Tools for Modeling and Analysis of Non-manifold Shapes

by

David CANINO
Tools for Modeling and Analysis of Non-manifold Shapes

by

David CANINO

April, 2012
Ph.D. Thesis in Computer Science (S.S.D. INF/01)

Submitted by David CANINO
DISI, Università degli Studi di Genova
canino@disi.unige.it

Date of submission: February 2012

Title: Tools for modeling and analysis of non-manifold shapes
Advisor: prof. Leila DE FLORIANI
DISI, Università degli Studi di Genova
deflo@disi.unige.it

External Reviewer: dr Franca GIANNINI
Istituto di Matematica Applicata e Tecnologie Informatiche (IMATI-CNR), Genova
franca.giannini@ge.imati.cnr.it

External Reviewer: prof. Jean-Claude LÉON
Institut National Polytechnique de Grenoble (INPG), Grenoble, France
jean-claude.leon@grenoble-inp.fr
Abstract

In this thesis, we address the effective representation of arbitrary shapes, called non-manifold shapes, discretized through simplicial complexes, and we introduce a set of tools for their modeling and analysis.

Specifically, we propose two dimension-independent data structures for simplicial complexes in arbitrary dimensions. The first contribution is the Incidence Simplicial (IS) data structure, based on the incidence relations for simplices of consecutive dimensions. The second contribution is the Generalized Indexed Data Structure with Adjacencies (IA*), based on the adjacency relations for top simplices. The IS and IA* data structures are compact, support efficient navigation, and exhibit a small overhead, if restricted to manifolds. In the literature, there are several topological data structures for cell and simplicial complexes, thus a framework targeted to their fast prototyping is a valuable tool. Here, we introduce the dimension-independent and extensible Mangrove Topological Data Structure (Mangrove TDS) framework. This framework describes any data structure through a graph-based representation, which we call a mangrove. In this thesis, we provide extensive experimental comparisons for several data structures implemented in the Mangrove TDS framework, including the IS and IA* data structures. At the same time, we complete the definition of several data structures, previously proposed in the literature.

In the second part of the thesis, we decompose any non-manifold shape into almost manifold parts in order to deal with its intrinsic complexity. We consider a dimension-independent decomposition of a non-manifold shape, called Manifold-Connected Decomposition (MC-Decomposition), previously investigated only for two- and three-dimensional complexes. Here, we propose several graph-based representations of such a decomposition, which can be combined with any topological data structure. We provide experimental comparisons about building times and storage costs of these data structures.

Recently, the computation of topological invariants, like the simplicial homology, has drawn much attention in several applications. Here, we design and implement the dimension-independent and modular Mayer-Vietoris (MV) algorithm, which exploits the MC-Decomposition for computing the simplicial homology of a non-manifold simplicial shape in arbitrary dimensions. The MV algorithm offers an elegant way for computing the homology of any simplicial complex from the homology of its MC-components and of their intersections.
# Contents

## Chapter 1 Introduction

1.1 Overview ............................................. 7

1.2 Thesis Outline ..................................... 10

## Chapter 2 Background Notions

2.1 Cell Complexes ....................................... 13

2.2 Simplicial Complexes ................................. 15

2.3 Characterization of Non-manifold Singularities .... 16

2.4 Topological Relations on Cell Complexes ............ 18

## Chapter 3 State of the Art

3.1 Dimension-independent Data Structures ............... 22

3.1.1 The Cell-Tuple Data Structure ................... 22

3.1.2 The Incidence Graph ............................... 24

3.1.3 The Extended Indexed data structure with Adjacencies .... 24

3.2 Representations of Cell and Simplicial 2-Complexes ... 25

3.2.1 Representations of Manifold 2-Complexes .......... 26

3.2.2 The Radial Edge Data Structure .................. 30

3.2.3 The Partial Entities Data Structure ............... 32

3.2.4 The Vertex-Face Data Structure ................. 33

3.2.5 The Directed Edges Data Structure .............. 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.6</td>
<td>The Loop-Edge Data Structure</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>Representations of Cell and Simplicial 3-Complexes</td>
<td>34</td>
</tr>
<tr>
<td>3.3.1</td>
<td>The Facet-Edge Data Structure</td>
<td>35</td>
</tr>
<tr>
<td>3.3.2</td>
<td>The Handle-Face Data Structure</td>
<td>37</td>
</tr>
<tr>
<td>3.3.3</td>
<td>The Extended Maps Data Structure</td>
<td>39</td>
</tr>
<tr>
<td>3.4</td>
<td>Decomposition-based data Representations</td>
<td>40</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Decompositions of Regular Shapes</td>
<td>41</td>
</tr>
<tr>
<td>3.4.2</td>
<td>The Combinatorial Stratification</td>
<td>43</td>
</tr>
<tr>
<td>3.4.3</td>
<td>The Initial Quasi-Manifold Decomposition</td>
<td>44</td>
</tr>
<tr>
<td>3.5</td>
<td>Editing Operators</td>
<td>46</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Editing Operators on Cell Complexes</td>
<td>47</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Editing Operators on Simplicial Complexes</td>
<td>47</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Editing Operators and Multiresolution Modeling</td>
<td>49</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Editing Operators for the Idealization Process</td>
<td>51</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Out-of-core Representations</td>
<td>53</td>
</tr>
</tbody>
</table>

**Chapter 4**  
The IS Data Structure and Incidence-based Representations  

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>The Incidence Simplicial (IS) Data Structure</td>
<td>57</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Design of the Data Structure</td>
<td>57</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Implementation and Storage Cost</td>
<td>60</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Retrieving Topological Relations</td>
<td>61</td>
</tr>
<tr>
<td>4.2</td>
<td>The Incidence Graph for Simplicial Complexes</td>
<td>64</td>
</tr>
<tr>
<td>4.3</td>
<td>The Simplified Incidence Graph</td>
<td>66</td>
</tr>
<tr>
<td>4.4</td>
<td>Experimental Comparisons</td>
<td>69</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Experimental Comparisons for Simplicial 2-Complexes</td>
<td>69</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Experimental Comparisons for Simplicial 3-complexes</td>
<td>73</td>
</tr>
</tbody>
</table>

**Chapter 5**  
The IA* Data Structure and Adjacency-based Representations  

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>The Generalized Indexed data structure with Adjacencies</td>
<td>80</td>
</tr>
</tbody>
</table>
Chapter 6  Rapid Prototyping of Topological Data Structures

6.1  The Mangrove Topological Data Structure Framework

6.1.1  Mangroves

6.1.2  Explicit Representation of Simplices

6.1.3  Implicit Representation of Simplices

6.1.4  Operations on Mangroves

6.2  Implementing the Incidence Simplicial Data Structure

6.2.1  Implementation of the Data Structure

6.2.2  Retrieving Boundary of a Simplex

6.2.3  Retrieving Star of a Simplex

6.2.4  Retrieving Simplices Adjacent to a Simplex

6.2.5  Retrieving Link of a Simplex

6.2.6  Recognizing Non-Manifold Singularities

6.2.7  Construction Algorithm

6.3  Implementing the Generalized Indexed data structure with Adjacencies

6.3.1  Implementation of the Data Structure

6.3.2  Retrieving Boundary of a Simplex

6.3.3  Retrieving Star of a Simplex

6.3.4  Retrieving Simplices Adjacent to a Simplex
Chapter 7 Analysis and Comparisons of Topological Data Structures 144

7.1 Implementing the Triangle Segment data structure 145
  7.1.1 Implementation of the Data Structure 145
  7.1.2 Retrieving Non-Manifold Adjacencies 146
  7.1.3 Construction Algorithm 148

7.2 Implementing the Non-Manifold Indexed data structure with Adjacencies 150
  7.2.1 Implementation of the Data Structure 150
  7.2.2 Retrieving Link of a Simplex 151
  7.2.3 Recognizing Non-Manifold Singularities 152
  7.2.4 Construction Algorithm 154

7.3 Implementing the Simplified Incidence Graph 158
  7.3.1 Implementation of the Data Structure 159
  7.3.2 Recognizing Non-Manifold Singularities 159
  7.3.3 Construction Algorithm 161

7.4 Implementing the Incidence Graph 162
  7.4.1 Implementation of the Data Structure 163
  7.4.2 Recognizing Non-Manifold Singularities 163
  7.4.3 Construction Algorithm 166

7.5 Experimental Comparisons 166
  7.5.1 Experimental Analysis of Boundary Retrieval 167
  7.5.2 Experimental Analysis of Star Retrieval 168
  7.5.3 Experimental Analysis of Adjacency Retrieval 172
  7.5.4 Experimental Analysis of Link Retrieval 177
  7.5.5 Experimental Analysis of Non-Manifold Singularities Recognition 181
Chapter 8  The Manifold-Connected Decomposition 190

8.1 Design and Properties of the Manifold-Connected Decomposition 191
  8.1.1 Manifold-Connected components 191
  8.1.2 Design of the Manifold-Connected Decomposition 194

8.2 Computing the Manifold-Connected Decomposition 195
  8.2.1 A Dimension-Independent Algorithm 195
  8.2.2 Manifold-Connected Components and Non-manifold Singularities 196

8.3 Graph-based Representations 199
  8.3.1 The Exploded Manifold-Connected Graph 199
  8.3.2 The Pairwise Manifold-Connected Graph 201
  8.3.3 The Compact Manifold-Connected Graph 203

8.4 Experimental Comparisons 206

Chapter 9  The Manifold-Connected Decomposition and Homology Computation 214

9.1 Background Notions on Simplicial Homology 215

9.2 State of the Art 218

9.3 Constructive Homology Theory 220

9.4 Homological Smith Reduction 224

9.5 The Mayer-Vietoris Algorithm 227
  9.5.1 Computing the Homology of the Union of Two Simplicial Complexes 227
  9.5.2 The Mayer-Vietoris Iterative Algorithm 230

9.6 Experimental Results 232

Chapter 10  Concluding Remarks 236

Bibliography 239
Chapter 1

Introduction

Shape modeling is a wide research field, and is related to different methods for processing digital representations of objects for several application domains, such as computer-aided design and manufacturing, medical applications, finite elements, animation, and visualization, just to mention a few [Req80, Ede87, Man87, Hof89, Mun99, Ago05, DDFM06, TBG09, BKP10].

Existing modeling tools are designed to handle shapes with a manifold domain, since they are topologically simpler. Informally, a manifold shape is a subset of the Euclidean space, such that the neighborhood of each point is locally equivalent to an open ball. Shapes, which do not fulfill this property at one or more points, are called non-manifold, while they are non-regular if they also contain parts of different dimensions. Such shapes are also known as arbitrary shapes. Points, which do not satisfy the manifold property, are known as non-manifold singularities.

The simple structure of manifold shapes allows for the design of efficient and elegant solutions in several applications. Conversely, the structure of arbitrary shapes is more complex, and their analysis is not simple. Nevertheless, non-manifold objects arise in several applications, including in the context of biological and medical research [ABA06, LBD08, Ngu11].

The use of non-manifold shapes is motivated by several reasons. First, such representations provide more accurate representations of real objects, usually formed by several components of different dimension, arbitrarily connected. For instance, mechanical objects are the most common example of arbitrary shape. Specifically, the problem of representing and manipulating any arbitrary shape has been mainly studied in the context of the Finite Element Methods (FEM) [TCMT56, TBG09].

Furthermore, Boolean operators and simplification techniques are closed on non-manifold shapes, since they may generate parts of different dimension and non-manifold configurations in the input shape. Usually, these operators, restricted to manifolds, are not sufficient for any applications needs. For instance, physics-based simulations operate on CAD (Computer Aided Design) data, and their computational performance depends on the number of geometric features [WSO03]. Hence, in order to perform a mechanical simulation of a product, the CAD model needs to be
prepared for the FEM analysis. Furthermore, it is mandatory to take into account the mechanical hypotheses specified on the product shape, expressed as constraints to be satisfied, like the boundary conditions. This process, known as the *idealization*, involves several types of geometrical and topological operators [VL97, VL98, VL01, CDMM04, LF05, TBG09]. To guarantee acceptable computation times, not only details having a low impact on the object behavior are removed, but also portions of the shape are *idealized*. For instance, parts presenting a beam behavior are substituted with any 1-dimensional entities, and parts presenting a plate behavior are replaced by any 2-dimensional surfaces. Thus, the resulting object will contain non-manifold singularities and parts of different dimensions. In this way, it is possible to independently represent each portion of a shape.

### 1.1 Overview

In this thesis, we address the representation of non-manifold shapes, namely non-manifold subsets of any Euclidean space, consisting of pieces of different dimension. Modeling arbitrary shapes requires efficient representations for their discretized version and topological methods for the analysis of their structure. In this context, the objective of our research is twofold. On one side, we investigate new effective data structures for modeling arbitrary shapes discretized as simplicial (or cell) complexes. On the other hand, we consider a topological decomposition of non-manifold simplicial shape into manifold or almost manifold parts in order to deal with the intrinsic complexity of arbitrary shapes.

A non-manifold shape is discretized as simplicial or cell complexes. We exploit their connectivity information through *topological relations*. Topological relations provide an effective framework for several *topological data structures*, described in terms of the entities and relations they encode [DF03]. In this context, we have provided several contributions, by focusing our attention on topological data structures specific for non-manifold shapes of arbitrary dimension discretized through simplicial complexes. These data structures are not specific for Euclidean complexes, but actually are for abstract simplicial complex of any dimension [Kov89, Mun99, Ago05].

Following [DFH05], both dimension-specific and dimension-independent data structures have been developed for cell and simplicial complexes. The former typically exploit properties of the embedding domain to reduce storage requirements, while the latter do not depend on the embedding domain. The *Incidence Graph (IG) [Ede87]* is one of the most common dimension-independent *incidence-based* data structures for representing arbitrary shapes discretized by cell complexes, not necessarily embedded in a metric space. It encodes all the cells of the complex as well as a subset of the incidence relations among the cells. The IG data structure is verbose, and exhibits a very large overhead, if restricted to manifolds. Also, it does not allow detecting non-manifold singularities efficiently. Thus, we need more compact representations, from which it is efficient to detect non-manifold singularities. The *Simplified Incidence Graph (SIG) [DFGH04]* is a dimension-independent incidence-based variant of the IG data structure for representing simplicial complexes.
which overcomes several limitations of the IG data structure.

In this thesis, we have developed the *Incidence Simplicial (IS)* data structure [DFHPC10]. The IS data structure is a dimension-independent and incidence-based variant of the IG data structure for representing abstract simplicial complexes. The IS data structure encodes all the simplices in any arbitrary shape, and a subset of incidence relations among these simplices. As a consequence, it is suitable for those applications, like the FEM analysis and numerical simulations, where one needs to attach attributes to all simplices (or cells) in the complex. The IS data structure is more compact than the SIG data structure, and it exhibits a small overhead, if restricted to manifolds.

The incidence-based data structure offer a complete description of any simplicial complex, but we can get more compact representations if the specific application does not require attaching attributes to all simplices in the complex. Following [DFH05], the *adjacency-based* data structures have been shown to be suitable for these applications, since they encode only the top simplices, namely those simplices not on the boundary of other simplices, as well as the adjacency relations among such simplices. The *Indexed data structure with Adjacencies (IA)* [PBCF93, Nie97] and the *Extended Indexed data structure with Adjacencies (EIA)* [DF03] are the most compact adjacency-based representations for manifold shapes of any dimension, discretized by simplicial complexes. Thus, we need to extend these representations to arbitrary shapes. In the literature, there are two extensions of these data structures, namely the *Non-Manifold Indexed data structure with Adjacencies (NMIA)* [DFH03], and the *Triangle-Segment (TS)* [DFMPS04]. In any case, the TS and NMIA data structures represent only 2D and 3D simplicial complexes embedded in the Euclidean space $E^3$, respectively.

In order to overcome these limitations, we have designed the *Generalized Indexed Data Structure with Adjacencies (IA*) [CDFW11]. The IA* data structure is a dimension-independent adjacency-based data structure for simplicial complexes not necessarily embedded in any Euclidean space. Specifically, it encodes all the vertices and top simplices, plus a subset of the adjacency relations. In other words, it extends the EIA data structure to non-manifold simplicial shapes in arbitrary dimensions. The IA* data structure is more compact than all the incidence-based representations and dimension-specific ones, like the TS and NMIA representations. Also, it efficiently supports the retrieval of all the topological relations. To the best of our experience, the IA* data structure is currently the most compact data structure for representing arbitrary shapes of any dimension.

As discussed in [DFH05], several topological data structures have been introduced in the literature. Hence, a framework which supports a wide number of topological data structures under a common application interface is a valuable tool, especially from the applicative point of view. Actually, a topological data structure can be described through a graph-based representation, which we call a *mangrove*. We have exploited this representation in order to design the dimension-independent and extensible *Mangrove Topological Data Structure (Mangrove TDS)* framework, targeted to the fast prototyping of topological data structures. Also, this framework provides an implicit description for any simplex not directly encoded in a topological data structure, which we call *ghost simplex*. As a consequence, the Mangrove TDS framework supports a wide range of topological...
data structures. In order to prove the validity of our approach, we have designed and implemented several data structures, including the IS and IA* representations. We have provided navigation and construction algorithms for these topological data structures. Also, we have performed quantitative comparisons regarding their performances. The complete implementation of the Mangrove TDS framework, including all the data structures, is contained in the Mangrove TDS Library, which we plan to release in public domain for the community.

As mentioned above, it is important to decompose any arbitrary shape into manifold (or almost manifold) parts, in order to deal with its intrinsic complexity. In this context, a decomposition of an arbitrary shape should remove as many singularities as possible, and cuts it along singularities without breaking it at manifold parts. In this thesis, we have considered the Manifold-Connected Decomposition (MC-Decomposition). The basic concepts underlying this decomposition, but limited to 2D and 3D simplicial complexes, have been introduced in [HDF07a, HDF07b]. Here, any arbitrary simplicial shape is decomposed into nearly-manifold components, known as the Manifold-Connected components (MC-components). This decomposition is unique, and it is the discrete counterpart of the Whitney stratification [Whi65]. A suitable representation for the MC-Decomposition is a graph-based data structure, where a node describes a MC-component, while an arc describes the connectivity among several MC-components connected through a set of non-manifold singularities. Specifically, we have designed and implemented several two-level graph-based representations of the MC-Decomposition, which can be combined with any topological data structure, as discussed in [CDF11]. The upper level of these graph-based data structures consists of a collection of MC-components, while the lower level consists of a unique topological data structure. In this context, each MC-component is completely defined in terms of references to simplices in the input shape, as well as in any spatial index [Sam06]. These graph-based data structures are more expressive than a topological data structure, since they explicitly expose singularities and connectivity of MC-components. Also, we have combined each graph-based data structure with all the representations implemented in the Mangrove TDS Library, and we have performed extensive comparisons in terms of their storage cost and building times.

Recently, computing any topological invariants of a shape has drawn much attention, because they provide any information, which can be very useful when pure geometric tools are not sufficient. Simplicial homology provides the most common topological invariants [Mun99, Ago05]. Classical techniques for computing the simplicial homology exploit an algebraic approach [Ago76], which does not yield to an iterative algorithm [Ser94]. Conversely, the Constructive Homology Theory [Ser99, SR06] offers an elegant way for iteratively computing the homology of a shape from the homology of its subcomplexes and of their intersections. In this thesis, we have exploited this approach in order to design the Mayer-Vietoris (MV) algorithm [BCMA+11]. The MV algorithm retrieves the complete homological information from a decomposition of any arbitrary shape. Specifically, we have combined the MV algorithm with the MC-Decomposition of any arbitrary shape. In this case, the MV algorithm relates the simplicial homology of each MC-component, and of their intersections. In this case, the intersection among the MC-components is given by any non-manifold singularities, whose size is usually limited. We have experimentally shown that
the MC-Decomposition increases the efficiency of the MV algorithm, which results more efficient than the classical approaches.

1.2 Thesis Outline

The remainder of this thesis is organized as follows.

In Chapter 2, we review some background notions, that we use throughout this thesis. This chapter is largely inspired by [DF03, Mor03]. Specifically, we briefly review some concepts on cell and simplicial complexes. We also review a characterization of topological manifolds, and their combinatorial versions [Mor03]. Finally, we discuss a theoretical framework for representing the connectivity information provided by cell and simplicial complexes through the topological data structures [DF03]. Note that, in Section 9.1, we also review the basic notions about the simplicial homology.

In Chapter 3, we discuss the state of the art in the fields of our research. Specifically, we propose a brief review and classification of several topological data structures, by extending [DFH05]. One of the contributions of our thesis is the Manifold-Connected Decomposition (MC-Decomposition), thus, we also discuss several decomposition approaches in the literature. Also, we briefly review several editing operators for cell and simplicial complexes, and specific for the idealization process. Note that, in Section 9.2, we review the state of the art regarding the computation of simplicial homology.

In Chapter 4, we present the Incidence Simplicial (IS) data structure [DFHPC10]. Here, we also review the IG and SIG data structures, and compare these representations with the IS data structure in terms of their storage cost. Our tests show that the IS data structure is more compact than the Incidence Graph. Also, the IS data structure is as compact as the SIG data structure for arbitrary simplicial 2-complexes, and more compact than the SIG representation for arbitrary simplicial 3-complexes.

In Chapter 5, we present the Generalized Indexed data structure with Adjacencies (IA∗), introduced in [CDFW11]. Here, we also propose slight variants of the TS and NMIA data structures, where we exploit an efficient encoding for the non-manifold adjacency along a simplex, which efficiently characterizes a non-manifold singularity. We compare these representations with the IA∗ data structure, restricted to 2D and 3D simplicial complexes, in terms of storage costs. Our tests show that the IA∗ data structure is more compact than the TS and NMIA representations.

In Chapter 6, we introduce our Mangrove Topological Data Structure (Mangrove TDS) framework. We highlight the main differences of our framework with respect to the existing frameworks in the literature. In order to prove the validity of our approach, we describe the complete implementations of the IS and IA∗ data structures in our framework. Specifically, we provide pseudo-code of all the topological queries, and of the construction algorithms from a soup of top simplices.
In Chapter 7, we present comparisons about performances of all the topological data structures discussed in Chapter 4 and Chapter 5. For each data structure, we provide the pseudo-code of topological queries and construction algorithms from a soup of top simplices. Our tests show that the IS and IA∗ data structures offer an optimal compromise regarding their expressive power, storage cost, and efficiency of all the queries. Finally, we show that the ghost simplices are an effective tool for extending the expressive power of several data structures.

In Chapter 8, we describe the Manifold-Connected Decomposition (MC-Decomposition), initially introduced in [HDF07a, HDF07b], and we review several properties of MC-components. Also, we propose a complete definition of our two-level graph-based data structures for representing the MC-Decomposition [CDF11]. Finally, we provide our experimental comparisons regarding the storage cost and building time for each of these graph-based representations, combined with all the data structures implemented in the Mangrove TDS Library.

In Chapter 9, we describe our Mayer-Vietoris (MV) algorithm [BCMA+11], which computes the complete homological information of a non-manifold shape. We also review here background notions and the state of the art related to homology computations, including basic notions of the constructive homology theory. We describe our implementation of the MV algorithm, and provide our experimental comparisons. Our tests show that several properties of the MC-Decomposition increase the efficiency of our MV-algorithm.

In Chapter 10, we briefly summarize the results of our research, and outline future developments and open problems.
Chapter 2

Background Notions

In this chapter, we review some background notions, that we will use throughout this thesis.

In our research, we consider shapes with an arbitrary topology, namely the arbitrary shapes. Moreover, although such shapes exist in the continuum, we need discrete models in order to represent and manipulate them computationally. Simple shapes admit a discrete analytic description based on a large number of parameters as, for instance, the fractals modeling [Man77, MM91, Mus93]. Although analytic representations may easily represent arbitrary shapes, they are not flexible and easy to use, due to the large number of parameters. Hence, we can exploit another type of representation, where shapes are described in terms of elementary and mutually connected pieces.

