corner Table. In *Proceedings Shape Modeling International 2001*, Genova, Italy, May 2001. IEEE Computer Society.

[72] J. R. Rossignac and M. A. O'Connor. SGC: a dimension-independent model for point-sets with internal structures and incomplete boundaries. In M. J. Wozny, J. U. Turner, and K. Preiss, editors, *Geometric Modeling for Product Engineering*, pages 145–180. Elsevier Science Publishers B. V. (North–Holland), Amsterdam, 1989.

[73] H. Samet. *Foundations of Multi-Dimensional Data Structures*. Morgan-Kaufmann, August 2006.

[74] J. Shewchuk. *General-Dimensional Constrained Delaunay and Constrained Regular Triangulations I: Combinatorial Properties*. 2004.

[75] I. J. Trotts, B. Hamann, and K. I. Joy. Simplification of tetrahedral meshes with error bounds. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):224–237, 1999.

[76] L. Velho and J. Gomes. Variable resolution 4-k meshes: concepts and applications. *Computer Graphics Forum*, 19(4):195–214, 2000.

[77] P. Véron and J. C. Léon. Static polyhedron simplification using error measurements. *Computer-Aided Design*, 29(4):287–298, 1997.

[78] P. Véron and J.-C. Léon. Using polyhedral models to authomatically sketch idealized geometry for structural analysis. *Engineering with Computers*, 17:373–385, 2001.

[79] K. Weiler. *Topological Structures for Geometric Modeling*. PhD thesis, Rensselaer Polytechnic Institute, Troy, N.Y., 1986.

[80] K. Weiler. The radial-edge data structure: a topological representation for non-manifold geometric boundary modeling. In J. L. Encarnacao, M. J. Wozny, and H. W. McLaughlin, editors, *Geometric Modeling for CAD Applications: Selected and Expanded Papers from the IFIP WG5.2 Working Conference, Rensselaerville, NY, USA, 12-16 May 1986*, pages 3–36. Elsevier Science Publishers B. V. (North–Holland), Amsterdam, 1988. ISBN: 0444704167.

[81] H. Whitney. Local properties of analytic varieties. In S. S. Cairns, editor, *Differential and Combinatorial topology, A Symposium in Honor of Marston Morse*, pages 205–244. Princeton University Press, 1965.

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