In Section 2.1, we review cell complexes, which are the most common discrete representations of a shape. The notion of cell complex is quite general, but cell-based representations may be quite verbose. In Section 2.2, we restrict our attention to a special subclass of cell complexes, namely the so-called simplicial complexes. In our research, we are interested in shapes, which may contain several points with a neighborhood not necessarily homeomorphic to a ball, namely the non-manifold singularities. These shapes are usually known as arbitrary shapes. In Section 2.3, we characterize any non-manifold singularities in the context of cell and simplicial complexes. In order to design an effective representation of a shape, we exploit the connectivity information provided by cell and simplicial complexes. In Section 2.4, we discuss a theoretical framework for representing the connectivity information provided by cell and simplicial complexes in terms of topological relations [DF03].

Most of topics discussed in this chapter are largely inspired by [DF03, Mor03]. In this chapter, we assume that some theoretical notions of point set and algebraic topology are known: an interested reader can refer to [Man87, Hof89, Mun99, Ago05] for more details.
2.1 Cell Complexes

In this section, we review the notion of cell complex, and we restrict our attention to the Euclidean case. The content of this section is largely inspired by [DF03]. An interested reader can refer to [Man87, Hof89, Mun99, Ago05] for more details. Intuitively, a Euclidean cell complex is a collection of basic elements, known as cells, which cover a domain in the Euclidean space [Man87].

In the Euclidean space $\mathbb{E}^n$, the standard open unit $d$-ball is the set $B^d = \{ x \in \mathbb{E}^d : \|x\| < 1 \}$, where $\|x\|$ is the standard Euclidean metric in $\mathbb{E}^d$. Similarly, the standard closed unit $d$-ball is the set $B^d = \{ x \in \mathbb{E}^d : \|x\| \leq 1 \}$, while the standard closed unit half $d$-ball is the set $B^d_+ = \{ x \in \mathbb{E}^d : x > 0 \land \|x\| \leq 1 \}$. Also, the standard unit $d$-sphere is the set $S^d = \{ x \in \mathbb{E}^{d+1} : \|x\| = 1 \}$.

A $k$-dimensional cell, namely a $k$-cell, in the Euclidean space $\mathbb{E}^n$, with $1 \leq k \leq n$, is a subset of $\mathbb{E}^n$ homeomorphic to $B^k$. A 0-cell is defined as a point in $\mathbb{E}^n$. The value of $k$ is called the dimension of a $k$-cell $\gamma$, and it is often denoted as $k = \text{dim}(\gamma)$. Note that a 0-cell is a vertex, a 1-cell is an edge, a 2-cell is a face, and a 3-cell is a volume.

The relative boundary $b(\gamma)$ of a $k$-cell $\gamma$ in the Euclidean space $\mathbb{E}^n$, with $1 \leq k \leq n$, is the boundary of $\gamma$ with respect to the topology of the metric space $\mathbb{E}^n$. Note that the relative boundary of a 0-cell is empty. The combinatorial boundary $B(\gamma)$ of a cell $\gamma$ is the collection of cells $\gamma'$ such that $\gamma' \subseteq b(\gamma)$, as point set. Any cell $\gamma'$ in $B(\gamma)$ bounds the cell $\gamma$.

A Euclidean cell complex $\Gamma$ in $\mathbb{E}^n$ is a collection of cells of dimension at most $d$, with $0 \leq d \leq n$, such that the cells of $\Gamma$ are disjoint, and the combinatorial boundary of each $k$-dimensional cell, with $0 \leq k \leq d$, consists of cells in $\Gamma$ of dimension less than $k$. The maximum dimension $d$ of cells in $\Gamma$ is called the dimension of a cell complex $\Gamma$, and it is usually denoted as $d = \text{dim}(\Gamma)$. Thus, $\Gamma$ is often called a Euclidean cell $d$-complex. We denote, for $0 \leq k \leq d$, all the $k$-cells in $\Gamma$ as $\Gamma^k$, and their number as $s_k$. We denote the total number of cells in $\Gamma$ as $S_\Gamma$. The domain $\Delta(\Gamma)$ of $\Gamma$ is the subset of the Euclidean space $\mathbb{E}^n$ spanned by all the cells in $\Gamma$. Since we have restricted our attention to the Euclidean case, the domain $\Delta(\Gamma)$ of any Euclidean cell complex $\Gamma$ is said to be a geometric polyhedron.

Any $d$-cell in a Euclidean cell $d$-complex $\Gamma$ is a maximal cell. A cell which does not belong to the boundary of any other cell in $\Gamma$ is said to be a top cell. We denote all the top $k$-cells in $\Gamma$, with $0 \leq k \leq d$, as $\Gamma^k_t$, and their number as $s^k_t$. Note that $s_d = s^d_t$. We denote the total number of top cells in $\Gamma$ as $S^t_\Gamma$. A Euclidean cell $d$-complex $\Gamma$ such that all the top cells are maximal cells is known as a regular cell $d$-complex. Here, all the sets of top simplices are empty, except $\Gamma^d_t \equiv \Gamma^d$.

An $h$-cell $\gamma'$ belonging to the combinatorial boundary $B(\gamma)$ of a $k$-cell $\gamma$, with $h \leq k$, is called a $h$-face of $\gamma$. If $\gamma' \neq \gamma$, then $\gamma'$ is a proper $h$-face of $\gamma$. Note that $\gamma$ is a face of itself. Cells $\gamma$ and $\gamma'$ are known as incident cells. Two Euclidean $j$-cells are said to be $k$-adjacent, with $0 \leq k \leq j$, if they share a $k$-cell. In particular, two Euclidean $k$-cells, with $k \neq 0$, are said to be adjacent if they are $(k-1)$-adjacent. Two vertices are adjacent if they are both incident at a common edge.

Given a Euclidean cell $d$-complex $\Gamma$, an $h$-path, with $h < d$, is a sequence of $(h+1)$-cells $(\gamma_i)_{i=0}^n$ in
Γ such that two consecutive cells γ_{i-1} and γ_i in such a sequence are h-adjacent. Two cells γ and γ* in Γ are said to be h-connected when there exists a h-path (γ_i)_i=0^n in Γ such that γ is a face of γ₀ and γ* is a face of γₙ. A Euclidean cell complex Γ is h-connected if all the pairs of cells in Γ are h-connected.

A subset Γ' of a Euclidean cell d-complex Γ is said to be a subcomplex of Γ if and only if Γ' is a Euclidean cell complex. Any maximal h-connected subcomplex Γ' of a Euclidean cell complex Γ is called h-connected component of Γ. Any (h - 1)-connected component of Γ, with 2 ≤ h ≤ d, formed by any top h-simplices in Γ is said to be an h-cluster of Γ. A top 1-cell in Γ is a 1-cluster of Γ. The k-skeleton of a Euclidean cell d-complex Γ, with 0 ≤ k ≤ d, is the subcomplex of Γ, which consists of all the cells in Γ of dimension less than or equal to k. Given a Euclidean cell d-complex Γ, the closure Ψ of a subset Ψ of Γ is the smallest subcomplex of Γ containing Ψ. The closure of Ψ consists of all the cells of Ψ together with all their faces.

The star (or combinatorial co-boundary) St(γ) of a k-cell γ in a Euclidean cell complex Γ is the set of all the h-cells of Γ, with h > k, which contain γ in their combinatorial boundary. Formally, St(γ) = {γ' ∈ Γ | γ ∈ B(γ')} We denote the number of top h-cells in St(γ) as γₖ⁺. Also, the total number of top cells in St(γ) is denoted as γ⁺ₖ, while the total number of cells in St(γ) is denoted as ||St(γ)||. The link Lk(γ) of a k-cell γ in a Euclidean cell complex Γ is formed by all the cells in Γ, which belong to the combinatorial boundary of the cells in St(γ), and are not incident at γ.

The valence of a k-cell γ in a Euclidean cell complex Γ is the number of (k + 1)-cells of Γ in St(γ), and it is denoted as val(γ). The degree of a k-cell γ in a Euclidean cell complex Γ is the number of (k - 1)-cells of Γ in B(γ), and it is usually denoted as deg(γ).

Given a Euclidean cell complex Γ, a decomposition of Γ is a non-empty collection of distinct subcomplexes in Γ, namely (Γᵢ)ᵢ=₀^ₙ, such that:

i) the union of ∆(Γᵢ), ∀i = ₀,...,ₙ, covers ∆(Γ);

ii) for each pair of subcomplexes Γᵢ and Γⱼ, either Γᵢ ∩ Γⱼ = ∅, or Γᵢ ∩ Γⱼ = {γ ∈ Γ | γ ∈ Γᵢ ∧ γ ∈ Γⱼ}.

This means that the intersection of two cells is either empty, or formed by common cells in Γ.

A decomposition of a Euclidean cell Γ is usually defined by application-dependent principles, which introduce several collections of subcomplexes of Γ. A decomposition of any Euclidean cell complex Γ is optimal if it minimizes the number of subcomplexes. In [Whi65] the author demonstrates that the optimal decomposition of any cell complex Γ always exists. In any case, finding the optimal decomposition of Γ is a NP-complete problem [CDST97].

Following [DFPM97], an update u = (u⁻, u⁺) of a Euclidean cell Γ replaces a set u⁻ of cells in Γ with another set of cells u⁺. The result of applying u to Γ is denoted with Γ[u]. Note that Γ[u] contains all cells in {Γ/u⁻} u u⁺, and may not be a cell complex. We are interested in the case where Γ[u] is a cell complex. Thus, we restrict our attention to valid updates, defined as
follows. An update $u = (u^-, u^+)$ is valid if the boundaries of $u^-$ and $u^+$ perfectly match, and the cell complex $\Gamma/u^-$ does not intersect $u^+$ outside its boundary. It can be proven that the result $\Gamma[u]$ of applying a valid update to a cell $k$-complex $\Gamma$ is either empty or a cell $k$-complex [Mag99]. Figure 2.1 shows several updates applied on a cell 2-complex, where we replace a set $u^-$ of triangles depicted in yellow. Update proposed in Figure 2.1(b) is valid. Update in Figure 2.1(c) is not valid, since it introduces a new vertex (in red), which does not belong to boundary of $u^-$. A valid update is a refinement if the number of cells in $u^+$ is greater than the number of cells in $u^-$. A decimation update is the inverse update of the refinement one.

A valid update is a refinement if the number of cells in $u^+$ is greater than the number of cells in $u^-$. A decimation update is the inverse update of the refinement one.

In the remainder of this document, we denote a Euclidean cell complex simply as cell complex, if no ambiguity may arise.

The definition of cell complex we give in this section is not powerful enough to capture all the characteristics of real objects. In some approaches proposed in the literature for modeling boundary representations, namely surfaces bounding a solid object in $E^3$, this drawback is overcome by allowing for multiply-connected faces. These faces have zero genus, and are bounded by several connected components of edges, usually known as loops, or rings. Conversely, volumes can have through holes or cavities. Basically, in these cell complexes, cells are not necessarily homeomorphic to a ball. Such cell complexes are known as general cell complexes. Representations for cell complexes described in Section 3.2, like the Radial Edge (RE) [Wei88b] and the Partial-Entities (PE) [LL01] data structures, can describe general cell 2-complexes.

### 2.2 Simplicial Complexes

In this section, we review simplicial complexes, which can be seen as special cell complexes. We restrict our attention to the Euclidean case. The content of this section is largely inspired by [DF03].
An interested reader can refer to [Man87, Hof89, Mun99, Ago05] for more details. Intuitively, simplicial complexes can be seen as cell complexes, where their cells, known as simplices, are closed and defined as the convex combination of points in the Euclidean space.

Let $k$ be a non-negative integer and let $V_\sigma$ be a set of $k + 1$ points in $\mathbb{E}^n$, then a Euclidean $k$-simplex $\sigma$ is the convex hull of the elements of $V_\sigma$, known as vertices of $\sigma$. The value of $k$ is known as the dimension of a $k$-simplex $\sigma$, and it is usually denoted as $k = \dim(\gamma)$. In this case, $\deg(\sigma) = k + 1$. Note that a 0-simplex is a vertex, a 1-simplex is an edge, a 2-simplex is a triangle, and a 3-simplex is a tetrahedron. Given a set of vertices $V_\sigma' \subseteq V_\sigma$ formed by $h + 1$ vertices, with $h \leq k$, the Euclidean $h$-simplex $\sigma'$ generated by $V_\sigma'$ is an $h$-face of $\sigma$. If $h \neq k$, then $\sigma'$ is a proper $h$-face of $\sigma$. Conversely, $\sigma$ is said to be a co-face of $\sigma'$. The number of $h$-faces of $\sigma$, with $h \leq k$, is equal to $\binom{k+1}{h+1}$, usually denoted as $s^k_h(\sigma)$ [Ede87].

A finite collection $\Sigma$ of Euclidean simplices forms a Euclidean simplicial complex if:

i) given a Euclidean simplex $\sigma$ in $\Sigma$, all the faces of $\sigma$ belong to $\Sigma$;

ii) given a pair of simplices $\sigma'$ and $\sigma''$ in $\Sigma$, either $\sigma' \cap \sigma'' = \emptyset$, or $\sigma' \cap \sigma'' \in \Sigma$.

This means that the intersection of any two simplices $\sigma'$ and $\sigma''$ in $\Sigma$ is either empty, or a common face in $\Sigma$.

The maximum dimension $d$ of any simplex in $\Sigma$ is called the dimension of a $\Sigma$, and it is usually denoted as $d = \dim(\Sigma)$. Thus, $\Sigma$ is called a Euclidean simplicial $d$-complex. The domain $\Delta(\Sigma)$ of a Euclidean simplicial $d$-complex $\Sigma$ embedded in $\mathbb{E}^n$, with $d \leq n$, is the subset of $\mathbb{E}^n$ formed by the union, as point sets, of all the Euclidean simplices in $\Sigma$. Also in this case, since we have restricted our attention to the Euclidean case, the domain $\Delta(\Sigma)$ of any Euclidean simplicial complex $\Sigma$ is said to be a geometric polyhedron, usually known as a polyhedron.

A Euclidean simplicial complex in the Euclidean space $\mathbb{E}^n$ is a topological space, and its topology is completely inherited from the one of $\mathbb{E}^n$. Since a simplicial complex is also a cell complex, all properties of cell complexes are inherited by simplicial complexes.

In the remainder of this thesis, we denote a Euclidean simplicial complex simply as simplicial complex, if no ambiguity may arise.

### 2.3 Characterization of Non-manifold Singularities

In this section, we provide a characterization of non-manifold singularities of any shape, namely points with a neighborhood not necessarily smooth. The key idea of this approach, proposed in [DFMMP03], is to give a characterization of topological manifolds, and then to adapt this definition to the combinatorial properties of cell complexes. The content of this section is largely inspired by [DFMMP03, Mor03].
A topological $d$-manifold in the Euclidean space $\mathbb{E}^d$ is a connected subset $M$ of $\mathbb{E}^d$, where each point of $M$ has a neighborhood homeomorphic to the standard open unit $d$-ball $B^d$. A topological $d$-manifold with boundary is a connected subset $M$ of the Euclidean space $\mathbb{E}^d$, where each point of $M$ has a neighborhood homeomorphic either to the standard opened unit $d$-ball $B^d$, or the standard closed unit half $d$-ball $B^d_+$. Any connected subset $M$ of $\mathbb{E}^d$ which do not fulfill this property at one or more points is said to be non-manifold. Usually, a shape described by any non-manifold domain with pieces of different dimensions is called an arbitrary shape. Figure 2.2(a) shows a topological 2-manifold, namely a torus embedded in $\mathbb{E}^3$. Conversely, Figure 2.2(b) shows a non-manifold shape in $\mathbb{E}^3$, which contains several non-manifold vertices.

A cell $d$-complex $\Gamma$ having $s^k_\Gamma$ top $k$-cells, with $k \neq d$ and $s^k_\Gamma \neq 0$, discretizes an arbitrary shape. Here, non-manifold singularities are located along connections among parts of different dimensions, but not only. In any case, we start our analysis from regular cell complexes.

A regular $(d-1)$-connected cell $d$-complex $\Gamma$, in which the star of any $(d-1)$-cell $\gamma$ consists of just one or at most two $d$-cells, is said to be a combinatorial pseudo-manifold $d$-complex, possibly with boundary. Figure 2.2(b) shows a combinatorial pseudo-manifold 2-complex. In Section 8.1, we review several properties of pseudo-manifold complexes, which have been demonstrated in [HDF07a].

In this context, manifold cell complexes are defined through the combinatorial equivalence between two cell complexes. Two cell complexes $\Gamma'$ and $\Gamma''$ are combinatorially equivalent if their carriers $\Delta(\Gamma')$ and $\Delta(\Gamma'')$ are homeomorphic. Basically, we should provide a combinatorial characterization of a topological manifold. A vertex, namely a 0-cell, is homeomorphic to the 0-ball $B^0$, by definition. Also, we define a combinatorial $d$-ball $B^d_\Gamma$ as a cell $d$-complex such that $\Delta(B^d_\Gamma)$ is homeomorphic to $B^d$. Similarly, we define a combinatorial $d$-sphere $S^{d-1}_\Gamma$ as the boundary of $B^{d+1}_\Gamma$, considered as point set. Hence, it quite clear that the neighborhood of a point in a topological manifold must be homeomorphic either to $\Delta(B^d_\Gamma)$ or $\Delta(S^{d-1}_\Gamma)$.

This information is analyzed through link $Lk(\gamma)$ of any vertex $\gamma$. Given a regular cell $d$-complex $\Gamma$, a vertex $\gamma$ in $\Gamma$ is manifold if and only if link $Lk(\gamma)$ is combinatorially equivalent either to $B^d_\Gamma$ or to $S^{d-1}_\Gamma$. If all vertices in $\Gamma$ are manifold, then $\Gamma$ is a combinatorial manifold $d$-complex. Figure 2.2(b) shows several non-manifold vertices in a pseudo-manifold 2-complex.

Given a regular cell $d$-complex $\Gamma$, any $(d-1)$-cell $\gamma$ is manifold if just one or two $d$-cells are incident at $\gamma$. Otherwise, $\gamma$ is called a non-manifold cell. Figure 2.2(c) shows a regular cell 2-complex, where three triangles are incident at a non-manifold edge $e$.

As demonstrated in [Hud69], combinatorial manifold $d$-complexes are topological $d$-manifolds. However, in [FQ90] the authors prove that there are topological 4-manifolds, which are not combinatorial manifold 4-complexes. Hence, topological manifolds are a proper superclass of combinatorial manifold complexes.

However, the combinatorial equivalence, used in the definition of combinatorial manifold complexes, is less "powerful" than the topological equivalence. In [Mil61] the author proves the existence of two 7-complexes, which are topologically equivalent, but not combinatorially equivalent.
Figure 2.2: Cell 2-complexes embedded in the Euclidean space $\mathbb{E}^3$. (a) A topological 2-manifold, namely a torus embedded in $\mathbb{E}^3$. (b) A combinatorial pseudo-manifold 2-complex. Here, several vertices are non-manifold, since their links are not combinatorially equivalent to $B^2_\Gamma$ or to $S^1_\Gamma$. (c) A non-manifold regular cell 2-complex, which contains a non-manifold edge $e$, shared by three triangles. Figures courtesy of [DFPH09].

In particular, it is interesting to analyze the problem of recognizing whether or not a $(d+1)$-complex is a combinatorial manifold $(d+1)$-complex. This problem involves the recognition of the $d$-sphere $S^d$, which implies, for any $d \geq 5$, the recognition of a trivial group in a finite sequence of homological groups [Mar58, KVF74]. This latter implies the Halting problem [Adj93, Nab96], which is not decidable [Tur36]. Hence, the recognition of the $d$-sphere $S^d$ is not decidable for any $d \geq 5$ [Nab96], while, at the moment, it is an open problem for $d = 4$. As a consequence, the problem of recognizing whether or not a $(d+1)$-complex is a combinatorial manifold $(d+1)$-complex is decidable for $d < 4$, is an open problem for $d = 4$, and is not decidable for $d \geq 5$.

In [Mor03] the author extends the characterization of manifold to any $k$-cell in arbitrary cell $d$-complexes, with $k < d$. Given a cell $d$-complex $\Gamma$, a $k$-cell $\gamma$, with $k < d$, is a manifold $k$-cell if and only if link $Lk(\gamma)$ is combinatorially equivalent either to $B^h_\Gamma$ or to $S^{h-1}_\Gamma$, for some $h \leq d - k - 1$. Otherwise, the $k$-cell $\gamma$ is called a non-manifold cell. We denote the number of non-manifold $k$-cells in $\Gamma$ as $s^k_\Gamma$. We denote the total number of non-manifold cells in $\Gamma$ as $S^n_\Gamma$. We denote the set of non-manifold cells in $\Gamma$ as $\Gamma_n$. If all the $k$-cells in $\Gamma$, for $0 \leq k < d$, are manifold, then $\Gamma$ is a combinatorial manifold $d$-complex.

In the following, we restrict our attention only to combinatorial manifold complexes. Also, we will omit the term "combinatorial", and talk about manifoldness to mean "combinatorial manifoldness".

### 2.4 Topological Relations on Cell Complexes

Cell and simplicial complexes, respectively discussed in Section 2.1 and Section 2.2, provide a discrete representation of a shape in terms of cells and simplices. In this section, we exploit connectivity information among entities in a cell complex, as discussed in [DF03]. Such information is formally described by the concept of topological relation between two cells. Topological relations provide an effective framework for defining, analyzing, and comparing a wide spectrum of topological data structures. These latter are formally described in terms of the topological entities.
and relations they encode. In this section, we analyze only topological relations for a cell complex, since simplicial complexes are a special case of cell complexes, following the same approach proposed in [DF03].

Let $\Gamma$ be a cell $d$-complex and $\Gamma^j$ be the collection of $j$-cells in $\Gamma$, with $0 \leq j \leq d$, then we can define $(d + 1)^2$ ordered topological relations by considering all the possible pairs $(\Gamma^k, \Gamma^m)$, with $0 \leq k, m \leq d$. A topological relation for a pair $(\gamma, \gamma')$ in $\Gamma^k \times \Gamma^m$, denoted by $\sim_{km}$, is defined as follows:

- **boundary relation** $\gamma \sim_{km} \gamma'$, with $0 < m < k \leq d$, if $\gamma'$ is a $m$-face of a $k$-cell $\gamma$;
- **co-boundary relation** $\gamma \sim_{km} \gamma'$, with $0 \leq k < m \leq d$, if the $m$-cell $\gamma'$ belongs to $St(\gamma)$;
- **adjacency relation** $\gamma \sim_{kk} \gamma'$, with $0 < k \leq d$, if the $k$-cells $\gamma'$ and $\gamma$ are $k$-adjacent;
- **adjacency relation** $\gamma \sim_{00} \gamma'$, if the two vertices $\gamma$ and $\gamma'$ are adjacent in $\Gamma$.

Boundary and co-boundary relations are known as incidence relations.

For each relation $\sim_{km}$, we define the corresponding relational operator $R_{k,m} : \Gamma^k \to P(\Gamma^m)$ such that, given a $k$-cell $\gamma$ in $\Gamma$, $R_{k,m}(\gamma) = \{\gamma' \in \Gamma^m. \gamma \sim_{km} \gamma'\}$. For the sake of simplicity, we confuse a topological relation $\sim_{km}$ with its relational operator $R_{k,m}$. In particular, given a relation $R_{k,m}$, we denote the total number of $m$-cells related to all the $k$-cells in $\Gamma$ through $R_{k,m}$ as $\|R_{k,m}\|$. Also, we denote the maximum number of $m$-cells related to one $k$-cell in $\Gamma$ through the relation $R_{k,m}$ as $M_{k,m}$.

Figure 2.3 provides several topological relations in a cell 2-complex. Figure 2.3(a) shows, for a face $f$, all the edges and vertices in the boundary relations $R_{2,1}(f) = \{e_1, e_2, e_3, e_4, e_5\}$ and $R_{2,0}(f) = \{v_1, v_2, v_3, v_4, v_5\}$. Figure 2.3(b) shows, for a vertex $v$, all the edges and faces in the co-boundary relations $R_{0,1}(v) = \{e_6, e_7, e_8, e_9, e_{10}\}$ and $R_{0,2}(v) = \{f_1, f_2, f_3, f_4\}$. Figure 2.3(c) shows all the vertices in the adjacency relation $R_{0,0}(v) = \{v_1, v_2, v_3, v_4, v_5\}$ for a vertex $v$, and all the faces in the adjacency relation $R_{2,2}(f_1) = \{f_2, f_5\}$, for a face $f_1$.

![Figure 2.3: Examples of topological relations in a cell 2-complex. (a) All the edges and vertices in the boundary relations $R_{2,1}(f)$ and $R_{2,0}(f)$ for a face $f$. (b) All the edges and faces in the co-boundary relations $R_{0,1}(v)$ and $R_{0,2}(v)$ for a vertex $v$. (c) All the vertices in the adjacency relation $R_{0,0}(v)$ for a vertex $v$, and faces $f_2$ and $f_5$ in the adjacency relation $R_{2,2}(f_1)$.](image-url)
We call *constant* any relation involving a constant number of cells, while relations involving a variable number of cells are known as *variable*. In a cell complex, topological relations are variable, since a $k$-cell contains an arbitrary number of vertices. Instead, in a simplicial complex, co-boundary and adjacency relations are variable, while boundary relations are constant, since a $k$-simplex $\sigma$ is defined by $k + 1$ vertices, and $\text{deg}(\sigma) = k + 1$. 

Chapter 3

State of the Art

In this chapter, we briefly review the state of the art in fields of our research. Most of this chapter is devoted to reviewing data structures developed in the literature for representing surfaces and volumetric shapes. Our aim is to review and classify topological data structures, extending [DFH05]. Most of topics discussed in this chapter are largely based on [DFH05, TBG09, CDF12].

As discussed in Section 2.4, connectivity information of cell complexes is exploited through topological relations. They provide an effective framework for several topological data structures, described in terms of the entities and relations they encode [DF03].

Following [DFH05], topological data structures can be briefly classified in terms of:

- **dimension** of cell complexes: we recognize dimension-independent and dimension-specific data structures. A dimension-independent data structure describes cell complexes of any dimension, while a dimension-specific data structure encodes only cell complexes of a given dimension;

- **domain** to be approximated: there are topological data structures for any domain, i.e., manifolds, regular and non-manifold shapes, non-regular and non-manifold shapes, and pseudo-manifolds;

- **amount** of topological information, which is directly encoded: there exist data structures, which encode all the cells in a cell complex, or only top cells and vertices;

- **organization** of topological information directly encoded: there exist explicit and implicit data structures. In explicit data structures, cells are explicitly represented through topological entities. Implicit data structures indirectly encode topological relations through tuples of cells in the same relation.

Explicit data structures are further classified into incidence-based and adjacency-based representations. Incidence-based data structures encode all the cells in a cell complex, plus a suitable subset
of incidence relations. Adjacency-based data structures encode only top cells and vertices, plus a suitable subset of adjacency and co-boundary relations.

Data structures, which are designed for cell complexes, are also used for simplicial complexes. In some cases, specializations of these data structures have been developed by taking advantage of properties of simplicial complexes.

An interesting property of a topological data structure describing an arbitrary shape is scalability to manifolds. It is defined as overhead required by a data structure for arbitrary shapes, when it is used for representing a manifold shape. This property is very interesting, because, in a typical modeling scenario, it may be mandatory to manage representations capable to deal with both manifold and non-manifold shapes.

In Section 3.1, we discuss several dimension-independent representations for cell and simplicial complexes of arbitrary dimension. In Section 3.2, we discuss several dimension-specific representations of surfaces discretized through 2-complexes. Similarly, in Section 3.3, we discuss several dimension-specific representations of volumetric shapes discretized through 3-complexes.

One of our contributions is to design efficient representations for the Manifold-Connected Decomposition (MC-Decomposition) [HDF07a, HDF07b]. In Section 3.4, we briefly review several decomposition-based approaches, which provide a decomposition of arbitrary shapes into simpler components.

Finally, in Section 3.5, we propose a brief analysis of several editing operators, which modify topological and combinatorial information provided by a shape representation.

### 3.1 Dimension-independent Data Structures

In this section, we briefly discuss several dimension-independent data structures for cell and simplicial complexes. Most of this section is largely inspired by [DFH05].

In Section 3.1.1, we review a dimension-independent and implicit representation for manifolds, discretized by cell complexes, namely the Cell-Tuple [Bri89] data structure. In Section 3.1.2, we briefly introduce the Incidence Graph (IG) [Ede87], an explicit incidence-based representation for arbitrary shapes discretized by cell complexes. Finally, in Section 3.1.3, we discuss the Extended Indexed data structure with Adjacencies (EIA) [DF03], an explicit adjacency-based representation for pseudo-manifolds of arbitrary dimension discretized by simplicial complexes.

#### 3.1.1 The Cell-Tuple Data Structure

In this section, we briefly review a dimension-independent and implicit representation for manifold shapes discretized by cell complexes, namely the Cell-Tuple data structure [Bri89], which is equivalent to the N-G-Map data structure [Lie94]. The Cell-Tuple data structure is a represen-
tation for Euclidean cell complexes with a manifold domain, while the N-G-Map data structure represents abstract cell complexes belonging to a superclass of combinatorial manifolds. For the sake of brevity, we describe only the Cell-Tuple data structure, following [DFH05].

Given a Euclidean $d$-dimensional cell complex, a cell tuple is a $(d+1)$-tuple $t = (c_0, c_1, \ldots, c_d)$, such that $c_i$ is an $i$-cell, which bounds all the cells from $c_{i+1}$ to $c_d$. A switch function $\delta_i$, for $i = 0, \ldots, d$, is defined on all the cell-tuples $t' = \delta_i(t)$, such that cell-tuple $t'$ differs from $t$ only in the element in position $i$. Switch functions subdivide a set of cell-tuples into equivalence classes of size 2, and satisfy two properties:

- $\delta_i$ is an involution for $i = 0, \ldots, d$, i.e., given a cell-tuple $t$, $\delta_i(\delta_i(t)) = t$;
- given a cell-tuple $t$, $\delta_i\delta_j(t) = \delta_j\delta_i(t)$ is an involution for $i = 0, \ldots, d - 2$ and $i + 2 \leq j \leq d$, i.e., $\delta_i\delta_j(\delta_i\delta_j(t)) = t$.

Figure 3.1 provides cell-tuples and switch functions encoded by the Cell-Tuple data structure for a cell 2-complex formed by a triangle $A$, a square $B$, and an external 2-cell $C$. Here, two tuples are related by a switch function $\delta_0$ if they are connected through a dotted line. Similarly, they are related by a switch function $\delta_1$ if they are connected by a thin solid line, or by a switch function $\delta_2$ if connected by a dashed line.

Figure 3.1: Cell-tuples and switch functions encoded by the Cell-Tuple data structure for a cell 2-complex formed by a triangle $A$, a square $B$, and an external 2-cell $C$. Figure courtesy of [DFH05].

In a cell $d$-complex $\Gamma$, the Cell-Tuple data structure encodes all the cell-tuples and switch functions $\delta_i$ in $\Gamma$, for $i = 0 \ldots d$. Here, cells and their mutual topological relations are implicitly represented by cell-tuples. In any case, topological relations are optimal in the Cell-Tuple data structure [DFMMP03].

In [CMP06] the authors generalize the N-G-Maps data structure for non-manifold cellular shapes, known as the Cell-Chains data structure. A similar approach is exploited in [CK10], in order to define the Extended Maps (X-Maps) data structure, which is briefly discussed in Section 3.3.3. The X-Maps data structure represents arbitrary cell-complexes of dimension up to 3.
3.1.2 The Incidence Graph

In this section, we briefly introduce the Incidence Graph (IG) [Ede87], an explicit incidence-based representation for arbitrary shapes discretized by cell complexes.

Following [DFH05], the IG data structure encodes all the cells in a cell d-complex \( \Gamma \), and, for each \( p \)-cell \( \gamma \) in \( \Gamma \), boundary relation \( \mathcal{R}_{p,p-1}(\gamma) \), with \( 0 < p \leq d \), and co-boundary relation \( \mathcal{R}_{p,p+1}(\gamma) \), with \( 0 \leq p < d \).

The IG data structure supports a recursive strategy to retrieve topological relations. Any boundary relation \( \mathcal{R}_{p,q}(\gamma) \), with \( p > q \), for a given \( p \)-cell \( \gamma \), is obtained by retrieving encoded boundary relations \( \mathcal{R}_{i,i-1} \) for all the \( i \)-faces of \( \gamma \), with \( q < i \leq p \). Any co-boundary relation \( \mathcal{R}_{p,r}(\gamma) \), with \( p < r \), for a given \( p \)-cell \( \gamma \), is obtained by retrieving encoded co-boundary relations \( \mathcal{R}_{i,i+1} \) for all the \( i \)-cells in \( St(\gamma) \), with \( p < i < r \). Any adjacency relation \( \mathcal{R}_{p,p}(\gamma) \), for a given \( p \)-cell \( \gamma \), is obtained by retrieving co-boundary relations \( \mathcal{R}_{p-1,p}(\gamma') \) for all the cells \( \gamma' \) in \( \mathcal{R}_{p,p-1}(\gamma) \). All the topological relations are optimal.

The IG data structure is used in the Selective Geometric Complexes (SGCs) [RO89], which describe a non-manifold shape through collections of mutually disjoint cells not necessarily topological, and defined as open subsets of manifolds. In [SG03] the authors introduce the Adjacency and Incidence Framework (AIF), a variant of the IG data structure, which is optimized for cell 2-complexes.

In Section 4.2, a restriction of the IG data structure to simplicial complexes is briefly discussed. Simplified versions of the IG data structure have been developed in the literature. A first variant of the IG data structure is given by the Simplified Incidence Graph (SIG) [DFGH04], a dimension-independent representation for arbitrary simplicial complexes discussed in Section 4.3. Specifically, the Incidence Simplicial (IS) data structure [DFHPC10], is another dimension-independent variant of the IG data structure, which is introduced in Section 4.1.

3.1.3 The Extended Indexed data structure with Adjacencies

In this section, we discuss the Extended Indexed data structure with Adjacencies (EIA) [DF03], an explicit adjacency-based representation of pseudo-manifold simplicial d-complexes.

Following [DFH05], the starting point of our analysis is the Indexed data structure with Adjacencies (IA) [PBCF93, Nie97], which encodes vertices and maximal simplices in a regular pseudo-manifold \( \Sigma \). For each maximal simplex \( \sigma \), it encodes boundary relation \( \mathcal{R}_{d,0}(\sigma) \) and adjacency relation \( \mathcal{R}_{d,d}(\sigma) \). Thus, the storage cost of the IA data structure is \( 2(d + 1)s_d \), where \( s_d \) is the number of \( d \)-simplices in \( \Sigma \). For each \( d \)-simplex \( \sigma \), only adjacency relation \( \mathcal{R}_{d,d}(\sigma) \) and boundary relation \( \mathcal{R}_{d,j}(\sigma) \), with \( j < d \), are optimal.

All the vertex-based co-boundary relations are optimal in an extension of the IA data structure, namely the Extended Indexed data structure with Adjacencies (EIA) [DF03]. It encodes the same topological entities and relations of the IA data structure, plus, for each vertex \( v \), partial co-
boundary relation $R^*_0, d(v)$, which consists of one $d$-simplex for each $d$-cluster in $St(v)$. Recall that a $d$-cluster is a collection of $d$-simplices, which are $(d - 1)$-connected. Let $k^d_0(v)$ be the number of $d$-clusters in $St(v)$. Thus, the storage cost of the EIA data structure is equal to:

$$2(d + 1)s_d + \sum_{v \in \Sigma^0} k^d_0(v)$$

If the input simplicial $d$-complex $\Sigma$ is manifold, then $k^d_0(v) = 1$, for each vertex $v$, and, thus, the storage cost of the EIA data structure becomes $2(d + 1)s_d + s_0$, where $s_0$ is the number of vertices.

Co-boundary relations $R_{0,k}(v)$, with $0 < k \leq d$, are retrieved in time linear in the number of $d$-simplices in $St(v)$. These relations are optimal only for $d \leq 3$. Co-boundary relations $R_{q,k}(\sigma)$, for $0 < q < k < d$, are retrieved by traversing the star of a vertex bounding $\sigma$, thus these relations are local.

Dimension-specific extensions of the EIA data structure are the Triangle-Segment (TS) data structure [DFMPS04], and the Non-Manifold Indexed data structure with Adjacencies (NMIA) [DFH03], which represent, respectively, arbitrary 2D and 3D simplicial shapes embedded in the Euclidean space $E^3$. The TS and NMIA data structures are described in Sections 5.2 and 5.3, respectively. The Generalized Indexed Data Structure with Adjacencies (IA*) [CDFW11], which is introduced in Section 5.1, is another dimension-independent extension of the EIA data structure for non-manifold shapes.

3.2 Representations of Cell and Simplicial 2-Complexes

In this section, we discuss dimension-specific representations of surfaces discretized through cell and simplicial 2-complexes. In the literature, there is a lot of research about representations of 2D simplicial shapes, but an extensive description of these representations is outside the scope of this thesis. Here, we propose a brief review, largely inspired by [DFKP04, DFH05], of the most important data structures.

In Section 3.2.1, we discuss representations of manifold 2-complexes. In Section 3.2.2, we briefly review the Radial Edge (RE) data structure [Wei88b], the first explicit data structures for arbitrary cell 2-complexes. In Section 3.2.3, we discuss the Partial Entities (PE) data structure [LL01], another explicit representation for cell 2-complexes, considerably more compact than the RE data structure. In Section 3.2.4, we briefly review the Vertex-Face (VF) data structure [VL97], representing regular simplicial 2-complexes. In Section 3.2.5, we review the Directed Edges (DE) data structure [CKS98], an explicit edge-based data structure for arbitrary simplicial complexes. Finally, in Section 3.2.6, we propose a brief description of the Loop-Edge (LE) data structure [MM00], an explicit representation for regular simplicial 2-complexes.

Note that the Triangle-Segment (TS) data structure [DFMPS04] is another representation of arbitrary surfaces, discretized by a simplicial 2-complex embedded in the Euclidean space $E^3$. The TS data structure is discussed in Section 5.2.
3.2.1 Representations of Manifold 2-Complexes

In this section, we discuss several representations for manifold cell and simplicial 2-complexes. Specifically, we describe data structures for cell 2-complexes and simplicial 2-complexes in Sections 3.2.1.1 and 3.2.1.2, respectively.

3.2.1.1 Representation of Manifold Cell 2-Complexes

In this section, we briefly review several data structures for representing manifold cell 2-complexes. The content of this section is largely based on [DFKP04, DFH05]. Specifically, we review the Star-Vertex (SV) [KT01], the Winged-Edge (WE) [Bau75], the Double-Connected Edge List (DCEL) [MP78], the Half-Edge [Man87], and the Quad-Edge [GS85] data structures. Recall that, given a simplicial 2-complex Σ, we denote the number of $j$-simplices in Σ as $s_j$, with $0 \leq j \leq 2$.

The Star-Vertex (SV) data structure [KT01] is an adjacency-based representation for manifold cell 2-complexes. In terms of topological relations, it explicitly encodes adjacency relation $R_{0,0}(v)$, for a vertex $v$, and partial relation $R_{2,0}(f)$, which associates a face $f$ with one of its vertices. In the SV data structure, only relations $R_{2,0}$ and $R_{0,0}$ are optimal, while co-boundary relations are local. In terms of storage cost, the SV data structure is the cheapest data structure for cell 2-complexes, but it has not full navigation capabilities. If restricted to manifold simplicial 2-complexes, then its storage cost is $4s_1$, as proposed in [DFH05].

The Winged-Edge (WE) data structure [Bau75], is probably the first explicit representation for cell 2-complexes. The WE data structure explicitly encodes, for each edge $e = (v_0, v_1)$, topological relations $R_{1,0}(e)$, $R_{1,2}(e)$, and $R_{1,1}(e)$, which consists of four edges adjacent to $e$, belonging to two faces $f_0$ and $f_1$ in $St(e)$, as shown in Figure 3.2(a). For each face $f$, the WE data structure encodes partial relation $R_{2,1}(f)$, which consists of one edge bounding $f$, and, for each vertex $v$, partial relation $R_{5,1}(v)$, consisting of one edge in $St(v)$. The WE data structure efficiently supports the retrieval of topological relations. If restricted to manifold simplicial 2-complexes, then its storage cost is $s_0 + 8s_1 + s_2$, as proposed in [DFH05].

The Double-Connected Edge List (DCEL) data structure [MP78] is a simplified version of the WE representation. For each edge $e = (v_0, v_1)$, instead of adjacency relation $R_{1,1}(e)$, it stores only partial adjacency relation $R_{1,1}'(e)$, which consists of two edges, one for each face in $St(e)$, for instance edges $e_1$ and $e_3$ in Figure 3.2(a). Topological relations are optimal in the DCEL data structure. If restricted to manifold simplicial 2-complexes, then its storage cost is $s_0 + 6s_1 + s_2$, as suggested in [DFH05].

The Half-Edge (HE) data structure [Man87] is based on oriented edges, known as half-edges. An edge $e$ is split in two half-edges, according to orientation of two faces in $St(e)$. For each half-edge $e = (v_0, v_1)$, the HE data structure encodes one vertex of $e$, the face $f$ associated with $e$, the previous and next half-edges bounding $f$ with respect to $e$, and the other half-edge corresponding to $e$ (with the reverse orientation). For instance, for half-edge $e$ related to face $f_0$ in Figure 3.2(b),...
references to $v_0, f_0, e_1, e_2,$ and $e'$ are stored. The HE data structure encodes, for vertices and faces, the same relations as in the WE and DCEL data structures. If restricted to manifold simplicial 2-complexes, then its storage cost is $s_0 + 10s_1 + s_2$, as proposed in [DFH05].

The Handle-Edge data structure [LPT+03] extends the Half-Edge data structure to manifold surfaces with boundary, and explicitly represents vertices, edges, and faces.

![Diagram](a) (b)

Figure 3.2: Topological entities encoded in (a) the Winged-Edge and in (b) the Half-Edge data structures for each edge $e = (v_0, v_1)$. For the sake of clarity, these representations are exploded. Note that boundary of faces are oriented in the Half-Edge data structure.

The Quad-edge (QE) data structure [GS85] is an implicit data structure for representing manifold cell 2-complexes. In the QE data structure, for each vertex $v$, edges in $St(v)$ are radially sorted, and edges bounding a face $f$ are ordered in clockwise or counter-clockwise order around $f$. Thus, an edge $e$ belongs to four loops, related to its extreme vertices and faces in $St(e)$, respectively. These edges form a quad-edge associated with $e$, namely relation $R_{1,1}^+(e)$ used in the DCEL data structure. If restricted to manifold simplicial 2-complexes, then its storage cost is $s_0 + 8s_1 + s_2$, as suggested in [DFH05].

In Section 4.4.1, we demonstrate that most of these data structures, restricted to simplicial 2-complexes, are more expensive than the IS data structure [DFHPC10]. For instance, our tests show that the WE and HE data structures are, respectively, 1.42 and 1.7 times more expensive than the IS data structure.

### 3.2.1.2 Representation of Manifold Simplicial 2-Complexes

In this section, we briefly review several data structures for representing manifold simplicial 2-complexes, which are also known in the literature as the triangulations. Specifically, in this section, we also review recent data structures, which have not been described in [DFH05].

All the data structures discussed in this section are Lath-based data structures [JLMC02]. They are implicit data structures for triangulations, which encode vertices and any further information, called a lath, uniquely identified with an element of the input simplicia 2-complex. This organization is equivalent to a cell-tuple, discussed in Section 3.1.1. Here, all the topological relations are optimal. A lath-based data structure does not require separate records for edges and triangles.
The Corner-Table (CoT) data structure [RSS01] is one of the most interesting lath-based data structures. It represents a triangulation Σ through two arrays, namely V and O. Connectivity of triangles is encoded through a corner, which is an integer reference to a triangle t, associated with one of its boundary vertices v. A triangle t is defined by three corners, one for each vertex, sorted in clockwise order. Figure 3.3(a) shows a corner c associated with a vertex v1 in a triangle t. Boundary of t is stored as a tuple (p(c), c, n(c)). In [RSS01] the authors introduce the following set of operators, known as the standard corners operators, for a corner c in a triangle t:

- operators n(c) and p(c), which return, respectively, the next and previous corner of c in triangle t, as shown in Figure 3.3(a);
- operator o(c), which returns opposite corner of c in a triangle t′ adjacent to t through edge in t opposite to c, as shown in Figure 3.3(b);
- operator s(c), which returns swing corner of c with respect to a triangle adjacent to t, as shown in Figure 3.3(c).

In the remainder of this thesis, we confuse the opposite and swing corners of c with o(s) and s(c), respectively.

Opposite corners are stored, for each triangle t, as three consecutive entries in the array O. Following [DFH05], the CoT data structure encodes topological relations R_{2,2}, R_{2,0}, and R_{0,2}. For each triangle t, we encode three references to its corners in the V array, and three references for the opposite corners of vertices in t. As a consequence, the storage cost of the CoT data structure is 6s_2, where s_2 is the number of triangles. In any case, s_2 ≈ 2s_0 in a triangulation Σ [Ede87], where s_0 is the number of vertices in Σ. Thus, the storage cost of the Corner-Table data structure is about 12s_0. All the topological relations are retrieved in optimal time through standard corner operators. In any case, the direct connection among a vertex and its corners is not supported.

In the literature, several variants of the Corner-Table data structure have been introduced. Specifically, we briefly discuss the Sorted Vertex Opposite Table (SVOT) [GR10], the SQuad [GLLR11a],
and the *Laced Ring (LR)* [GLLR11b] data structures.

In [GR10] the authors propose the *Sorted Vertex Opposite Table (SVOT)* data structure, which encodes the same arrays \( V \) and \( O \) of the Corner-Table data structure. In any case, the SVOT data structure does not explicitly encode a reference, which relates a vertex \( v \) and a triangle in \( St(v) \). In other words, it does not explicitly encode corners related to each vertex. This information can be implicitly encoded through a reorganization of triangles and corners. Specifically, if \( j \) is the identifier of a vertex \( v \), then \( 3j \) becomes location of the first corner in the \( j \)-th triangle. In this way, it is possible to access corners related to a vertex \( v \) in constant time and without additional storage. As a consequence, the storage cost of the SVOT data structure is the same as the Corner Table data structure.

In [GLLR11a] the authors propose the *SQuad* data structure, another compact variant of the Corner-Table data structure. They avoid to store explicit references among vertices and corners in order to reduce storage cost of the Corner-Table and SVOT representations. This information, which relates a vertex \( v \) and a corner of a triangle in \( St(v) \), can be easily retrieved by the swing operator \( s(c) \). It can be proven that, given a corner \( c \), we can retrieve opposite corner \( o(c) \) as \( o(c) = p(s(p(c))) \), and swing corner \( s(c) \) as \( s(c) = n(o(n(c))) \). Thus, for each triangle \( t \), we store only two references to swing corners, instead of three references to opposite corners. As a consequence, the \( V \) array of the Corner-Table data structure is not stored anymore, while the \( O \) array is replaced by a table \( S \), which contains references to swing corners. The SQuad data structure stores, on average, two references per triangle, hence its storage cost is about \( 2s_2 \), where \( s_2 \) is the number of triangles. In any case, \( s_2 \approx 2s_0 \) in a triangulation \( \Sigma \) [Ede87], where \( s_0 \) is the number of vertices in \( \Sigma \). Thus, storage cost of the SQuad data structure is about \( 4s_0 \). All the topological relations are optimal in the SQuad data structure.

In [GLLR11b] the authors propose the *Laced Ring (LR)* data structure. The LR data structure is based on the reorganization of vertices in a triangulation \( \Sigma \) along a nearly-Hamiltonian path, which visits all the vertices in \( \Sigma \). The optimal solution consists of the Hamiltonian cycle of edges, which has been studied in graph theory: in any case, it is a NP-complete problem [Kar72]. The authors identify an oriented 1-path formed by edges in \( \Sigma \), which visits most of vertices in \( \Sigma \). Since the input triangulation \( \Sigma \) is manifold, at most two triangles are incident at each edge of this path, which is called the ring. Edges and vertices belonging to this path are known as ring edges and ring vertices, respectively. Ring vertices are stored in the same order they are traversed in the ring. Figure 3.4 shows an example of ring in a triangulation, where ring edges and vertices are depicted in red.

A triangle with at least one ring edge is said to be a *ring triangle*. The LR data structure encodes only ring vertices and triangles in two tables, denoted as \( L \) and \( R \), while remaining vertices and triangles are stored in a CoT data structure. Let \( v \) be a ring vertex and \( v_n \) be the next ring vertex following \( v \) in the ring, then a ring edge \( e = (v, v_n) \) bounds, at most, two ring triangles, namely \( t_L \) and \( t_R \), as shown in Figure 3.4. Then, triangles \( t_L \) and \( t_R \) are moved in locations \( 2v \) and \( 2v + 1 \), and vertices opposite to \( e \) in these triangles (namely \( v_L \) and \( v_R \)) are stored in \( L[v] \).
Figure 3.4: An example of ring, depicted in red, in a manifold triangulation. Each ring edge \(e\) bounds at most two triangles \(t_L\) and \(t_R\), thus we store the opposite vertices \(v_L\) and \(v_R\) of \(e\) in \(t_L\) and \(t_R\), respectively.

and \(R[v]\), respectively. In other words, we store opposite corners of triangles \(t_L\) and \(t_R\) through edge \(e\). Experimental results provided by the authors prove that the storage cost of the LR data structure is about 1.1 references per triangle. The number \(s_2\) of triangles in a triangulation \(\Sigma\) is \(s_2 \approx 2s_0\) [Ede87], where \(s_0\) is the number of vertices in \(\Sigma\). Thus, the storage cost of the LR data structure is about \(2.2s_0\). In spite of its compactness, the LR data structure supports random-access operators, and all the topological relations are optimal. In any case, the LR representation does not efficiently support editing operators, since the ring may be modified, and thus it is mandatory to modify most of the LR data structure.

### 3.2.2 The Radial Edge Data Structure

In this section, we briefly review the *Radial Edge (RE)* data structure [Wei88b], the first explicit data structure proposed in the literature for arbitrary cell 2-complexes. We follow the definition of the RE data structure given in [DFH05].

The RE data structure has been developed in order to describe the decomposition of the boundary of non-manifold 3D shapes. This decomposition is not a cell complex as defined in algebraic topology, since faces are not necessarily homeomorphic to closed disks, but they can be multiply-connected 2-manifolds with boundary. Here, connected components formed by edges bounding a face are known as the *loops*. A region is a solid object bounded by a collection of shells. In other words, a shell is the oriented boundary of a region, which consists of maximal connected sets of faces. In addition, faces, loops, edges, and vertices are characterized by their orientations, called as *face-uses*, *loop-uses*, *edge-uses*, and *vertex-uses*, respectively. Figure 3.5 shows a cell 2-complex formed by two shells, sharing a face. A face \(f\) is associated with two face-uses, namely \(f_u\) and \(f'_u\), corresponding to orientations of \(f\). Oriented boundary of a face-use is described by a loop-use, formed by a circular list of oriented edges, known as the edge-uses. An edge-use associates an edge \(e\) with the orientation induced on \(e\) by face-uses to which it belongs. Figure 3.5(a) shows
two face-uses \( f_u \) and \( f'_u \) of a face \( f \), imposed by two orientations of \( f \), and two loop-uses bounding \( f \). A top 1-simplex \( w \) is described by two edge-uses, one for each orientation of \( w \). A vertex-use associates a vertex \( v \) with one edge-use originating from \( v \). Each edge \( e \) is bounded by two vertices, and it can be oriented in two possible directions for each face in \( St(e) \), then a vertex \( v \) is associated with two possible edge-uses, one for each edge in \( St(v) \). Figure 3.5(b) shows a vertex \( v \) associated with five oriented faces, and, thus, five vertex-uses, one for each face in \( St(v) \). Here, orientations of a face are represented through an oriented arrow, and edge-uses are implicitly defined through the right-hand rule.

The content of the RE data structure is quite complex. Here, we limit our attention to a simplified version of the RE data structure, proposed in [DFH05], which represents a cell 2-complex, in which all the faces are bounded by exactly one loop. Only vertices, edges, faces, face-uses, edge-uses, and vertex-uses are stored. As proposed in [DFH05], this simplified version of the RE data structure can be formalized in terms of topological relations as follows:

- for each vertex \( v \), co-boundary relation \( R_{0,1}(v) \);
- for each edge \( e \):
  - boundary relation \( R_{1,0}(e) \) and co-boundary relation \( R_{1,2}(e) \);
  - partial adjacency relation \( R^*_{1,1}(e) \), which is defined as two pairs of edges adjacent to \( e \) bounding all the faces in \( St(e) \), such that elements in positions \( 2i \) and \( 2i + 1 \) belong to face in position \( i \) in \( R_{1,2}(e) \);
- for each face \( f \), partial boundary relation \( R^*_{2,1}(f) \), which consists of an arbitrarily selected edge bounding \( f \).
Note that this formalization does not explicitly take into account orientations imposed by face-uses, edge-uses, and vertex-uses. These orientations are implicitly described by relations $R_{1,0}(e)$, $R_{1,1}^*(e)$, and $R_{1,2}^*(e)$. As discussed in [DFH05], all the topological relations are optimal. However, the RE data structure is not scalable to manifolds. The RE data structure requires about four times as much storage cost as the WE data structure, if restricted to manifolds [LL01]. Storage cost of the RE data structure, restricted to simplicial 2-complexes, is $s_0 + 4s_1 + s_1 + 73s_2$, as proposed in [DFH05], where $s_j$, with $0 \leq j \leq 2$, is the number of $j$-simplices in the input simplicial 2-complex. In Section 4.4.1, we demonstrate that the RE data structure is about 7.7 times more expensive than the IS representation [DFHPC10].

Several extensions of the RE data structure have been developed in the literature. For instance, the Tri-Cyclic Cusp representation [GCP90] extends the RE data structure with new elements, namely the cusps, which handle inclusion relations of topological disks at non-manifold vertices. The Coupling Entities representation [YK95] is another improvement over both the RE and the Tri-Cyclic Cusp data structures, and introduces additional entities, which describe several topological relations for loops, radial cycles, and cycles.

### 3.2.3 The Partial Entities Data Structure

In this section, we briefly review the Partial Entities (PE) data structure [LL01], another explicit representation of cell 2-complexes, but considerably more compact than the RE data structure, discussed in Section 3.2.2. Here, we follow the definition of the PE data structure given in [DFH05].

Following [DFH05], the primary difference between the PE and RE data structures regards faces orientation. In the RE data structure, face-uses describe orientations of a face $f$. In the PE data structure, orientation of $f$ is geometrically defined by its normal, thus orientation of its boundary is uniquely defined. A face $f$ is bounded by a cycle of oriented partial-edges. Each partial-edge corresponds to an edge bounding $f$. Hence, if there are $m$ faces incident at any edge $e$, then $m$ partial-edges corresponding to $e$ are stored. In this context, a top edge $w$ is described through two partial-edges. Partial-faces are basic components of shells: each face may belong to at most two shells, and, thus, it has two partial-faces. Finally, a partial-vertex is a copy of a non-manifold vertex, shared by several manifold surfaces.

Following [DFH05], the PE data structure encodes almost the same relations as the RE data structure, except co-boundary relation $R_{0,1}(v)$, for a vertex $v$, which is replaced with partial co-boundary relation $R_{0,1}^*(v)$. An implementation of the PE data structure, described in [LL01], has half the storage cost of the RE representation for non-manifold cell 2-complexes. In the RE data structure, an edge-use is associated with a face-use, and, since there are two face-uses for each face, the number of edge-uses is twice the number of partial-edges in the PE data structure. In any case, the PE data structure, restricted to manifolds, requires twice as much space as that of the WE data structure, discussed in Section 3.2.1.1.

Following [DFH05], the storage cost of the PE data structure, restricted to simplicial 2-complexes,
is \(s_1 + 4s'_1 + 22s_2 + C_v\), where \(C_v\) is the number of connected components in the star of all vertices, and \(s_j\), with \(0 \leq j \leq 2\), is the number of \(j\)-simplices in the input simplicial 2-complex. In addition, \(s'_1\) is the number of top edges. In Section 4.4.1, we demonstrate that the PE data structure is about 2.7 times more expensive than the IS representation [DFHPC10].

### 3.2.4 The Vertex-Face Data Structure

In this section, we briefly review the *Vertex-Face (VF) data structure* [VL97], an explicit data structure for regular simplicial 2-complexes. Here, we follow the definition of the VF data structure given in [DFH05].

Given a simplicial 2-complex \(\Sigma\), the VF data structure encodes explicitly all the vertices, edges, triangles in \(\Sigma\), plus the following topological relations:

- for each triangle \(t\), boundary relation \(R_{2,1}(t)\), which contains all the edges bounding \(t\);
- for each edge \(e\), boundary relation \(R_{1,0}(e)\), which contains all the vertices bounding \(e\);
- for each vertex, co-boundary relation \(R_{0,2}(v)\), which contains all the triangles in \(St(v)\).

Thus, the storage cost of the VF data structure is \(2s_1 + 6s_2\), as proposed in [DFH05], where \(s_j\), with \(0 \leq j \leq 2\), is the number of \(j\)-simplices in \(\Sigma\).

Boundary relations and relations \(R_{0,0}(v), R_{0,1}(v), \) and \(R_{0,2}(v)\) are optimal. Edge-based co-boundary relations are local, since we consider co-boundary relation \(R_{0,2}\) for each vertex bounding an edge. Adjacency relation \(R_{2,2}\) is local, since it is retrieved through relations \(R_{2,1}\) and \(R_{1,2}\).

### 3.2.5 The Directed Edges Data Structure

In this section, we briefly review the *Directed Edges (DE) data structure* [CKS98], an explicit edge-based representation for arbitrary simplicial complexes. The DE data structure is an extension of the HE data structure, discussed in Section 3.2.1.1. This section is largely based on [DFH05].

The DE data structure is based on the concept of directed edge. A directed edge \(e_d\) of an edge \(e\) is an oriented occurrence of \(e\), related to a triangle in \(St(e)\). The concept of directed edge is similar to edge-use and partial-edge in the RE and PE data structures, discussed in Sections 3.2.2 and 3.2.3, respectively. In the DE data structure, only directed edges and vertices are encoded. Triangles are implicitly referenced through directed edges belonging to their oriented boundary. Top edges are represented as directed edges.

Following [DFH05], if we consider undirected edges instead of directed edges, then it is clear that the DE data structure encodes exactly the same topological relations as the PE data structure, described in Section 3.2.3. With this modification, any information regarding an oriented edge \(e\) is transferred to boundary relation \(R_{1,0}(e)\), and to partial adjacency relation \(R_{1,1}(e)\), if \(e\) is not top.
Following [DFH05], the storage cost of the PE data structure, restricted to simplicial 2-complexes, is 

\[ 15s_2 + 2s'_1 + C_v, \]

where \( C_v \) is the number of connected components in the star of vertices, \( s_2 \) and \( s'_1 \) are, respectively, the number of triangles and top triangles in the input simplicial 2-complex. In Section 4.4.1, we demonstrate that the DE data structure is about 1.7 times more expensive than the IS representation [DFHP10]. In any case, the DE data structure is scalable to manifolds. Its storage cost is 68\( s_2 \) bytes [CKS98], which is only 1.13 times the storage cost of the WE data structure, discussed in Section 3.2.1.1.

### 3.2.6 The Loop-Edge Data Structure

In this section, we propose a brief description of the Loop-Edge (LE) data structure [MM00], an explicit representation for regular simplicial 2-complexes, in which non-manifold singularities occur only at edges.

Here, the key concept of the LE data structure consists of edge-use. In this context, an edge-use is similar to any partial-edge in the PE data structure, described in Section 3.2.3. An edge-use associates an edge \( e \) with a triangle \( t \) in \( S(t(e)) \). If the edge \( e \) is non-manifold, an edge-use can be used for encoding several triangles which are incident at \( e \). In this case, all the triangles incident at a non-manifold \( e \) can be sorted in counter-clockwise or clockwise order around \( e \). Given an edge-use \( (e,t) \), adjacency relation \( R_{2,1}(t) \) along edge \( e \) can be encoded through partial co-boundary relation \( R_{1,2}^*(e) \). This latter encodes the triangle adjacent to \( t \) which follows \( t \) in counter-clockwise or clockwise order around \( e \). The LE data structure encodes all the vertices, edges, triangles, and edge-uses, plus the following topological relations:

- for each vertex \( v \), partial co-boundary relation \( R_{0,1}^*(v) \);
- for each edge \( e \), boundary relation \( R_{1,0}^*(e) \) and relation \( R_{1,2}^*(e) \);
- for each triangle \( f \), boundary relation \( R_{2,1}^*(f) \).

It is clear that the LE representation encodes almost the same relations as the PE data structure, discussed in Section 3.2.3. Hence, all the topological relations are optimal. In [MMHS01] the authors introduce an out-of-core algorithm for constructing the LE data structure.

### 3.3 Representations of Cell and Simplicial 3-Complexes

In this section, we discuss dimension-specific representations of volumetric shapes discretized through cell and simplicial 3-complexes. In the literature, there are few representations for volumetric shapes, and most of them are restricted to manifold 3D shapes. Most of this section is largely based on [DFH05].

In Section 3.3.1, we review the Facet-Edge (FE) data structure [DL89], an implicit representation for manifold cell 3-complexes. In Section 3.3.2, we discuss and review the Handle-Face (HF)
data structure [LT07], an explicit representation for manifold cell 3-complexes. These complexes are usually known as the tetrahedral grids. Finally, in Section 3.3.3, we describe the Extended Maps (X-Maps) data structure [CK10], an implicit representation for arbitrary cell complexes of dimension up to 3.

Note that the Non-Manifold Indexed data structure with Adjacencies (NMIA) [DFH03] is another representation of non-manifold simplicial shapes, discretized by simplicial 3-complexes embedded in the Euclidean space $\mathbb{E}^3$. The NMIA representation is described in Section 5.3.

3.3.1 The Facet-Edge Data Structure

In this section, we propose a brief discussion about the Facet-Edge (FE) data structure [DL89], an implicit representation for manifold cell 3-complexes. The FE data structure is an extension of the QE data structure, described in Section 3.2.1.1, to cell 3-complexes. In this section, we follow the definition of the FE data structure proposed in [DFH05].

Given a cell 3-complex $\Gamma$, the FE data structure encodes all the vertices in $\Gamma$, and the so-called facet-edges, defined between faces and edges in $\Gamma$. Boundary of a face is formed by a ring (namely a loop) of edges, called a face-ring. A face-ring may be ordered in two directions. Conversely, the star of an edge $e$ contains a ring of faces, known as an edge-ring, which can be sorted in two directions. A facet-edge pair $(f,e)$ uniquely associates a face $f$ with an edge $e$ bounding $f$, and, thus, a face-ring of $f$ is associated with an edge-ring of $e$. As a consequence, it is possible to define four versions of a facet-edge, considering all the possible orientations. Figure 3.6(a) shows a facet-edge $(f,e)$ in a cell 3-complex formed by two 3-cells sharing a face $f$. Figure 3.6(b) shows all the possible versions a facet-edge $(f,e)$. Here, for the sake of clarity, the orientation of a face is represented through an oriented arrow.

![Figure 3.6](image)

Figure 3.6: (a) A facet-edge $(f,e)$ in a cell 3-complex, formed by two 3-cells sharing a face $f$. (b) Given the possible orientations of $f$, and an edge $e$, represented through oriented arrows, four possible versions of the facet-edge $(f,e)$ can be defined. Here, for the sake of clarity, the orientation of a face is represented through an oriented arrow.
For each face-edge \((f, e)\), the FE data structure encodes the previous and next facet-edges in both the face-ring of \(f\) and edge-ring of \(e\): for instance, in Figure 3.6, the previous and next faces-edges of \((f, e)\) are \((f_1, e)\) and \((f_2, e)\), respectively. Furthermore, a facet-edge \((e, f)\) has a dual version \((e^*, f^*)\), which is a facet-edge defined in the dual cell 3-complex. Here, edge \(e^*\) corresponds to the \(f\) and face \(f^*\) corresponds to edge \(e\).

The FE data structure encodes the following topological relations:

- for each face \(f\), boundary relation \(R_{2,1}(f)\), expressed in terms of facet-edges, and co-boundary relation \(R_{2,3}(f)\), implicitly expressed in terms of vertices;
- for each edge \(e\), boundary relation \(R_{1,0}(e)\), implicitly defined by vertices of \(e\), co-boundary relation \(R_{1,2}(e)\), expressed in terms of face-rings, and partial adjacency relation \(R_{1,1}^*(e)\) in terms of edge-rings;
- for each vertex \(v\), co-boundary relation \(R_{0,1}(v)\), expressed as boundary relation \(R_{1,0}\) in the dual complex;
- for each tetrahedron \(t\), boundary relation \(R_{3,2}(t)\), expressed as co-boundary relation \(R_{2,3}\) in the dual complex.

All the topological relations are efficiently retrieved from the FE data structure. Given a cell 3-complex \(\Gamma\), the storage cost of the FE data structure is equal to:

\[
\sum_{v \in \Gamma^0} \text{val}(v) + \sum_{e \in \Gamma^1} \text{deg}(e) + 4 \sum_{f \in \Gamma^2} \text{deg}(f)
\]

where \(\Gamma^0\), \(\Gamma^1\), and \(\Gamma^2\) are the collections of vertices, edges, and triangles in \(\Gamma\), respectively.

Following [DFH05], if we restrict the FE data structure to simplicial 3-complexes, then there is a constant number of triangle-edge pairs for each triangle, which allows for an efficient implementation. In fact, in this case, the storage cost of the FE data structure becomes \(2s_1 + 12s_2 + 4s_3\), where \(s_j\), with \(0 \leq j \leq 3\), is the number of \(j\)-simplices in the input simplicial 3-complex. In Section 4.4.2, we demonstrate that the FE data structure is about 1.7 times more expensive than the IS data structure [DFHPC10], restricted to manifold simplicial 3-complexes.

In [Muc93] the author proposes the Triangle-Edge (TE) data structure, an extension of the FE data structure to manifold simplicial 3-complexes, which does not take into account the dual complex. Here, any triangle-edge pair \((t, e)\) relates a triangle \(t\) and an edge \(e\) bounding \(t\), and corresponds to a triangle-ring of \(t\) and to an edge-ring of \(e\). As a consequence, there exist four possible versions of a triangle-edge pair. In any case, since we limit the scope to simplicial complexes, there is a constant number of triangle-edge pairs for each triangle, corresponding to the three vertices of a triangle. This fact allows for an efficient implementation. The TE data structure has been employed while reducing the complexity of tetrahedral grids [NE04].
3.3.2 The Handle-Face Data Structure

In this section, we briefly review the Handle-Face (HF) data structure [LT97], an explicit representation for cell 3-complexes. The HF data structure is similar to the RE and PE data structures, discussed in Sections 3.2.2 and 3.2.3, respectively. In this section, we follow the definition of the HE data structure proposed in [DFH05].

The HF data structure is based on the concept of half-face. A half-face $h_f$ corresponds to one possible orientation of a face $f$ in a cell 3-complex. A half-face is bounded by a cycle of oriented edges, which are oriented according to the orientation of $h_f$. Figure 3.7 illustrates several half-faces for a cell 3-complex formed by two 3-cells. Here, for the sake of clarity, the orientation of a face is represented through an oriented arrow, imposed through the right-hand rule. Figure 3.7(b) shows two half-faces with opposite orientation, but representing the same face.

![Figure 3.7](image)

Figure 3.7: (a) Examples of half-faces in a cell 3-complex formed by two 3-cells. Here, for the sake of clarity, the orientation of a face is represented through an oriented arrow, imposed through the right-hand rule. (b) Two half-faces with opposite orientation, but representing the same face.

Given a cell 3-complex $\Sigma$, the HF data structure directly encodes all the half-faces, edges, and vertices in $\Sigma$, which are known as the basic entities. Furthermore, it encodes all the surface entities, which describe the boundary of 3-cells in $\Sigma$. The HF data structure further distinguishes surface entities, which bounds $\Sigma$, and surfaces that are in the interior. Here we briefly review a simplified version of the HF data structure, proposed in [DFH05], where all the surface entities describe the boundary of 3-cells.

The HF data structure encodes the following topological relations:

- for each vertex $v$, partial co-boundary relation $R^*_{0,1}(v)$, which encodes one edge in $St(v)$;
- for each edge $e$, partial adjacency relation $R^*_1(e)$, which encodes all the edges adjacent to $e$ sharing a common face with $e$, and co-boundary relation $R^*_{1,2}(e)$, which encodes all the faces in $St(e)$: these relations are radially sorted around $e$ in a coherent way;
- for each half-face $f$, partial boundary relation $R^*_2(f)$, which encodes one edge bounding the face corresponding to $f$. 
Following [DFH05], the HF representation encodes the same relations as in the RE and PE data structures, discussed in Sections 3.2.2 and 3.2.3, respectively, but it cannot represent arbitrary shapes. As all the representations encoding orientations, the HF data structure is quite verbose. All the topological relations are optimal in the HF data structure, including boundaries of 3-cells, even if they are not explicitly represented.

The Compact Half-Face (CHF) data structure [LLLV05] is a specialization of the HF data structure for manifold simplicial 3-complexes. The CHF data structure is a multi-level representation, which is formed by four levels, numbered as $0 - 3$. Here, we discuss only levels up to 2. Level 3 addresses information boundary of the manifold 3-complex, whereby boundary simplices are encoded. In any case, this level can be considered as a sort of application-specific layer for the CHF data structure, and it can be discarded. Here, we follow the definition of the CHF data structure discussed in [DFH05].

Given a tetrahedral grid $\Sigma$, the CHF data structure encodes basic entities in $\Sigma$, namely tetrahedra, edges, and vertices of $\Sigma$, plus surface-entities, namely half-triangles and half-edges. Each half-triangle corresponds to the orientation of a triangle bounding a tetrahedron in the same spirit of a half-face in the HF data structure. Each half-triangle is bounded by a circular list of half-edges, whose orientation is aligned with that of the related half-triangle.

In the remainder of this section, given a tetrahedral grid $\Sigma$, we denote the number of $j$-simplices in $\Sigma$, with $0 \leq j \leq 3$, as $s_j$.

Level 0 of the CHF data structure encodes only vertices and tetrahedra in $\Sigma$. Furthermore, it explicitly encodes, for any tetrahedron $t$, boundary relation $R_{3,0}(t)$, which consists of all the vertices bounding $t$. Half-triangles and half-edges are not explicitly encoded. Given a tetrahedron $t$, half-triangles and half-edges of $t$ can be combinatorially expressed in terms of vertices bounding $t$. At this level, only topological relation $R_{3,0}$ is optimal. The storage cost of Level 0 is $4s_3$, as discussed in [DFH05].

Level 1 of the CHF data structure encodes explicitly only boundary relation $R_{3,0}(t)$ for each tetrahedron $t$ in $\Sigma$. Conversely, adjacency relation $R_{3,3}(t)$ is implicitly encoded through the pairing information of all the half-triangles, which belong to the same triangle. As a consequence, mate half-edges can be implicitly represented, and co-boundary relation $R_{1,3}(e)$ is partially encoded for each edge $e$ implicitly represented by its half-edges. Hence, all the topological relations for tetrahedra, triangles, and edges are supported, since edges and triangles can be expressed as half-faces and half-edges of tetrahedra. The storage cost of Level 1 is $4s_3$, as discussed in [DFH05].

Level 2 of the CHF data structure explicitly encodes boundary relation $R_{4,0}(t)$, and adjacency relation $R_{3,3}(t)$, for each tetrahedron $t$ in $\Sigma$, plus partial co-boundary relation $R_{0,3}(v)$, which encodes one tetrahedron incident at a vertex $v$. In addition, all the triangles and edges are encoded explicitly for the purpose of attribute assignment. Each triangle $f$ is mapped on one half-triangle sharing it. A half-edge defined by its vertices is mapped on one of its incident half-triangles. At this level, all the topological relations are optimal. The storage cost of Level 2 is $s_0 + s_1 + s_2$, as
discussed in [DFH05].

Following [DFH05], the total storage cost of the CHF data structure is $s_0 + s_1 + s_2 + 8s_3$. In Section 4.4.2, we demonstrate that the IS data structure [DFHPC10], restricted to manifold simplicial 3-complexes, is about 1.6 times more expensive than the CHF data structure.

In [GR10] the authors demonstrate that the CHF and CoT data structure, extended to tetrahedral grids, are equivalent. In addition, they define the Sorted Opposite Table (SOT) data structure [GR09], where they generalize a corner as an association between a vertex $v$ and a tetrahedron $t$ incident at $v$ in the same spirit of the SVOT representation, discussed in Section 3.2.1.2. In other words, the SOT data structure can be considered as an extension of the SVOT representation to manifold simplicial 3-complexes.

### 3.3.3 The Extended Maps Data Structure

In this section, we discuss the Extended Maps (X-Maps) data structure [CK10], an implicit representation for arbitrary cell complexes of dimension up to 3. It is a specialization of the N-G-map data structure [Lie94]. This latter is equivalent to the Cell-Tuple data structure, described in Section 3.1.1.

The X-Maps data structure is based on an unique topological entity, known as the dart, which is an half-edge. It provides a compact representation of topological links, and an ordered access to adjacency relations through an extended-map, which is equivalent to a cell-tuple. Formally, an extended-map is defined as a tuple $(B, \psi_1, \psi_2, \psi_3, \theta)$, where $B$ is a finite set of darts. Darts can be combined together in an oriented face through permutation $\psi_1$. In order to form oriented volumes, pairs of oriented faces can be combined together along common edges through permutation $\psi_2$: in this case, $\psi_2$ must be an involution, i.e. a permutation such that $\psi_2 \circ \psi_2 = Id$. This means that only a whole edge is shared by the input pair of faces. Pairs of oriented volumes can be combined together along common faces through permutation $\psi_3$. In order to guarantee a consistent union of oriented volumes, an integrity statement $\psi_1 \circ \psi_3 = Id$ must hold. This means that only whole oriented faces are combined through permutation $\psi_3$. Darts can be combined together through permutation $\theta$ along a non-manifold vertex. Figure 3.8 shows a cell 2-complex, represented through the X-Maps data structure. We consider two possible representations related to permutation $\psi_1$ and involution $\psi_2$, respectively.

In the X-Maps data structure, a cell $\gamma$ is defined by its dimension and by one of its darts $x_\gamma$. Other darts of $\gamma$ are retrieved by navigating from $x_\gamma$ through encoded permutations and involutions, known as relations. This operation is known as the orbit of a given dart $x_\gamma$, and, given two relations $\psi'$ and $\psi''$, it is denoted as $< \psi', \psi'' > (x_\gamma)$. In this context, volumes are expressed as the orbit $< \psi_1, \psi_2 >$, edges as the orbit $< \psi_2, \psi_3 >$, and vertices as $< \psi_1 \circ \psi_2 \circ \psi_3, \theta >$. All the topological relations are retrieved in optimal time and reformulated in terms of orbits.

The X-Maps data structure is flexible, and allows describing a wide range of domains by encoding several type of relations. For instance, if $\theta$ is the identity for each dart $x$, then the input domain
is manifold. The X-Maps representation is a very compact data structure for cell complexes: for instance, as reported by authors, the PE data structure, described in Section 3.2.3, requires from about 160% to 400% of the storage cost required from the specialization of the Extended Maps for cell 2-complexes. The X-Maps data structure also represents simplicial complexes of dimension up to 3. In any case, as reported in [CK10], it is more expensive than the Simplified Incidence Graph (SIG) data structure [DFGH04], if restricted to manifold simplicial 3-complexes. As demonstrated in Section 4.4.1, the X-Maps data structure is as compact as the IG data structure, if restricted to manifold simplicial 2-complexes. Specifically, it is 1.26 times more expensive than the IS data structure [DFHPC10]. In Section 4.4.2, we prove that the X-Maps is about 2.4 times more expensive than the IS data structure, if restricted to manifold simplicial 3-complexes.

3.4 Decomposition-based data Representations

In this section, we discuss several decompositions of an arbitrary shape into simpler parts. Most of topics discussed in this section are largely inspired by [DFH05].

Following [DFH05], decomposition-based approaches are either interior-based or boundary-based [Sha08]. Interior-based approaches implicitly subdivide a simplicial shape by describing it as a geometric or a topological skeleton [CSM07]. Boundary-based methods provide a decomposition of object boundaries into several parts by considering local properties, like critical features or curvature [SM95]. These latter methods aim at decomposing an object into meaningful components, i.e., components, which can be perceptually distinguished from the remaining part of the object. Generally speaking, most of boundary-based approaches proposed in the literature try to provide a stratification of a shape in the discrete case, initially defined for analytic sets [Whi65].

In our research, we are more interested in boundary-based approaches, since they may be exploited
as basis of data structures for representing arbitrary shapes. We restrict our attention to decompositions of non-manifold simplicial shapes along non-manifold singularities. We should remove as many singularities as possible without introducing arbitrary cuts through manifold parts. Hence, a non-manifold shape is decomposed in almost manifold components, which exhibit a common intersection formed by several non-manifold singularities. In this way, we reduce the complexity of an arbitrary shape by obtaining a representation, which highlights components and their connectivity. Under these assumptions, a decomposition into manifold components is possible only for 2-complexes. In three or higher dimensions, a decomposition into manifold components may introduce artificial "cuts" in the input object, and create almost manifold components [DFMMP03]. In six or higher dimensions, this decomposition is not feasible, since the class of manifolds is not decidable, as discussed in Section 2.3.

In Section 3.4.1, we initially restrict our attention to decompositions of regular shapes by cutting them along non-manifold singularities. In any case, it is possible to generalize these approaches, and to define similar decompositions of arbitrary shapes into almost manifolds. The key idea consists of considering an arbitrary shape as the union of regular components, and composing together decompositions of these regular components. In Section 3.4.2, we briefly discuss the Combinatorial Stratification [PTL04] of an arbitrary cell 2-complex. In Section 3.4.3, we discuss the Initial Quasi-Manifold (IQM) Decomposition [DFMMP03] for an abstract simplicial complex of arbitrary dimension.

In Chapter 8, we also discuss a sound decomposition of arbitrary shapes, namely the Manifold-Connected (MC) Decomposition [HDF07a, HDF07b]. This decomposition provides a structural model of a shape discretized through a simplicial complex, in which we highlight components relevant from a topological point of view, and their connectivity. This decomposition is uniquely based on topological properties, and does not take in account any geometric information.

### 3.4.1 Decompositions of Regular Shapes

In this section, we restrict our attention to decompositions of regular shapes, which are the boundary of a solid, not necessarily manifold. Here, we follow the same approach proposed in [DFH05].

A non-manifold shape is cut along their non-manifold singularities. The resulting decomposition is a collection of singularities-free components, which are represented by data structures for manifold complexes, like those described in Section 3.2. In this context, cuts are local, since topology of the input shape changes only in the neighborhood of a non-manifold singularity.

A regular shape is the limit of a sequence of manifolds [DS92], where their distance is measured through the Hausdorff metric [Ago05]. In this way, we can introduce a minimal set of operators, known as the regularization operators, which are sufficient for manipulating regular sets. Hence, it is possible to apply modeling tools developed for manifold shapes to regular shapes.

An extension of this approach has been presented in [FR92], where the resulting decomposition is represented as the Two-manifold Cell Decomposition (TCD) graph, where each node corresponds
to a 2-manifold component of a regular shape, while each arc defines a non-manifold adjacency among two or more components. This graph has been exploited for identifying form features in regular shapes.

Another possible application is the conversion of a regular shape to a manifold representation by removing non-manifold singularities. Most of techniques proposed in the literature operate on surfaces, discretized through cell and simplicial 2-complexes.

In [GTLH98] the authors propose a technique for removing non-manifold singularities from a non-manifold surface. This technique does not address geometric issues, and it is uniquely based on topological aspects. This strategy is based on two high-level operations, namely cutting and stitching. The cutting operation involves the identification of non-manifold edges, and the cutting of a surface along these edges, in such a way that, given a non-manifold vertex \( v \), 2-cells in \( St(v) \) belong to the same component. Two strategies are available for cutting, namely the global and local methods. The global method, which operates on all the elements, is appropriate for cuts covering a large portions of the input surface. The local strategy, which operates only on a subset of vertices and edges, is more efficient in case of a small number of marked elements. As a result of cutting operations, we obtain a manifold surface, which may contain several boundary edges, corresponding to non-manifold edges in the input surface. Hence, a stitching operation is performed, guaranteeing a manifold surface as result. It involves joining two boundary edges. There are two greedy strategies for stitching, namely the pinching and snapping strategies. Pinching strategy attempts to simply zip boundary edges created during cutting operations. Snapping strategy attempts to stitch along boundaries not created in the cutting phase, and reduces the number of connected components in the shape.

In [RC99] the authors improve the technique introduced in [GTLH98] by taking in account also geometric issues. In this context, a non-manifold simplicial shape is described through an indexed representation formed by two tables, namely the vertex table \( V \) and triangle table \( TV \), which contain, respectively, all the vertices and an index in \( V \) for all the vertices of a triangle \( t \). A triangle adjacency table \( TA \) is also stored, where, for each triangle \( t \), all the indices of triangles adjacent to \( t \) are stored. As stated in Section 2.3, in a simplicial 2-complex, an edge \( e \) is non-manifold if its star contains more than two triangles. Each non-manifold edge is split a number of times sufficient to assign at most two incident triangles to each copy. Resulting representation is known as the Edge-Manifold representation, and may still contain non-manifold vertices. Then, in order to guarantee a manifold topology, it is mandatory to identify and duplicate remaining non-manifold vertices. Authors demonstrate that each non-manifold vertex \( v \) has to be replicated at least as many instances as the number of loops in boundary of \( St(v) \). In other words, the objective of this step consists of finding a new \( TA \) table, which minimizes the number of replications of \( v \), and does not produce self-intersections. Hence, each triangle in \( St(v) \) becomes incident at one copy of \( v \). Since only one of the instances can be associated with the same index originally used for \( v \), all the triangles, which are now incident at the other copies of \( v \), must be updated.

Recent variational meshing techniques [MBTF03, CDMM04, ACSYD05, TWAD09] allow convert-
ing a surface to a volumetric shape. These techniques define non-convex energies to be minimized through local modifications in the current shape. In this context, the placement of vertices in inner parts guarantees both a smooth transition in sampling density and well-shaped tetrahedra. Nevertheless, these volumetric shapes are not guaranteed to be manifold.

In [AFG07, AGFF07] the authors extend the technique described in [RC99] for converting non-manifold volumetric shapes into combinatorial manifold 3-complexes, by applying local updates on non-manifold singularities. For the sake of simplicity, new vertices, introduced while splitting singularities, are assigned the same Euclidean coordinates of a non-manifold vertex. Hence, this method produces a pseudo-manifold, and the output complex must be further modified in order to become a combinatorial manifold. The “displacement” of vertices to new positions requires several consistency checks, because new position of vertices might cause the realized complex either to self-intersect, or to contain “bad” tetrahedra. In order to overcome this problem, the authors suggest to remove a small portion of the neighborhood of a singularity. They prove that this operation is possible under several assumptions, and the removal is local in both combinatorial and geometric sense. The authors also show that this technique can be implemented through the Tetrahedral Data Structure [BDF90], a data structure for manifold volumetric shapes.

3.4.2 The Combinatorial Stratification

In this section we propose a brief description of the Combinatorial Stratification [PTL04], directly based on the stratification of analytic sets [Whi65]. Also in this case, we follow the same approach proposed in [DFH05].

In this representation, a cell 2-complex is decomposed into subcomplexes, which are analogous to the strata in a stratification of an analytic set [Whi65]. Formally, a combinatorial stratification of a cell 2-complex \( \Gamma \) is a collection of connected combinatorial manifolds \( \Gamma_i \) such that their union covers \( \Gamma \), and the intersection between two manifolds \( \Gamma_i \) and \( \Gamma_j \) is either empty, or a subcomplex of both \( \Gamma_i \) and \( \Gamma_j \). A combinatorial stratification is not necessarily unique: for instance, two valid stratifications of a cell 2-complex are shown in Figure 3.9.

![Figure 3.9: Two valid stratifications for a (a) cell-complex formed by a top edge and three triangles sharing a non-manifold edge. Resulting manifold components in (b) are different from the manifold components in (c).](image-url)
Resulting set of strata (and their connectivity) provide a basis for the Handle-Cell (HC) data structure. The HC data structure represents two types of cells, namely global cells and local cells. Global cells belong to the given cell complex, and, thus, we can recognize global vertices, global edges, and global faces. Local cells belong to the strata, namely points, curves, and surfaces. Curves are manifold cell 1-complexes, while surfaces are manifold cell 2-complexes, represented through the HE data structure, described in Section 3.2.1.1. Connectivity information among strata is captured through global vertices and edges.

As proposed in [DFH05], the HC data structure encodes following topological data structures:

- for each vertex \( v \), co-boundary relation \( R_{0,1}(v) \), which consists of all the edges incident at \( v \);
- for each edge \( e \):  
  - boundary relation \( R_{1,0}(e) \), which consists of two extreme vertices of \( e \);
  - partial adjacency relation \( R_{1,1}(e) \), which consists of all the edges adjacent to \( e \) belonging to 2-manifold strata: this relation is ordered, so that both the \( (2i) \)-th and \( (2i + 1) \)-th elements are adjacent along the \( i \)-th vertex of \( e \);
  - co-boundary relation \( R_{1,2}(e) \), which consists of all the faces in \( St(e) \);
- for each face \( f \), partial boundary relation \( R_{2,1}(f) \), which consists of one arbitrarily selected edge belonging to the boundary of \( f \);

The HC data structure supports efficient topological navigation: for instance, incidence relations among \((q - 1)\)-cells and \( q \)-cells are fully encoded, and edge-based adjacency relations among edges in the strata are directly encoded.

The HC data structure is quite similar to the RE and PE data structures, described in Sections 3.2.2 and 3.2.3, respectively, and, it is also closely related to the HF data structure, discussed in Section 3.3.2. In any case, the HC data structure is more expressive than these data structures, because it directly exposes the connectivity among the strata, and provides any information about the structure of the input shape. In other words, it provides a structural representation of a non-manifold shape.

### 3.4.3 The Initial Quasi-Manifold Decomposition

In this section, we describe the Initial Quasi-Manifold (IQM) Decomposition and its variants [DFMMP03, HVDF06], suitable to represent abstract simplicial complexes of arbitrary dimension. We follow the same definition proposed in [DFH05].

In [DFMMP03] the authors define the Initial Quasi-Manifold (IQM) Decomposition, a decomposition of abstract simplicial \( d \)-complexes of arbitrary dimension. Components of this decomposition, known as the Initial Quasi-Manifolds (IQM) components, are regular \( k \)-complexes, with \( k \leq d \), such that any pair of \( k \)-simplices in the star of a vertex are connected through a \((k - 1)\)-path,
where each pair of consecutive $k$-simplices in this path shares a $(k-1)$-simplex. As demonstrated in [HDF07a] and also in Section 8.1, the class of the IQM components coincides with the class of manifolds up to dimension two. Generally speaking, an IQM component is not always a manifold, and not even a pseudo-manifold in three or higher dimensions. We conjecture that, if an IQM component is embedded in any Euclidean space $\mathbb{E}^d$, then it must be a pseudo-manifold $d$-complex. The IQM Decomposition is computed by splitting the star of each non-manifold singularity into manifold components, as described in [DFMMP03]. Hence, the IQM Decomposition is unique, since it does not make any arbitrary choice in deciding where the object has to be cut, and removes singularities by splitting the complex only at non-manifold simplices.

Connections among IQM components are described through vertices bounding singularities, which are shared by several IQM components. A split simplex $\sigma$ is duplicated in each IQM component incident at $\sigma$. The IQM Decomposition of an arbitrary shape is represented through a hyper-graph, known as the *IQM Decomposition Graph*, in which each node corresponds to one IQM component, while each hyper-arc connects all the IQM components incident at vertices of a split simplex $\sigma$. For instance, vertex $v_1$ in Figure 3.10(a) is replicated, respectively, into split simplices $w$, $w'$, and $w''$ in IQM components $C_1$, $C_2$, and $C_3$, which are shown in Figure 3.10(b). Vertex $v_2$ is replicated into split simplices $v$, $v'$, $v''$, and $z$ in IQM components $C_1$, $C_2$, $C_3$, and $C_4$, respectively. In the IQM Decomposition Graph, shown in Figure 3.10(c), there exists a hyper-arc connecting IQM components $C_1$, $C_2$, and $C_3$ through three copies of vertex $v_1$, and a hyper-arc connecting IQM components $C_1$, $C_2$, $C_3$, and $C_4$ through four copies of vertex $v_2$.

![Figure 3.10](image)

Figure 3.10: (a) An arbitrary simplicial 2-complex, (b) the related IQM components, and (c) the IQM Decomposition Graph.

Following [DFH05], the *Double-Level Decomposition (DLD)* data structure [HVDF06] is a representation of a non-manifold simplicial 3D shape, based on the IQM Decomposition Graph. In this context, there are IQM components of dimension up to three. In the DLD data structure, each IQM component is described through an independent EIA data structure, discussed in Section 3.1.3. A $k$-dimensional IQM component, with $k \leq 3$, can be effectively described by the EIA data structure (see Section 3.1.3), since the star of a vertex, restricted to an IQM component, is efficiently traversed, with $h \leq 3$, through relations $R_{0,h}^*$, $R_{h,h}$, and $R_{h,0}$. Thus, the DLD data structure encodes only vertices and top simplices, plus adjacency relation among top simplices, as provided by the EIA data structure. As a consequence, the DLD data structure supports both a
vertex-based, and an edge-based traversal among IQM components connected through the same hyper-arc. For instance, given a simplex $\sigma$, and a simplex copy $\sigma_i$ of $\sigma$, in an IQM component $C_i$, we can follow references to its hyper-arc, and find copies $\sigma_j$ connected with $\sigma$, and IQM components incident at $\sigma$.

The IQM Decomposition approach of the DLD data structure differs from the stratification approach exploited in the HC data structure, discussed in Section 3.4.2. First, the HC data structure is based on a decomposition of cell 2-complexes, while the IQM Decomposition is dimension-independent and based on simplicial complexes. Also, the IQM Decomposition is unique. The DLD data structure provides an effective representation of arbitrary simplicial 3-complexes. As demonstrated experimentally in [HVD06], the storage costs of the DLD data structure and of the NMIA data structure [DFH03], discussed in Section 5.3, are comparable.

One of our main contributions, namely the Generalized Indexed Data Structure with Adjacencies (IA*) [CDFW11], which we introduce in Section 5.1, is a dimension-independent representation of arbitrary simplicial complexes. In [CDFW11] and in Section 5.4.2, we prove that the IA* data structure, restricted to simplicial 3-complexes, is more compact than the NMIA data structure, and, thus, also than the DLD data structure. In any case, the DLD data structure is more expressive than the NMIA and IA* data structures, since it explicitly exposes singularities and connections among IQM components. Finally, the DLD, NMIA, and IA* data structures reduce to the EIA data structure, discussed in Section 3.1.3, if restricted to manifold simplicial 3-complexes.

### 3.5 Editing Operators

In this section, we briefly review several editing operators, which modify topological and combinatorial properties of cell and simplicial complexes. Following [DFH05], editing operators aim to reduce complexity of densely sampled representations, or to improve shape quality [BEG94]. Generally speaking, a shape is simplified by applying a sequence of decimation updates, and this process is known as the simplification, while the reverse process is said to be the refinement. In any case, obtaining the optimal simplification, namely the simplification which applies the minimum number of decimation updates, is known to be a NP-hard problem [AS94]. As a consequence, several heuristic methods have been developed in the literature in order to simplify a shape.

In the remainder of this section, we concentrate our attention on editing operators, which can be applied on non-manifold shapes, discretized by cell and simplicial complexes. Broadly speaking, there is a relevant literature about editing operators, but it is quite disorganized. Here, we follow the classification of editing operators, discussed in [CDF12].

In Section 3.5.1, we briefly review editing operators for cell complexes, while in Section 3.5.2, we review editing operators specific for simplicial complexes. In Section 3.5.3, we briefly analyze discuss connections between editing operators and multiresolution modeling. In Section 3.5.4, we briefly discuss several editing operators required for preparing a CAD model to numerical
simulations. Finally, in Section 3.5.5, we briefly review the so-called *out-of-core representations*, whose size exceeds the RAM amount in a workstation.

### 3.5.1 Editing Operators on Cell Complexes

In this section, we briefly review several editing operators specific for cell complexes. We concentrate our attention on the *Euler operators* [Man87, Hof89, Ago05], and the *Handle-body operators* [Sma62, LT97, LPT+03]. Most of this section is based on [CDF12].

The first editing operators developed in the literature have been defined in [Man87] on manifold cell 2-complexes, which discretize the boundary of a shape. These operators are known as the *Euler operators*, and satisfy the *Euler-Poincaré formula* [Hof89, Ago05]. It can be proven that each editing operator can be expressed as a sequence of Euler operators [Man87]. Several variants of the Euler operators have been defined, and, thus, many variants of the Euler-Poincaré formula have been designed. Euler operators specific for manifold cell 2-complexes [Bau75, Wil85], for general cell 2-complexes [EW79, ADFF85, Man87, LL01], for general cell 3-complexes [MSNK89, Mas93], for non-manifold complexes [Wei88a, LL91, Hei91, MG95, HG01, SG05], and for stratified sets [SV91, GMR99, Gom04], have been developed. An interested reader may refer to [CDF12] for a complete discussion about Euler operators on cell complexes.

Another interesting class of editing operators for cell complexes is given by the *Handle-body operators*, which are based on the *Handle-body theory* [Sma62, LT97, LPT+03]. This theory studies topological modifications, which are generated by attaching handles to a manifold complex with boundary. Handle-body operators also modify the topological properties of the input complex. In [LT97] the authors observe that a compact orientable 3-manifold can be iteratively built by attaching $k$-handles, with $0 \leq k \leq 3$, to a 3-ball. These operators are known as the *Morse operators*. In [LPT+03] the authors prove that a surface can be obtained from a 2-ball by iteratively attaching $j$-handles, with $0 \leq j \leq 2$.

### 3.5.2 Editing Operators on Simplicial Complexes

Generally speaking, Euler operators are not closed on simplicial complexes, since their application may not produce simplicial complexes as result. Several editing operators, specific for simplicial complexes, have been proposed in the literature. A complete review of these techniques is beyond the scope of this document. In this section, we briefly discuss editing operators specific for simplicial complexes. An interested reader can refer to [Gar99, LRC+02, DDFM+06, CDF12] for more information.

Following [CDF12], a class of interesting operators on simplicial complexes consists of *Stellar operators*, introduced in [Lic99, Vel03]. These operators are defined in arbitrary dimension, and modify the neighborhood of a simplex without affecting topological properties of a simplicial complex. In [Vel03] three operators, specific for simplicial 2-complexes, have been defined, namely
the Split, Weld, and Flip operators. Split operators are refinement operators, while Weld operators are simplification operators. Split and Weld operators are mutually inverse.

Flip operators change only the connectivity in the simplicial complex without altering the number of simplices, and they are expressed in terms of Split and Weld operators. There are two instances of the Split operator, namely the Face Split and Edge Split operators. Face Split operator inserts a new vertex \( p \) inside a triangle \( t \), and connects it to the three vertices of \( t \). Edge Split operator introduces a new vertex \( p \) on an existing edge \( e \) splitting \( e \) in two edges. Furthermore, it splits also two triangles in \( St(e) \) by connecting the new vertex \( p \) to vertices of two triangles not in \( St(e) \).

Weld operators, namely the Vertex Weld and Edge Weld operators, are the inverse ones of the split operators. Vertex Weld operator is the inverse of the Vertex Split operator. In any case, it is applied on a vertex \( v \) of degree three in a simplicial complex. It deletes \( v \), three edges in \( St(v) \), and it merges three triangles in \( St(v) \) in one triangle. Edge Weld operator is applied on a vertex \( v \) of degree four, and deletes \( v \) and two non-consecutive edges in \( St(v) \). Then, it merges two pairs of triangles in \( St(v) \).

Edge Flip operator deletes one edge \( e \), and replaces it by another edge, which connects two vertices in \( Lk(e) \).

In the literature, there are several techniques for updating simplicial complexes, like those described in [Gar99, DFM02, LRC+02, DDFM+06], for simplifying simplicial 2-complexes and 3-complexes. The most interesting editing operators are the Edge-Collapse, Vertex-Split, and Vertex-Pair Collapse (VPC) [PH97] operators.

Edge-Collapse operator is a simplification operator and contracts an edge to a vertex. If we must contract an edge \( e = (v', v'') \) to a vertex \( v \), which does not coincide with any endpoint of \( e \), then we apply the Full-Edge Collapse operator on \( e \). This operator modifies simplices in \( St(v') \) and \( St(v'') \). In this case, \( k \)-simplices in \( St(e) \) become \((k-1)\)-simplices in \( St(v) \), while simplices incident at either vertex \( v' \) or \( v'' \) become incident at \( v \). If we must contract an edge \( e = (v, w) \) to a vertex \( w \), then we apply the Half-Edge Collapse operator on \( e \). In this case, \( k \)-simplices in \( St(e) \) become \((k-1)\)-simplices in \( St(w) \), while other simplices in \( St(v) \) becomes incident at \( w \).

Vertex-Split operator is the inverse operator of Edge Collapse operator and it expands a vertex into a new edge. Full-Vertex Split operator expands a vertex \( v \) into two new vertices \( v' \) and \( v'' \) and a new edge \( e = (v', v'') \). In other words, it is the reverse operator of the Full-Edge Collapse operator, as shown in Figure 3.11(a). Half-Vertex Split operator creates a new vertex \( v \), and a new edge \( e = (v, w) \). In other words, it is the reverse operator of the Half-Edge Collapse operator, as shown in Figure 3.11(b).

Given two vertices \( v_1 \) and \( v_2 \) in a simplicial \( d \)-complex \( \Sigma \), and, a new vertex \( v \), not necessarily in \( \Sigma \), the Vertex-Pair Collapse (VPC) operator [PH97] contracts two vertices \( v_1 \) and \( v_2 \) to a new vertex \( v \). This operation is often denoted as \((v_1, v_2) \rightarrow v \). As a consequence, all the simplices incident either at \( v_1 \) or at \( v_2 \) become incident at \( v \). Usually, this operator does not change the number of simplices in the complex, but it updates simplices incident at the collapsed vertices. As a result of
this operator, we modify topology of the input simplicial complex. The VPC operator may merge two components, if vertices $v_1$ and $v_2$ belong to different pieces of the input complex, or it may close or create holes, especially if vertices $v_1$ and $v_2$ belong to the same component. Figure 3.12 shows the execution of the VPC operator for a pair of vertices $v_1$ and $v_2$ in a simplicial 2-complex.

In [DFMPS04, DFH04] the authors discuss how the VPC operator can be applied on the TS and NMIA data structures, respectively.

### 3.5.3 Editing Operators and Multiresolution Modeling

In this section, we briefly discuss connections between editing operators and multiresolution modeling, following the approach proposed in [DFPM97, Pup98, Mag99].

A critical aspect in the interactive graphics is the *resolution* of the input shape, which can be expressed in terms of density of cells or simplices. A multiresolution model provides representations...
of a shape at different resolutions. Following [DFPM97], a multiresolution model can be seen as a base representation $\Sigma_0$ at coarse resolution, plus a sequence of local updates, which are applied to $\Sigma_0$ in order to refine shapes at variable resolution. For the purpose of defining a multiresolution model, we are interested in minimal updates, which cannot be split into two or more valid modifications to be performed in a sequence.

Following [DFPM97], a sequence $U = (u_i)_{i=0}^m$ of updates is valid for a simplicial complex $\Sigma$ if and only if each update $u_i$ is valid for the simplicial complex obtained as the result of successively applying all the updates preceding $u_i$ in the sequence.

An update $u_i$ directly depends on another update $u_j$, with $i \neq j$, (and, thus, $u_j$ directly blocks $u_i$) if and only if the effect of applying $u_i$ is removing some of the simplices introduced by applying $u_j$.

In a sequence of updates, two updates depend on each other if they are in the transitive closure of the direct dependency relation.

In a valid sequence, a simplex $\sigma$ may be created and removed several times and this creates cycles in the relation of dependency. Thus, an update is non-redundant with respect to a set of updates if it does not recreate simplices removed by some other update in the set. It is easy to verify that, in non-redundant and valid sequences, each simplex either is in the initial complex, or there is exactly one modification specifying its “creation”. Moreover, each simplex is either in the final complex, or there is exactly one modification specifying its “deletion”.

Even if two modifications are independent, they cannot necessarily be applied to the same simplicial complex without interfering. Two valid updates $u_i = (u_i^-, u_i^+)$ and $u_j = (u_j^-, u_j^+)$ are conflict-free when all the pairs formed by $u_i^-$ and $u_j^-$, $u_i^-$ and $u_j^+$, $u_i^+$ and $u_j^-$, $u_i^+$ and $u_j^+$, intersect at most in a subset of their boundaries, which are preserved in the updates $u_i$ and $u_j$. A sequence $(u_i)_{i=0}^m$ of updates on $\Sigma$ is conflict-free if and only if there are not any conflicts between all the pairs of independent updates.

The Multi-Tessellation (MT) model [DFPM97] is a variable-resolution continuous multiresolution model which can represent multiresolution simplicial regular shapes in arbitrary dimensions. The MT model is defined by abstracting over the relation of direct dependency from a valid, non-redundant, and conflict-free sequence $U$ of minimal refinements. The non-redundancy of the sequence ensures that the direct dependency relation is a partial order. The validity and absence of conflicts ensure that, for every subset of updates closed with respect to the dependency relation, the result of applying its elements, sorted in any total order consistent with the partial one, is a simplicial complex [DFPM97].

Let $\Sigma_0$ be a simplicial complex, called base representation, at coarse resolution, and let $U$ be a valid, non-redundant, and conflict-free sequence of minimal updates. Following [DFPM97], a Multi-Tessellation (MT) is a tuple $\mathcal{M} = (U, \prec)$, where $\prec$ denotes the direct dependency relation. Clearly, this dependency relation can be represented with Directed Acyclic Graph (DAG). Given two updates $u_i = (u_i^-, u_i^+)$ and $u_j = (u_j^-, u_j^+)$, there is an arc $(u_i, u_j)$ in the DAG if and only if $u_i$ directly depends on $u_j$. 50
Figure 3.13(a) and Figure 3.13(b) show, respectively, a valid, non-redundant, and conflict-free sequence U of minimal updates, and the corresponding DAG.

Given a Multi-Tessellation $\mathcal{M}$, and a valid sequence $U' = (u'_i)_{i=0}^m$ of updates applied to $\Sigma_0$, the front simplicial complex of $\mathcal{M}$ can be defined as $\mathcal{F} = \Sigma_0[u_1] \cdots \Sigma_0[u_m]$. In other words, it is the simplicial complex obtained after the application of all the updates in $U$.

Figure 3.13(c) show the front simplicial complex associated to the front of the MT in Figure 3.13(b), which is represented by a bold red line.

Following [Mag99], the front simplicial complex is a regular simplicial complex, and it is uniquely defined because it does not depend on the specific sequence considered. A sub-MT $\mathcal{M}'$ of a Multi-Tessellation $\mathcal{M}$ identifies a subset of the modifications of $\mathcal{M}$ which contains the base representation $\Sigma_0$, and it is is closed with respect to the dependency relation. Simplicial complexes at intermediate resolutions are front complexes of sub-MTs.

These techniques have become relevant, and several multi-resolution models have been developed [DFMPS04, DDFM+06]. Specifically, in [DFMPS04] the authors define an extension of the MT model, known as the Non-Manifold Multi-Tessellation (NMT) model, which provides a multiresolution model for non-manifold simplicial shapes, currently restricted to arbitrary simplicial 2-complexes.

### 3.5.4 Editing Operators for the Idealization Process

Physics-based simulations operate on Computer-Aided Design (CAD) data, namely boundary representations, and their computational performances depend on the number of geometric features [WSO03]. Hence, a geometric model needs to be accurately prepared for the Finite Element
Methods (FEM) [TCMT56] in order to perform numerical and mechanical simulations. Furthermore, it is mandatory to take into account several mechanical hypotheses for a given product, expressed as constraints to be satisfied, like boundary conditions. This process, known as the *idealization*, involves several types of geometrical and topological operators [VL97, VL98, VL01, CDMM04, LF05, TBG09]. A complete review of these methods is beyond the scope of this research. In this section, we provide only a brief review of a subset of idealization operators, which are more interesting in the context of our research. Specifically, we follow the classification discussed in [TBG09].

A first class of idealization operators consists of removing elements, which do not satisfy several requirements for the FEM generation. They apply classical editing operators to the input model, like the face-based clustering, or vertex decimation. In these approaches, the validity of locally modified representations is ensured by imposing constraints on the resulting topology. In [FRL00] the authors introduce a complete set of idealization operators, based on vertices decimation. In this context, there are several methods with these properties [TBG09].

The *Mesh Constraint Topology (MCT)* [FCFL08] is an interesting technique in the context of our research, since it provides a structural representation of the input model. An MCT model is formed by two types of entities, known as the *MCT-faces* and *MCT-edges*. MCT-faces and MCT-edges are, respectively, poly-surfaces and poly-edges, and they can be seen as the union of Riemannian surfaces and curves, which define the reference model. Thus, MCT entities preserve exact geometries of the input model, since they provide a high-level representation of entities in the reference CAD model. An MCT model is represented through three hyper-graphs based on topological relations Face-Edge, Face-Vertex, and Edge-Vertex (namely boundary relations $R_{2,1}$, $R_{2,0}$, and $R_{1,0}$). A set of graph-based operators is defined on MCT models, namely for adding and removing MCT-edges and MCT-vertices, collapsing an MCT-edge to an MCT-vertex, and merging MCT-vertices. These operators are used together with several constraints, which are based on the size and curvature of MCT-entities, for improving quality of models. The size of an MCT-face is defined as the distance among face boundaries, while the size of an MCT-edge is defined as its length. The curvature is leading to the deviation angle, namely the angle between normals of adjacent model segments, for a given discretization error. The size of the MCT entities must be greater than a size threshold, while the curvature must be smaller than a curvature threshold in order to satisfy several quality constraints.

Another interesting class of idealization operators is based on the recognition of *topological features* before simplifying the input model [Lee05, FMLG09, LDFH09, TBG09]. These techniques are organized in three steps. In the first step, features are recognized and classified in the input model through several techniques. Then, these features are removed from the input model, keeping information about their placement. In the second step, the current model is further simplified by using several idealization operators. In the third step, features are reinserted in the resulting model, if they satisfy several quality constraints.

For instance, in [Lee05], the system supports a feature-based multiresolution model. In this con-
text, it is possible to define features at different resolutions, which are rearranged in the third step. This rearrangement of features may result in different final shapes owing to the non-commutative nature of simplification operators. At the end, the extracted idealization features are rearranged, according to application-dependent multiresolution criteria.

We can also recognize techniques based on the *dimension reduction*, which modify the dimension of a component. As observed in [BCMA+11], parts presenting a beam behavior may be replaced with 1-dimensional entities, while parts presenting a plate behavior may be replaced by 2-dimensional surfaces. Hence, there may be a negligible effect on accuracy of the numerical results, but the computational time may reduce dramatically. In any case, errors are controlled by introducing appropriate constraints. One of the well established techniques for reducing dimension of a shape is based on the *Medial Axis Transform (MAT)* [ABE04]. In [DAP00, SFM05] the authors exploit the MAT of a model to reduce its dimension. To determine whether the dimension of the object is to be reduced, the aspect ratio and the taper criteria are used. The lower bound for the aspect ratio is determined as the ratio of the length of the shortest edge bounding a region, and the maximum disk diameter in a local region. The taper is determined as the maximum rate of change of diameter with respect to medial edge length. A high aspect ratio or a low threshold indicates variation in slenderness property of the object. If a region has an aspect ratio greater than the threshold, or a taper value lower than the taper threshold, then it is suitable for modeling with 1-dimensional element.

It is clear that the idealization process is not a simple task to be resolved, because it needs to integrate different types of information, regarding geometric, topological, combinatorial, structural, and mechanical properties of input models. Hence, it is mandatory to exploit representations which allow combining several descriptions of the input shape, like the *Mixed Shape Representation* [LFG08]. It supports both CAD and polyhedral representations of arbitrary models, formed by components of different dimensionality. This representation is defined on the *High Level Topology (HLT)* [Ham06], which provides a low-level description of a model as collections of cells of dimension up to three, known as the HTL-entities. We recognize HTL-vertices (0-cells), HTL-edges (1-cells), and HTL-faces (2-cells). In addition, some entities logically group the low-level HTL-entities, namely the HTL-loops, HTL-shells, HTL-regions, and HTL-body. Broadly speaking, the HTL-entities are equivalent to the entities encoded in the RE and PE data structures, discussed in Sections 3.2.2 and 3.2.3, respectively.

### 3.5.5 Out-of-core Representations

Nowadays, performances of graphics subsystems has enormously improved, but unfortunately the complexity of graphics applications has also increased. Some huge representations can be easily produced in several applications, for example by the 3D scanning of real objects, or in medical applications [ABA06]. In order to improve the accuracy of a representation, we need an accurate object sampling, and, thus, the size of a representation increases, and it often exceeds the amount of RAM in a workstation. Hence, these representations introduce severe overheads, and their
management has often prohibitive costs, even for high-performance workstations. Moreover, it is important to encode them in the most efficient way as possible, maintaining the opportunity to execute operations. Increasing the RAM size may be a trivial solution, since its cost is going sharply down. However, it is difficult to establish the correct amount of RAM, since an arbitrary amount of space is required. Simplification techniques are a reasonable solution for this problem, since they reduce the models size through local updates. Unfortunately, also these techniques may require an high RAM amount.

Thus, an out-of-core technique is mandatory. In this context, we maintain the entire shape in External Memory (EM), and then we dynamically load in RAM only portions of interest, small enough to be processed in-core. In this way, we directly operate on each portion, by removing any limitation on the input model size. In any case, an EM access is slower than a RAM access: if an efficient control of EM accesses is not performed, then this fact is a bottleneck, and it degrades performance. In the literature, several EM techniques have been designed and proposed: for instance, we recall several techniques regarding the EM visualization [SCESL02, DC10] and simplification [Hop98, Pri00, CMRS03, DDFPS07, VCL+07, SK11].

There is a lot of research about these techniques, and a complete review is beyond the scope of this thesis. An interested reader can refer to [Can09, DC10] for more information. Specifically, we are interested in EM simplification operators based on the spatial indexing techniques [Sam06], since they provide an effective EM representation for simplicial shapes. Generally speaking, these methods subdivide any simplicial shape into hierarchical patches, small enough to be simplified in-core through a conventional simplification technique, like those discussed in Section 3.5.2.

One of the first algorithms [Hop98] has been designed for the representation of terrain models. The input terrain is hierarchically divided in blocks, and each block is simplified, by collapsing edges not incident at the boundary of a block. Once each block has been simplified, this algorithm traverses bottom-up the hierarchical structure by merging sibling cells and again simplifying, until the resulting terrain is contained in few blocks (one or two at most), small enough to be processed in-core. In this approach, a complete bottom-up structure traversal is required to remove elements incident at the inter-blocks boundaries, where intermediate results present unpleasant elements at the original resolution. In [Pri00] this technique has been generalized to arbitrary shapes, but with the same disadvantages.

In [CMRS03] the authors propose a simplification method based on the Octree-based External Memory Mesh (OEMM) data structure. Here, a shape is subdivided through an octree [Ore82, Sam06], and the subdivision process stops, when each node fits in a disk page. Each portion of the input model, stored in each octree leaf, is independently simplified through iterative edge collapses. Once simplified, leaves are merged and simplified further, similarly as in [Hop98]. The elements spanning adjacent cells are identified through a tagging strategy during the subdivision phase, performing octree-based regular splits.

In [Can10] we propose the Objects Management in Secondary Memory (OMSM) framework, a general-purpose framework for managing an unstructured and huge set of spatial objects. In the
OMSM framework, a generic storage architecture is described in terms of three aspects, namely a space partitioning tree, the clustering policy of nodes in a spatial index, grouped in atomic units, known as clusters [DSS06]. Finally, a cluster is transferred between a storage support and RAM. Each of these aspects does not depend on each other, and is described by a dynamic plugin, including the represented objects. Hence, the OMSM framework may be easily adapted to the users needs through dynamic plugins, providing many techniques to be integrated in a storing architecture. A new technique is made available in the OMSM framework without modify the framework.

Recently, in [WFDFV11] the authors propose the PR-star Octree, a topological and spatial data structure, which efficiently supports topological queries on simplicial 3-complexes. It extends the Point Region octree (PR-Octree) [Sam06], storing the list of 3-simplices in the star of each vertex. Thus, each leaf node encodes the minimal amount of information needed to locally reconstruct the topological connectivity of its indexed 3-simplices. The PR-star Octree can be exploited in several applications, including detection of the domain boundaries, computation of 3D local curvature, and simplification of the input 3-complex.
In this chapter, we introduce the Incidence Simplicial (IS) data structure [DFHPC10], one of our contributions in the context of topological data structures. The IS data structure is a dimension-independent and incidence-based data structure for representing abstract simplicial complexes. The IS data structure encodes all the simplices in a non-manifold simplicial shape, and a subset of incidence relations. As a consequence, it is suitable for those applications, like the FEM analysis and numerical simulations, where one needs to attach attributes to all the simplices in the input simplicial shape.

In Section 4.1, we propose the complete design of the IS data structure, and introduce a graph-based representation for the IS data structure. We evaluate its storage cost, and provide algorithms for retrieving all the topological relations from the IS data structure.

Following [DFH05], the most common dimension-independent data structures for cell and simplicial complexes are the incidence-based representations. Recall that an incidence-based data structure encodes all the simplices in a complex, plus a subset of incidence relations. The Incidence Graph (IG) [Ede87] is one of the most common dimension-independent incidence-based data structures for abstract cell complexes. In Section 4.2, we discuss a restriction of the IG data structure to abstract simplicial complexes, introduced in [DF03].

However, the IG data structure is verbose and exhibits a very large overhead, if restricted to manifolds. Moreover, it does not allow detecting non-manifold singularities efficiently. Thus, we need more compact representations, from which it is efficient to detect non-manifold singularities. The IS data structure is a simplicial variant of the IG data structure for abstract simplicial complexes of any dimension, and overcomes several limitations of the Incidence Graph.

The Simplified Incidence Graph (SIG) [DFGH04] is another simplicial variant of the IG data structure for representing abstract simplicial complexes, which we discuss in Section 4.3. The SIG
data structure overcome several limitations of the IG data structure, but its storage cost may be slightly expensive.

Finally, in Section 4.4, we perform some experimental comparisons for the incidence-based data structures, discussed in this chapter, in terms of their storage cost and encoded relations. Our tests show that the IS data structure exhibits a small overhead, if restricted to manifolds. Moreover, it is always more compact than the IG representation, and it is as compact as the SIG data structure for non-manifold simplicial 2D shapes, and more compact than the SIG representation for non-manifold simplicial 3D shapes.

In Chapter 6, we propose an implementation of the IS data structure in our Mangrove Topological Data Structure (Mangrove TDS) framework. We provide a pseudo-code description of algorithms used for retrieving topological relations, and for constructing the IS data structure. Furthermore, in Chapter 7, we propose a quantitative analysis among performances of several data structures, including the IS, IG, and SIG representations, regarding the efficiency of topological relations.

4.1 The Incidence Simplicial (IS) Data Structure

In this section, we introduce the Incidence Simplicial (IS) data structure [DFHPC10], one of our contributions in the context of topological data structures. The IS data structure is an explicit, dimension-independent, and incidence-based data structure for representing abstract simplicial complexes, not necessarily embedded in any Euclidean space.

In the following, we propose a complete analysis of the IS data structure. In Section 4.1.1, we discuss the complete design of the IS data structure, and define a graph-based representation for the Incidence Simplicial data structure, which we call the IS-graph. In Section 4.1.2, we illustrate the basic ideas for implementing the IS data structure, and we evaluate its storage cost. Finally, in Section 4.1.3, we describe algorithms for retrieving all the topological relations from the IS data structure.

4.1.1 Design of the Data Structure

In this section, we discuss the complete design of the IS data structure [DFHPC10], and define a graph-based representation for the Incidence Simplicial data structure, which we call the IS-graph.

Broadly speaking, the IS data structure is a simplicial variant of the IG data structure, which we describe in Section 4.2. The IS data structure encodes, for a given $p$-simplex $\sigma$, with $0 < p \leq d$, $p+1$ simplices of dimension $p-1$, which bounds $\sigma$. These simplices form boundary relation $R_{p,p-1}$ for simplex $\sigma$, which is also encoded in the IG data structure.

Furthermore, the IS data structure is based on a decomposition of the star of a simplex $\sigma$, induced by the connected components in $Lk(\sigma)$. Given any $p$-simplex $\sigma$ in $\Sigma$, with $0 \leq p < d$, each connected component $\Sigma^*_{\sigma}$ in $Lk(\sigma)$ corresponds to several top simplices in $St(\sigma)$, which are bounded
by simplices in \( \Sigma^*_\sigma \). Thus, each connected component \( \Sigma^*_\sigma \) can be represented by one \((p + 1)\)-simplex \( \sigma' \) (arbitrarily selected), which belongs to the boundary of a top simplex, bounded by simplices in \( \Sigma^*_\sigma \). Simplex \( \sigma' \) is known as \textit{representative simplex} of \( \Sigma^*_\sigma \). For any \( p \)-simplex \( \sigma \), we denote the set of representative \((p + 1)\)-simplices of each connected component in \( Lk(\sigma) \) as partial co-boundary relation \( R^*_{p,p+1}(\sigma) \).

Figure 4.1 shows relations between the link (in bold lines) and star of a vertex \( v \) in a simplicial 3-complex. Here, there are two connected components in the link of \( v \), formed, respectively, by a vertex \( v' \) and a 0-connected component, consisting of a triangle \( f_t \) and an edge \( e_f \). In the star of \( v \), these components correspond, respectively, to a top edge \( w \), and a 1-connected component, formed by a tetrahedron \( t \) and a triangle \( f \). This latter component can be represented by edge \( e \). Thus, \( R^*_{0,1}(v) = \{w, e\} \).

Figure 4.1: Relations between the link and star of a vertex \( v \) in a simplicial 3-complex. Here, there are two connected components in \( Lk(v) \), formed, respectively, by a vertex \( v' \) and a 0-connected component, which consists of a triangle \( f_t \) and an edge \( e_f \). In \( St(v) \), these components correspond, respectively, to a top edge \( w \), and a 1-connected component, which is formed by a tetrahedron \( t \) and a triangle \( f \). This latter component can be represented by edge \( e \). Thus, \( R^*_{0,1}(v) = \{w, e\} \).

Formally, the IS data structure encodes all the simplices in a simplicial \( d \)-complex \( \Sigma \), and, for each \( p \)-simplex \( \sigma \) in \( \Sigma \), boundary relation \( R_{p,p-1}(\sigma) \), with \( 0 < p \leq d \), and partial co-boundary relation \( R^*_{p,p+1}(\sigma) \), with \( 0 \leq p < d \), as defined above.

The IS data structure can be described through a graph-based representation \( G = (N, A) \), which we call the \textit{IS-graph}. Each node in \( N \) corresponds to a simplex in \( \Sigma \), while an arc in \( A \) corresponds to an incidence relation for the simplices described by nodes in the IS-graph. An arc \((\sigma, \sigma')\) is said to be a \textit{boundary arc}, if it connects nodes related to a \( p \)-simplex \( \sigma \), with \( 0 < p \leq d \), and a \((p - 1)\)-simplex \( \sigma' \) in \( \Sigma \), such that \( \sigma' \in R_{p,p-1}(\sigma) \). An arc \((\sigma, \sigma')\) is said to be a \textit{co-boundary arc}, if it connects nodes related to a \( p \)-simplex \( \sigma \), with \( 0 \leq p < d \), and a \((p + 1)\)-simplex \( \sigma' \) in \( \Sigma \), such that \( \sigma' \in R^*_{p,p+1}(\sigma) \). We define the IS \textit{boundary graph} as the spanning subgraph of \( G \), which contains all the nodes and boundary arcs. Finally, we define the IS \textit{co-boundary graph} as the spanning subgraph of \( G \), which contains containing all the nodes and co-boundary arcs.

Figure 4.2(a) shows a simplicial 3-complex formed by two tetrahedra and a top triangle: here, each simplex is expressed in terms of its vertices, and the corresponding IS-graph is also represented. For the sake of clarity, Figure 4.2(b) shows the IS boundary graph, while Figure 4.2(c) shows the IS co-boundary graph of the input simplicial 3-complex. In Figure 4.2(b), node in red represents tetrahedron \( t_1 = (1, 2, 3, 4) \), nodes in green and in blue represent faces and edges bounding \( t_1 \), re-
spectively. Arcs in red denote boundary relation $R_{3,2}(t_1)$, arcs in green denote boundary relations $R_{2,1}$ for triangles bounding $t_1$, and, finally, arcs in blue denote the boundary relations $R_{1,0}$ for edges bounding $t_1$.

![Diagram](image)

Figure 4.2: (a) A simplicial 3-complex formed by two tetrahedra and a top triangle. For the sake of clarity, (b) the IS boundary graph and (c) IS co-boundary graph are separately shown.

In Figure 4.2(c), nodes in red, which correspond to edges (1, 4) and (1, 5), have more than one outgoing arcs, thus their link is formed by more than one connected component. Generally speaking, if a simplex $\sigma$ is manifold, then $Lk(\sigma)$ is formed by only one connected component. The reverse is not true: even if $Lk(\sigma)$ may have one connected component, $\sigma$ might be a non-manifold simplex.
For instance, the vertex 1 in Figure 4.2(a) is a non-manifold vertex, still its link is formed by only one connected component. In Chapter 6 (see Section 6.2.6), we exploit these properties of the IS data structure for recognizing non-manifold singularities.

It can be easily proven that the IS-graph is a subgraph of the IG-graph, which we introduce in Section 4.2. The IG-graph is the graph-based representation of the IG data structure. These graphs are defined on the same collection of nodes, and encode the same boundary relations. Furthermore, a co-boundary arc in the IS-graph is surely a co-boundary arc in the IG-graph, while the reverse is not true. In fact, partial co-boundary relation $R^*_{p,p+1}$ and boundary relation $R_{p,p-1}$ are not symmetric, like in the IG data structure (see Section 4.2).

### 4.1.2 Implementation and Storage Cost

In this section, we illustrate the basic ideas for implementing the IS data structure, and propose a precise evaluation of its storage cost.

In order to represent a simplicial $d$-complex $\Sigma$ through the IS data structure, we exploit the same approach described in [AJ05]. Here, we encode a topological relation $R$ as a table with one record for each simplex involved in $R$. We assign a unique index to each simplex $\sigma$ in $\Sigma$, namely an integer value to be used for accessing records related to $\sigma$ in each table.

The IS data structure encodes boundary relation $R_{p,p-1}$, with $0 < p \leq d$, as a table, where each record corresponds to a $p$-simplex $\sigma$. Here, $\deg(\sigma) = p + 1$, thus, $R_{p,p-1}(\sigma)$ is formed by $p + 1$ simplices of dimension $p - 1$ bounding $\sigma$, namely the $(p - 1)$-faces of $\sigma$. Hence, we need $p + 1$ indices for encoding boundary relation $R_{p,p-1}(\sigma)$, thus the storage cost of boundary relations for all the simplices in $\Sigma$ is equal to:

$$\sum_{p=1}^{d} (p + 1)s_p$$

where $s_p$ is the number of $p$-simplices in $\Sigma$.

The IS data structure encodes co-boundary relation $R^*_{p,p+1}$, with $0 \leq p < d$, as a table, where each record corresponds to a $p$-simplex $\sigma$. Here, a simplex $\sigma$ is related to $h_\sigma$ representative $(p+1)$-simplices of connected components in $Lk(\sigma)$. Thus, we need $h_\sigma$ indices for encoding partial co-boundary relation $R^*_{p,p+1}(\sigma)$ of each $p$-simplex $\sigma$ in $\Sigma$, with $0 \leq p < d$. We denote the total number $H_p$ of connected components in the link of $p$-simplices in $\Sigma^p$ as:

$$H_p = \sum_{\sigma \in \Sigma^p} h_\sigma$$

where $\Sigma^p$ is the collection of $p$-simplices in $\Sigma$. Thus, the storage cost $S_{IS}$ of the IS data structure is equal to:

$$S_{IS} = \sum_{p=1}^{d} (p + 1)s_p + \sum_{p=0}^{d-1} H_p$$

(4.1)
The IS data structure scales well to manifolds. In fact, if the simplicial $d$-complex $\Sigma$ is manifold, then all the top simplices in $\Sigma$ are maximal, and $h_{\sigma'} = 1$, for all the $p$-simplices $\sigma'$ in $\Sigma$, with $0 \leq p \leq d - 2$. Also, a $(d - 1)$-simplex belongs to at most two $d$-simplices, and, conversely, a $d$-simplex is bounded by its $(d - 1)$-faces. Thus, the storage cost $S_{IS}$ of the IS data structure, restricted to manifold simplicial $d$-complexes, reduces to:

$$S_{IS} \leq (d + 1)s_d + \sum_{p=1}^{d} (p + 1)s_p + \sum_{p=0}^{d-2} s_p$$

### 4.1.3 Retrieving Topological Relations

In this section, we demonstrate that the information encoded in the IS data structure is sufficient to retrieve all the topological relations for simplices in an abstract simplicial complex. In Section 4.1.3.1, we propose an algorithm for retrieving boundary relations, and, in Section 4.1.3.2, we describe an algorithm for retrieving co-boundary relations. Finally, in Section 4.1.3.3, we describe an algorithm for retrieving adjacency relations.

#### 4.1.3.1 Retrieving Boundary Relations

In this section, given a simplicial $d$-complex $\Sigma$, we propose an algorithm for retrieving boundary relation $R_{p,q}(\sigma)$, with $p > q$, for any $p$-simplex $\sigma$ in $\Sigma$, with $0 < p \leq d$.

Any boundary relation $R_{p,q}(\sigma)$, with $p > q$, is obtained by retrieving encoded boundary relation $R_{p,p-1}(\sigma)$, and, then, by recursively combining boundary relations $R_{i,i-1}(\sigma')$ for all the $i$-faces $\sigma'$ of $\sigma$, for $q < i < p$. This traversal ends when we reach a $q$-simplex. We visit a subgraph of the IS boundary graph, formed by all the arcs and nodes of dimension greater or equal than $q$, which are reachable from the node representing $\sigma$. Hence, the time complexity of this algorithm is linear in the number of simplices in boundary relation $R_{p,q}$.

Recall that a $p$-simplex $\sigma$ has $\sigma^p_k = \binom{p+1}{k+1}$ faces of dimension $k$, with $0 \leq k < p$ [Ede87]. In this algorithm, we visit all the $k$-faces of $\sigma$, with $q \leq k < p$, and their number is:

$$\sum_{k=q}^{p-1} \binom{p+1}{k+1}$$

This number can be expressed as a constant $C_{p,q}$, which depends only on $p$ and $q$. Hence, the retrieval of boundary relation $R_{p,q}(\sigma)$, for a $p$-simplex $\sigma$, has a time complexity in $O(1)$. Boundary relations are constant in a simplicial complex, thus they are optimal in the IS data structure.

For instance, in order to retrieve boundary relation $R_{3,0}$ of tetrahedron $t = (1, 2, 3, 4)$ in Figure 4.2(a), we visit a subgraph of the IS boundary graph, shown in Figure 4.2(b). First, we retrieve triangles bounding $t$, directly encoded in boundary relation $R_{3,2}$ (in red), and then, for each of these triangles, we visit their edges, directly encoded in boundary relation $R_{2,1}$ (in green). Finally, we retrieve their vertices through boundary relation $R_{1,0}$ (in blue).
4.1.3.2 Retrieving Co-boundary Relations

In this section, given a simplicial $d$-complex $\Sigma$, we describe an algorithm for retrieving co-boundary relation $R_{p,q}(\sigma)$, with $p < q$, for any $p$-simplex $\sigma$ in $\Sigma$, with $0 \leq p < d$. All the $q$-simplices incident at a $p$-simplex $\sigma$ are either top $q$-simplices, or $q$-faces belonging to any top $h$-simplex in $St(\sigma)$, with $h > q$. Thus, in order to compute co-boundary relation $R_{p,q}(\sigma)$, we need to retrieve top $h$-simplices in $St(\sigma)$, for $p < h \leq d$, and visit their $q$-faces, which are incident at $\sigma$.

We can also describe this algorithm in terms of the IS co-boundary graph. In this context, we introduce the IS star-graph $G^{\sharp S}_d$ of a $p$-simplex $\sigma$ as a subgraph of the IS co-boundary graph. Each node in the IS star-graph corresponds to one simplex in $St(\sigma)$, and each arc corresponds, for any $q$-simplex $\sigma'$ in $St(\sigma)$, with $q > p$, either to boundary relation $R_{q,q-1}(\sigma')$, or to partial co-boundary relation $R_{q-1,q}(\sigma')$, restricted to $St(\sigma)$. Figure 4.3(a) shows all the simplices belonging to the star of a vertex $v$ in the same simplicial 3-complex of Figure 4.2(a). In other words, Figure 4.3(a) shows the IS star-graph for a vertex $v$. For the sake of clarity, boundary and co-boundary relations, restricted to $St(v)$, are shown separately. Figure 4.3(b) shows boundary relations $R_{1,0}$, $R_{2,1}$, and $R_{3,2}$, while Figure 4.3(c) shows partial co-boundary relations $R^*_{0,1}$, $R^*_{1,2}$, and $R^*_{2,3}$.

Thus, the retrieval of co-boundary relation $R_{p,q}(\sigma)$ is equivalent to a breadth-first traversal of the IS star-graph $G^{\sharp S}_d$ for a $p$-simplex $\sigma$. Figure 4.3 shows how retrieve co-boundary relation $R_{0,1}(v)$ for a vertex $v$ belonging to the simplicial 3-complex of Figure 4.2(a). This traversal starts from vertex $v$, and, then, we reach edge $e_2$ through partial co-boundary relation $R^*_{0,1}$. Then, tetrahedron $t_1$ is visited through partial co-boundary relations $R^*_{1,2}(e_1)$ and $R^*_{2,3}(f_1)$, as shown in Figure 4.3(d). We visit all the faces of $t_1$ through boundary relation $R_{3,2}(t_1)$. Similarly, all the edges of triangles $f_1$, $f_2$, and $f_3$ are visited through their boundary relations $R_{2,1}$, as shown in Figure 4.3(e). Partial co-boundary relation $R_{1,2}(e_3)$ leads to top triangle $df_1$. Then, we reach edge $e_4$ through boundary relation $R_{2,1}(df_1)$, as shown in Figure 4.3(f). Figure 4.3 shows a breadth-first traversal of the second biconnected component in the IS star graph $G^{\sharp S}_d$ of $v$. At the end of this traversal, all the simplices in $St(v)$ have been visited.

In this algorithm, we perform a breadth-first traversal of the IS star-graph $G^{\sharp S}_d$ for a $p$-simplex $\sigma$, and, thus, all the nodes and arcs of $G^{\sharp S}_d$ are visited exactly once. It is mandatory to visit all the top simplices in $St(\sigma)$ in order to retrieve $q$-simplices incident at $\sigma$. The number of arcs is linear in the number of nodes in $G^{\sharp S}_d$, since each simplex is bounded by a constant number of simplices. The number of nodes in $G^{\sharp S}_d$ is $O(\sigma^*_1)$, where $\sigma^*_1$ is the number of top simplices in $St(\sigma)$.

Hence, the time complexity for retrieving co-boundary relation $R_{p,q}(\sigma)$ is linear in $\sigma^*_1$. In any case, $\sigma^*_1$ is linear in $|St(\sigma)|$ only for simplicial 2- and 3-complexes embedded in $\mathbb{E}^3$. Thus, in the IS data structure, co-boundary relations are optimal only for simplicial complexes embedded in $\mathbb{E}^3$. Conversely, co-boundary relations are local for simplicial $h$-complexes, with $h \geq 4$. 

62
Figure 4.3: (a) A reorganization of all the simplices in a simplicial 3-complex $\Sigma$ in Figure 4.2(a). Each node in the IS star-graph $G_v^{IS}$ of vertex $v$ in $\Sigma$ corresponds to one simplex in $St(v)$, while each arc describes, for $h \geq p$, either (b) boundary relations $R_{h,h-1}$ or (c) partial co-boundary relations $R_{h-1,h}$, restricted to $St(\sigma)$. (d-g) We perform a breadth-first traversal of $G_v^{IS}$ for retrieving co-boundary relation $R_{0,1}(v)$. 

$t_1 = (1,2,3,4) \quad t_2 = (1,5,6,7)$

$t_3 = (1,2,3) \quad t_4 = (1,5,6)$

$t_5 = (1,3,4) \quad t_6 = (1,5,7)$

$e_1 = (1,2) \quad e_7 = (1,5)$

$e_2 = (1,3) \quad e_8 = (1,6)$

$e_3 = (1,4) \quad e_9 = (1,7)$

$\nu = 1 \quad df = (1,4,5)$
4.1.3.3 Retrieving Adjacency Relations

In this section, we describe an algorithm for retrieving adjacency relation of any simplex from the IS data structure. In particular, given a simplicial $d$-complex $\Sigma$, the challenge is to retrieve, for any $p$-simplex $\sigma$, with $0 \leq p \leq d$, adjacency relation $R_{p,p}(\sigma)$.

Given a $p$-simplex $\sigma$ in $\Sigma$, with $0 < p \leq d$, any adjacency relation $R_{p,p}(\sigma)$, is obtained by retrieving, for all the simplices $\sigma'$ in $R_{p,p-1}(\sigma)$, co-boundary relations $R_{p-1,p}(\sigma')$. Given a vertex $v$ in $\Sigma$, adjacency relation $R_{0,0}(v)$ is obtained by retrieving, for all the edges $e$ in $R_{0,1}(v)$, boundary relation $R_{1,0}(e)$.

Boundary relations $R_{1,0}$ and $R_{p,p-1}$ are directly encoded in the IS data structure. Running time for retrieving adjacency relation $R_{p,p}$, with $0 \leq k \leq d$, is dominated by the retrieval of either co-boundary relation $R_{0,1}$ or co-boundary relation $R_{p,p+1}$. As discussed in Section 4.1.3.2, any co-boundary relation is performed in time linear in the number $\sigma^*$ of top simplices incident at any simplex $\sigma$. As a consequence, adjacency relations are optimal only for simplicial complexes embedded in $\mathbb{E}^3$. Conversely, they are local for simplicial $h$-complexes, with $h \geq 4$.

4.2 The Incidence Graph for Simplicial Complexes

In this section, we briefly review the Incidence Graph (IG) [Ede87], restricted to abstract simplicial complexes. In Section 3.1.2, we have already discussed a generic version of the IG data structure for the representation of abstract cell complexes.

Following [DF03], the IG data structure encodes all the simplices of an abstract simplicial $d$-complex $\Sigma$, and, for each $p$-simplex $\sigma$ in $\Sigma$, boundary relation $R_{p,p-1}(\sigma)$, with $0 < p \leq d$, and co-boundary relation $R_{p,p+1}(\sigma)$, with $0 \leq p < d$.

The Incidence Graph can be described through a graph-based representation $G = (N,A)$, which we call the IG-graph. Each node in $N$ corresponds to a simplex in $\Sigma$, while an arc in $A$ corresponds to an incidence relation for the simplices described by nodes in the IG-graph. An arc $(\sigma,\sigma')$ is said to be a boundary arc, if it connects nodes related to a $p$-simplex $\sigma$, with $0 < p \leq d$, and a $(p-1)$-simplex $\sigma'$ in $\Sigma$, such that $\sigma' \in R_{p-1,p}(\sigma)$. The reverse arc $(\sigma',\sigma)$ is called a co-boundary arc. We define the IG boundary graph as the spanning subgraph of $G$ containing all the nodes and boundary arcs of $G$. Also, we define the IG co-boundary graph as the spanning subgraph of $G$ containing all the nodes and co-boundary arcs of $G$.

Note that, in this case, arcs in the IG boundary graph and IG co-boundary graph are symmetric. Moreover, the IG data structure encodes the same boundary relations as the IS data structure, thus the IG boundary graph coincides with the IS boundary graph. Figure 4.2(b) shows the IS boundary graph (and thus also the IG boundary and co-boundary graphs) for a simplicial 3-complex formed by two tetrahedra and a top triangle. Note that the orientation of each co-boundary arc is the reverse of a boundary arc.
Given a $p$-simplex $\sigma$ in $\Sigma$, we can consider a subgraph $G_\sigma^{IG}$ of the IG co-boundary graph related to a $p$-simplex $\sigma$. Each node in this graph, which we call a IG star-graph $G_{IG}^\sigma$, corresponds to one simplex in $St(\sigma)$, and each arc corresponds, for any $q$-simplex $\sigma'$ in $St(\sigma)$, with $q > p$, either to boundary relation $R_{q,q-1}(\sigma')$, or to co-boundary relation $R_{q,q+1}(\sigma')$, restricted to $St(\sigma)$. Figure 4.4 shows the IG star-graph of a vertex $v$ in a simplicial 2-complex. For the sake of clarity, boundary relations $R_{2,1}$ and $R_{1,0}$ are shown in Figure 4.4(b), while co-boundary relations $R_{0,1}$ and $R_{1,2}$ are shown in Figure 4.4(c).

In order to encode a simplicial $d$-complex $\Sigma$ through the IG data structure, we can exploit the same approach used in Section 4.1 for the IS data structure, largely inspired by [AJ05]. The IG data structure encodes the same boundary relations as the IS data structure. Thus, for each $p$-simplex $\sigma$, with $0 < p \leq d$, we need $p+1$ indices for encoding boundary relation $R_{p,p-1}(\sigma)$. Given a $(p-1)$-simplex $\sigma'$ bounding a $p$-simplex $\sigma$, we can state that $\sigma$ is also stored in co-boundary relation $R_{p-1,p}(\sigma')$. Hence, for all $1 \leq p \leq d$, relations $R_{p,p-1}$ and $R_{p-1,p}$ involve the same number of elements. As a consequence, the storage cost $S_{IG}$ of the IG data structure is equal to:

$$S_{IG} = 2 \sum_{p=1}^{d} (p+1)s_p$$

(4.2)

where $s_p$ is the number of $p$-simplices in $\Sigma$.

Let $S_{IS}$ be the storage cost of the IS data structure, provided by Equation 4.1, then we can define the difference $\Delta_{IS}^{IG} = S_{IG} - S_{IS}$ as:

$$\Delta_{IS}^{IG} = \sum_{p=1}^{d} (p+1)s_p - \sum_{p=0}^{d-1} H_p$$

Our experimental tests, presented in Section 4.4, show that the IG data structure results in a verbose representation, which does not scale well to manifolds. In fact, if any simplicial $d$-complex $\Sigma$ is manifold, then $S_{IG}$ is always provided by Equation 4.2, while the IS data structure scales well to manifolds, as demonstrated in Section 4.1.2. Hence, the difference $\Delta_{IS}^{IG} = S_{IG} - S_{IS}$ is:

$$\Delta_{IS}^{IG} \geq ds_{d-1} - s_0 - \sum_{p=1}^{d-2} p s_p$$

65
The IG data structure supports a simple recursive strategy to retrieve topological relations in a simplicial d-complex Σ, as discussed in [DF03]. All the topological relations can be retrieved in optimal time from the IG data structure.

Given a p-simplex σ in Σ, with 0 < p ≤ d, any boundary relation \( R_{p,q}(σ) \), with \( p > q \), is obtained by retrieving the encoded boundary relation \( R_{p,p−1}(σ) \), and recursively combining boundary relation \( R_{i,i−1}(σ') \), for all the \( i \)-faces \( σ' \) of σ, with \( q < i < p \). Thus, the time complexity of this algorithm is linear in the number of simplices in \( R_{p,q}(σ) \).

Given a p-simplex σ in Σ, with 0 ≤ p < d, any co-boundary relation \( R_{p,q}(σ) \), with \( p < q \), is obtained by recursively retrieving, for all \( p < i < q \), the encoded co-boundary relation \( R_{i,i+1}(σ) \) of simplices in \( St(σ) \), starting from simplices in \( R_{p,p+1}(σ) \). Thus, the time complexity of this algorithm is linear in the number of simplices in \( R_{p,q}(σ) \).

Given a p-simplex σ in Σ, with 0 < p ≤ d, any adjacency relation \( R_{p,p}(σ) \), is obtained by retrieving, for all the simplices \( σ' \) in \( R_{p,p−1}(σ) \), co-boundary relation \( R_{p−1,p}(σ') \). Given a vertex \( v \) in Σ, adjacency relation \( R_{0,0}(v) \) can be obtained by retrieving, for all the edges \( e \) in \( R_{0,1}(v) \), boundary relation \( R_{1,0}(e) \). Thus, the time complexity of this algorithm is linear in the number of simplices in \( R_{p,p}(σ) \), with 0 ≤ p ≤ d.

4.3 The Simplified Incidence Graph

In this section, we briefly review the Simplified Incidence Graph (SIG) [DFGH04], a simplicial variant of the IG data structure. The SIG data structure is an explicit, dimension-independent, and incidence-based representation for abstract simplicial complexes, not necessarily embedded in any Euclidean space.

The SIG data structure encodes the same boundary relations as the IS data structure. Moreover, the star of a p-simplex σ in Σ, with 0 ≤ p < d, is decomposed into a set of maximal q-clusters, with \( p < q ≤ d \) (see Section 2.1). Each q-cluster is represented by one of its top q-simplices, arbitrarily selected, which is known as the representative simplex. We denote all the representative simplices of q-clusters in \( St(σ) \) as partial co-boundary relation \( R^∗_{p,q}(σ) \). For instance, Figure 4.5 shows a simplicial 3-complex, where the star of a vertex \( v \) is formed by three clusters, namely a 1-cluster (corresponding to a top edge \( w \)), a 2-cluster (corresponding to a top triangle \( f \)), and a 3-cluster (corresponding to a tetrahedron \( t \)). Here, \( R^*_{0,1}(v) = \{ w \} \), \( R^*_{0,2}(v) = \{ f \} \), and \( R^*_{0,3}(v) = \{ t \} \).

As a consequence, the SIG data structure encodes all the simplices of a simplicial d-complex Σ, and, for each p-simplex σ in Σ, boundary relation \( R_{p,p−1}(σ) \), with 0 < p ≤ d, and partial co-boundary relation \( R^*_{p,q}(σ) \), with 0 ≤ p < d and \( p < q ≤ d \), as defined above.

The Simplified Incidence Graph can be described through a graph-based representation \( G = (N, A) \), which we call the SIG-graph. Each node in N corresponds to a simplex in Σ, while an arc in A corresponds to an incidence relation for the simplices described by nodes in the SIG-
Figure 4.5: Examples of clusters in the star of a vertex $v$ in a simplicial 3-complex. Here, $St(v)$ is formed by three clusters, which consist, respectively, of a top edge $w$, a top triangle $f$, and a tetrahedron $t$. Thus, $R^*_0(v) = \{w\}$, $R^*_1(v) = \{f\}$, and $R^*_2(v) = \{t\}$.

Figure 4.6: The SIG co-boundary graph for the same simplicial 3-complex in Figure 4.2(a). The SIG boundary graph coincides with the IS boundary graph shown in Figure 4.2(b). Nodes and arcs in red correspond to non-manifold vertices and edges.

Nodes in red, which correspond to vertices 1, 4 and 5, and edges (1,4) and (4,5), have more than one outgoing arcs (in red), thus their star is formed by more than one cluster. Note that, if a simplex $\sigma$ is manifold, then $St(\sigma)$ is formed by only one cluster. In Chapter 7 (see Section 7.3.2), we exploit these properties of the SIG data structure for recognizing non-manifold singularities.

In order to encode a simplicial $d$-complex $\Sigma$ through the SIG data structure, we can exploit the same approach used in Section 4.1 for the IS data structure, largely inspired by [AJ05]. The SIG data structure encodes the same boundary relations as the IS data structure. Thus, for each
$p$-simplex $\sigma$, with $0 < p \leq d$, we need $p + 1$ indices for encoding boundary relation $R_{p,p-1}(\sigma)$. The SIG data structure encodes, for $0 \leq p < q \leq d$, each partial co-boundary relation $R^*_{p,q}(\sigma)$ as a table, where each record corresponds to a $p$-simplex $\sigma$. Each simplex $\sigma$ is related to $k^q(\sigma)$ representative $q$-simplices for all the $q$-clusters in partial co-boundary relation $R^*_{p,q}(\sigma)$. Thus, we need $k^q(\sigma)$ indices for encoding partial co-boundary relation $R^*_{p,q}(\sigma)$. The total number $K^q_p$ of $q$-clusters for all the $p$-simplices in $\Sigma$ is equal to:

$$K^q_p = \sum_{\sigma \in \Sigma^p} k^q(\sigma)$$

where $\Sigma^p$ is the collection of $p$-simplices in $\Sigma$. As a consequence, the storage cost $S_{SIG}$ of the SIG data structure is equal to:

$$S_{SIG} = \sum_{p=1}^{d} (p + 1)s_p + \sum_{p=0}^{d-1} \sum_{q=p+1}^{d} K^q_p$$

(4.3)

where $s_p$ is the number of $p$-simplices in $\Sigma$.

The SIG data structure scales well to manifolds. If the simplicial $d$-complex $\Sigma$ is manifold, then all the top simplices in $\Sigma$ are maximal, and, for any $p$-simplex $\sigma$ in $\Sigma$, only partial co-boundary relations $R^*_{p,d}(\sigma)$ are not empty. Hence, $k^d(\sigma') = 1$, for all the $p$-simplices $\sigma'$ in $\Sigma$, with $0 \leq p \leq d - 2$. Also, a $(d-1)$-simplex belongs to at most two $d$-simplices, and, conversely a $d$-simplex is bounded by its $(d-1)$-faces. Thus, the storage cost $S_{SIG}$ of the SIG data structure, restricted to manifold simplicial $d$-complexes, reduces to:

$$S_{SIG} \leq (d + 1)s_d + \sum_{p=1}^{d} (p + 1)s_p + \sum_{p=0}^{d-2} s_p$$

which coincides with the storage cost of the IS data structure, restricted to manifold simplicial $d$-complexes (see Section 4.1.2).

Let $S_{IS}$ be the storage cost of the IS data structure, provided by Equation 4.1, then we can define the difference $\Delta_{SIG} = S_{SIG} - S_{IS}$ as:

$$\Delta_{SIG} = \sum_{p=0}^{d-1} \sum_{q=p+1}^{d} K^q_p - \sum_{p=0}^{d-1} H_p$$

Following [DFGH04], the SIG data structure supports a simple recursive strategy to retrieve topological relations in a simplicial $d$-complex $\Sigma$.

Boundary relations are retrieved in the same way as they are retrieved from the IS data structure, discussed in Section 4.1. Thus, boundary relations are optimal in the SIG data structure.

Following [DFGH04], given a $p$-simplex $\sigma$, with $0 \leq p < d$, any co-boundary relation $R_{p,q}(\sigma)$, with $p < q$, is retrieved in two steps. In the first step, we retrieve, for $q \leq k \leq d$, all the top $k$-simplices incident at $\sigma$ by performing a breadth-first traversal of each $k$-cluster in $St(\sigma)$. We can expand a $k$-cluster incident at $\sigma$ as the transitive closure of the $(k-1)$-adjacency relation of
its representative simplex $\sigma'$. In the second step, we retrieve, for each top simplex in $St(\sigma)$, all the $q$-faces incident at $\sigma$. The time complexity of this algorithm is linear in the number $\sigma^*_q$ of top simplices in $St(\sigma)$. In any case, $\sigma^*_q$ is linear in the number of simplices in $St(\sigma)$ only for simplicial 2-complexes and 3-complexes embedded in $\mathbb{E}^3$. Hence, co-boundary relations are optimal only for simplicial complexes embedded in $\mathbb{E}^3$, while they are local for simplicial $h$-complexes, with $h \geq 4$.

Following [DFGH04], given a $p$-simplex $\sigma$, with $0 < p \leq d$, any adjacency relation $R_{p,p}(\sigma)$ is retrieved in two steps. First, we retrieve all the top $q$-simplices, with $p < q \leq d$, which are incident at each face of $\sigma$. In the second step, for each top simplex in $St(\sigma)$, we extract all the $p$-faces sharing a $(p-1)$-simplex with $\sigma$. Given a vertex $v$ in $\Sigma$, adjacency relation $R_{0,0}(v)$ is obtained by retrieving, for all the edges $e$ in $R_{0,1}(v)$, boundary relation $R_{1,0}(e)$. The time complexity of this algorithm is dominated by the retrieval of all the top simplices in $St(\sigma)$, similarly as co-boundary relations. Hence, adjacency relations are optimal only for simplicial complexes embedded in $\mathbb{E}^3$, while they are local for simplicial $h$-complexes, with $h \geq 4$.

In [DFGH04, Hui08] the authors provide an algorithm for building the SIG data structure in two steps from an existing Incidence Graph, which represents a simplicial $d$-complex. In the first step, we copy all the $p$-simplices, with $0 \leq p \leq d$, and their boundary relations, directly encoded in the IG data structure. In the second step, we retrieve all the maximal $h$-clusters in the star of each $p$-simplex, with $h > p$, and select one top $h$-simplex for each of them. This algorithm is very simple and efficient, since all the topological relations are optimal in the Incidence Graph. However, this algorithm requires an existing IG data structure, which results in a verbose representation.

### 4.4 Experimental Comparisons

In this section, we present a quantitative comparison of the incidence-based and dimension-independent representations discussed in this chapter, namely the IS, IG, and SIG data structures. Our tests are performed in terms of their storage costs and encoded relations. In Section 4.4.1 and Section 4.4.2, we compare specializations of the IS, IG, and SIG data structures, restricted to arbitrary simplicial 2-complexes and 3-complexes, respectively. We have performed our tests on all the digital shapes freely available from [GGG09]. For the sake of brevity, we present results only on a subset of these shapes. Our comparisons show that the IS data structure is more compact than the IG representation, and, it is as compact as the SIG data structure for arbitrary simplicial 2-complexes, and more compact than the SIG representation for arbitrary simplicial 3-complexes.

#### 4.4.1 Experimental Comparisons for Simplicial 2-Complexes

In this section, we present quantitative comparisons for specializations of the IS, IG, and SIG data structures, restricted to a simplicial 2-complex $\Sigma$, namely the $IS(2D)$, $IG(2D)$, and $SIG(2D)$ representations.

The IS(2D) data structure encodes all the simplices in $\Sigma$, plus boundary relations $R_{1,0}$ and $R_{2,1}$,
partial co-boundary relation $R_{0,1}$, and co-boundary relation $R_{1,2}$. A triangle is encoded for each of its three edges, thus $H_1 = 3s_2$. For a vertex $v$, the IS(2D) data structure encodes several top edges, and one edge for each connected component in $Lk(v)$, which corresponds to 1- and 2-clusters in $St(v)$. Moreover, a top edge is encoded for each of its vertices, thus $H_0 = 2s_1^* + K_0^2$. From Equation 4.1, we deduce that the storage cost $S^{IS}_{IS}$ of the IS(2D) data structure is:

$$S^{IS}_{IS} = 2s_1 + 6s_2 + 2s_1^* + K_0^2$$

If $\Sigma$ is regular, then $s_1^* = 0$, thus the storage cost $S^{IS}_{IS}$ becomes:

$$S^{IS}_{IS} = 2s_1 + 6s_2 + K_0^2$$

The IS(2D) data structure scales well to manifolds. In this case, there are no top edges, and only a reference to one edge is stored, for each vertex. Hence, $S^{IS}_{IS} = s_0 + 2s_1 + 6s_2 \approx 19s_0$, since $s_1 \approx 3s_0$ and $s_2 \approx 2s_0$ [Ede87].

The IG(2D) data structure encodes the same simplices and boundary relations as the IS(2D) data structure. Specifically, it encodes co-boundary relations $R_{0,1}$ and $R_{1,2}$. From Equation 4.2, we deduce that the storage cost $S^{IG}_{IG}$ of the IG(2D) data structure is:

$$S^{IG}_{IG} = 4s_1 + 6s_2$$

Hence, we can define the difference $\Delta^{IG}_{IG}(2D) = S^{IG}_{IG} - S^{IS}_{IS} = 2s_1 - 2s_1^* - K_0^2$. If $\Sigma$ is regular, then $\Delta^{IG}_{IG}(2D) = 2s_1 - K_0^2$. Finally, if $\Sigma$ is manifold, then $\Delta^{IG}_{IG}(2D) = 2s_1 - s_0 \approx 5s_0$ and $S^{IG}_{IG} \approx 24s_0$. Both the IG(2D) and IS(2D) data structures encode boundary relations $R_{1,0}$ and $R_{2,1}$, plus co-boundary relations $R_{1,2}$. These relations require $||R_{1,0}|| = 2s_1$, and $||R_{2,1}|| = ||R_{1,2}|| = 3s_2$ indices, respectively, and do not play a key role in the difference between the IG(2D) and IS(2D) representations. Thus, we must compare the storage cost $||R_{0,1}|| = 2s_1$ of co-boundary relation $R_{0,1}$, and the storage cost $||R_{0,1}^*|| = 2s_1 + K_0^2$ of partial co-boundary relation $R_{0,1}^*$. Here, $\Delta^{IG}_{IG}(2D) = ||R_{0,1}^*|| - ||R_{0,1}||$.

The SIG(2D) data structure encodes the same simplices and boundary relations as the IS(2D) data structure. Specifically, it encodes partial co-boundary relations $R_{0,1}^*$ and $R_{0,2}^*$, plus co-boundary relation $R_{1,2}$. In this context, 1- and 2-clusters incident at any vertex coincide with the connected components of $Lk(v)$, encoded in the IS(2D) data structure. Moreover, a top edge is encoded for each of its vertices, thus $K_1^* = 2s_1$. A triangle is encoded for each of its three edges, thus $K_1^2 = 3s_2$. From Equation 4.3, we deduce that the storage cost $S^{SIG}_{SIG}$ of the SIG(2D) data structure is:

$$S^{SIG}_{SIG} = 2s_1 + 6s_2 + 2s_1^* + K_0^2$$

In the remainder of this section, we consider only the IS(2D) data structure, since $S^{SIG}_{SIG} = S^{IS}_{IS}$.

Now, we compare the IS(2D) and IG(2D) data structures with several data structures, which represent triangulations, namely manifold simplicial 2-complexes. In Table 4.1, we compare edge-based representations discussed in Section 3.2.1.1, like the WE data structure [Bau75], and HE data structure [Man87]. We also compare a specialization to triangulations of the X-Maps data structure [CK10], discussed in Section 3.3.3. In this context, $s_1 \approx 3s_0$ and $s_2 \approx 2s_0$ [Ede87].
Table 4.1: Storage costs of the IS(2D) ($S_{IS}^{2D}$), IG(2D) ($S_{IG}^{2D}$), and other data structures, discussed in Section 3.2.1.1, which describe triangulations. Specifically, we analyze storage costs of the HE ($S_{HE}$), WE ($S_{WE}$), and X-Maps ($S_{XM}$) data structures. Recall that $s_j$ is the number of $j$-simplices in a triangulation, with $0 \leq j \leq 2$.

These tests show that the IS(2D) data structure scales well for manifolds. In this context, $S_{IS}^{2D} \approx 24s_0$, and $S_{IG}^{2D} \approx 19s_0$ indices. As a consequence, the IG(2D) data structure is about 1.26 times more expensive than the IS(2D) representation. The X-Maps data structure has the same storage cost as the IG(2D) representation. The WE and HE data structures are more expensive than the IS(2D) data structure, since they require $27s_0$ and $33s_0$ indices, respectively. Hence, the WE and HE data structures are about 1.42 times and 1.7 times more expensive than the IS(2D) data structure, respectively.

Now, we compare the IS(2D) and IG(2D) data structures with several data structures for arbitrary cell 2-complexes, restricted to simplicial 2-complexes. Specifically, we compare the RE data structure [Wei88b], discussed in Section 3.2.2, the PE data structure [LL01], discussed in Section 3.2.3, and the DE data structure [CKS98], discussed in Section 3.2.5. In Table 4.2, we summarize these tests. Specifically, it is interesting to study the behavior of all the data structures with regular and non-manifold simplicial shapes, and with non-manifold and non regular simplicial shapes.

Table 4.2: Storage costs of the IS(2D) ($S_{IS}^{2D}$), IG(2D) ($S_{IG}^{2D}$), and some data structures, discussed in Section 3.2, which describe arbitrary simplicial 2-complexes. Specifically, we analyze storage costs of the RE ($S_{RE}$), PE ($S_{PE}$), and DE ($S_{DE}$) data structures. Recall that $s_j$ is the number of $j$-simplices in a simplicial 2-complex, with $0 \leq j \leq 2$.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$s_0$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$S_{RE}$</th>
<th>$S_{PE}$</th>
<th>$S_{DE}$</th>
<th>$S_{IS}^{2D}$</th>
<th>$S_{IG}^{2D}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cone</td>
<td>0.6k</td>
<td>1.8k</td>
<td>1.2k</td>
<td>20k</td>
<td>16k</td>
<td>14.4k</td>
<td>14.4k</td>
<td>11.4k</td>
</tr>
<tr>
<td>Crumb</td>
<td>0.3k</td>
<td>0.9k</td>
<td>0.6k</td>
<td>10k</td>
<td>8k</td>
<td>7.2k</td>
<td>7.2k</td>
<td>5.7k</td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>0.08k</td>
<td>0.24k</td>
<td>0.16k</td>
<td>2.6k</td>
<td>2.2k</td>
<td>1.92k</td>
<td>1.92k</td>
<td>1.52k</td>
</tr>
<tr>
<td>Football 1</td>
<td>1.2k</td>
<td>3.6k</td>
<td>2.4k</td>
<td>39.6k</td>
<td>32.4k</td>
<td>28.8k</td>
<td>28.8k</td>
<td>22.8k</td>
</tr>
<tr>
<td>Football 2</td>
<td>0.9k</td>
<td>2.7k</td>
<td>1.8k</td>
<td>29.7k</td>
<td>24.3k</td>
<td>21.6k</td>
<td>21.6k</td>
<td>17.1k</td>
</tr>
<tr>
<td>Torus</td>
<td>10.2k</td>
<td>30.6k</td>
<td>20.4k</td>
<td>336k</td>
<td>275k</td>
<td>244.8k</td>
<td>244.8k</td>
<td>191.8k</td>
</tr>
</tbody>
</table>
These tests show that the IS(2D) data structure is more compact than the RE, PE, DE, and IG(2D) data structures. Specifically, the RE data structure, if restricted to simplicial complexes, results in a verbose representation, and it is about 7.7 times more expensive than the IS(2D) representation. The PE data structure is about 2.7 times more expensive than the IS(2D) representation. The DE data structure, which is 1.3 to 1.5 times larger than the Incidence Graph, is about 1.7 times more expensive than the IS(2D) data structure. Finally, the IG(2D) data structure is about 1.26 times more expensive than the IS(2D) representation.

Now, we compare the IG(2D) and IS(2D) representations by evaluating their storage costs in terms of topological relations they encode. Table 4.3 summarizes contributions to $S_{IG}^D$ and $S_{IS}^D$ for all the simplicial 2-complexes analyzed in Table 4.1 and Table 4.2. Specifically, we consider storage cost of boundary relations, namely $2s_1 + 3s_2$. We also analyze the total number $\|R_{0,1}\|$, $\|R_{1,2}\|$, and $\|R_{0,1}^*\|$ of elements in relations $R_{0,1}$, $R_{1,2}$, and $R_{0,1}^*$, respectively. Then, we consider the number $s_1^*$ and $K_0^2$ of, respectively, top edges and 2-clusters incident at vertices. Finally, we consider the maximum number $M_{0,1}$ and $M_{0,1}^*$ of simplices in relations $R_{0,1}$ and $R_{0,1}^*$, respectively.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$2s_1 + 3s_2$</th>
<th>$|R_{0,1}|$</th>
<th>$|R_{1,2}|$</th>
<th>$|R_{0,1}^*|$</th>
<th>$s_1^*$</th>
<th>$K_0^2$</th>
<th>$M_{0,1}$</th>
<th>$M_{0,1}^*$</th>
<th>$\Delta_{IG}^D(2D)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cone</td>
<td>7.2k</td>
<td>3.6k</td>
<td>3.6k</td>
<td>0.6k</td>
<td>0.6k</td>
<td>0.6k</td>
<td>64</td>
<td>1</td>
<td>3k</td>
</tr>
<tr>
<td>Crumb</td>
<td>3.6k</td>
<td>1.8k</td>
<td>1.8k</td>
<td>0.3k</td>
<td>0.3k</td>
<td>0.3k</td>
<td>12</td>
<td>1</td>
<td>1.5k</td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>0.96k</td>
<td>0.48k</td>
<td>0.48k</td>
<td>0.08k</td>
<td>0.08k</td>
<td>0.08k</td>
<td>9</td>
<td>1</td>
<td>0.4k</td>
</tr>
<tr>
<td>Football 1</td>
<td>14.4k</td>
<td>7.2k</td>
<td>7.2k</td>
<td>1.2k</td>
<td>1.2k</td>
<td>1.2k</td>
<td>10</td>
<td>1</td>
<td>6k</td>
</tr>
<tr>
<td>Football 2</td>
<td>10.8k</td>
<td>5.4k</td>
<td>5.4k</td>
<td>0.9k</td>
<td>0.9k</td>
<td>0.9k</td>
<td>10</td>
<td>1</td>
<td>4.5k</td>
</tr>
<tr>
<td>Tower</td>
<td>122.4k</td>
<td>61.2k</td>
<td>61.2k</td>
<td>10.2k</td>
<td>10.2k</td>
<td>10.2k</td>
<td>8</td>
<td>1</td>
<td>51k</td>
</tr>
<tr>
<td>Armchair</td>
<td>63.6k</td>
<td>31.8k</td>
<td>31.8k</td>
<td>5.3k</td>
<td>5.3k</td>
<td>5.3k</td>
<td>22</td>
<td>1</td>
<td>26.5k</td>
</tr>
<tr>
<td>800-Cubes</td>
<td>50.2k</td>
<td>21.4k</td>
<td>28.8k</td>
<td>2.5k</td>
<td>2.5k</td>
<td>2.5k</td>
<td>18</td>
<td>1</td>
<td>18.9k</td>
</tr>
<tr>
<td>Cylinders</td>
<td>1.2k</td>
<td>0.6k</td>
<td>0.6k</td>
<td>0.1k</td>
<td>0.1k</td>
<td>0.1k</td>
<td>34</td>
<td>2</td>
<td>0.5k</td>
</tr>
<tr>
<td>Pinched-pie</td>
<td>12.7k</td>
<td>5.8k</td>
<td>6.9k</td>
<td>0.9k</td>
<td>0.9k</td>
<td>0.9k</td>
<td>32</td>
<td>2</td>
<td>4.9k</td>
</tr>
<tr>
<td>Twist</td>
<td>14.2k</td>
<td>7k</td>
<td>7.2k</td>
<td>1.1k</td>
<td>1.1k</td>
<td>1.1k</td>
<td>19</td>
<td>1</td>
<td>5.9k</td>
</tr>
<tr>
<td>Robot</td>
<td>40.1k</td>
<td>20k</td>
<td>20.1k</td>
<td>3.3k</td>
<td>3.3k</td>
<td>3.3k</td>
<td>11</td>
<td>1</td>
<td>16.7k</td>
</tr>
<tr>
<td>Balance</td>
<td>48k</td>
<td>24k</td>
<td>24k</td>
<td>3.9k</td>
<td>3.4k</td>
<td>3.8k</td>
<td>64</td>
<td>3</td>
<td>20.1k</td>
</tr>
<tr>
<td>Carter</td>
<td>47.5k</td>
<td>23.8k</td>
<td>23.7k</td>
<td>4.1k</td>
<td>2</td>
<td>4.1k</td>
<td>11</td>
<td>3</td>
<td>19.7k</td>
</tr>
<tr>
<td>Chandelier</td>
<td>110.1k</td>
<td>55.2k</td>
<td>54.9k</td>
<td>9.5k</td>
<td>136</td>
<td>9.2k</td>
<td>64</td>
<td>4</td>
<td>45.7k</td>
</tr>
<tr>
<td>Frame</td>
<td>7.7k</td>
<td>4.4k</td>
<td>3.3k</td>
<td>1.1k</td>
<td>216</td>
<td>668</td>
<td>8</td>
<td>4</td>
<td>3.3k</td>
</tr>
<tr>
<td>Tower</td>
<td>110.6k</td>
<td>55.4k</td>
<td>55.2k</td>
<td>9.3k</td>
<td>160</td>
<td>9k</td>
<td>160</td>
<td>160</td>
<td>46k</td>
</tr>
<tr>
<td>Tower-wir</td>
<td>96.9k</td>
<td>49.2k</td>
<td>47.7k</td>
<td>9.7k</td>
<td>896</td>
<td>7.9k</td>
<td>128</td>
<td>128</td>
<td>39.5k</td>
</tr>
</tbody>
</table>

Table 4.3: Contributions to the storage costs $S_{IG}^D$ and $S_{IS}^D$, for all the simplicial 2-complexes analyzed in Table 4.1 and Table 4.2. Specifically, we consider the storage cost of boundary relations, namely $2s_1 + 3s_2$. We also analyze the total number $\|R_{0,1}\|$, $\|R_{1,2}\|$, and $\|R_{0,1}^*\|$ of elements in relations $R_{0,1}$, $R_{1,2}$, and $R_{0,1}^*$, respectively. Then, we consider the number $s_1^*$ of top edges, and the number $K_0^2$ of 2-clusters incident at vertices. Finally, we consider the maximum number $M_{0,1}$ and $M_{0,1}^*$ of simplices in relations $R_{0,1}$ and $R_{0,1}^*$, respectively.

In the IG(2D) data structure, the storage costs of boundary and co-boundary relations are both 50% of $S_{IG}^D$. Again, $\Delta_{IG}^D(2D)$ depends only on $\|R_{0,1}\|$ and $\|R_{0,1}^*\|$.

If the input simplicial 2-complex $\Sigma$ is manifold, then $s_1 \approx 3s_0$, $s_2 \approx 2s_0$, $s_1^* = 0$, and $K_0^2 = s_0$. 72
Thus, $\|R_{1,2}\| = \|R_{0,1}\| \approx 6s_0$, and $\|R_{0,1}\| = s_0$. As consequence, $\Delta^D_{IS}(2D) \approx 5s_0$, and $\|R_{0,1}\| \approx 6 \times \|R_{0,1}\|$. In this context, the link of a vertex $v$ is formed by only one connected component, and, thus, the IS(2D) data structure encodes only one edge in partial co-boundary relation $R^*_0(v)$. In other words, $M^0_{0,1} = 1$. Conversely, the IG(2D) representation encodes all the edges in $St(v)$. For instance, $M^0_{0,1} = 64$ in the “Cone” shape. In the IS(2D) data structure, the storage cost of co-boundary relations requires about $7s_0$, and, thus, about $37\%$ of $S^D_{IS}$. The storage cost of boundary relations requires about $12s_0$, and, thus, about $73\%$ of $S^D_{IS}$.

If the input simplicial 2-complex $\Sigma$ is non-manifold, then the star of a vertex $v$ is formed by top edges and 2-clusters. Thus, $\Delta^D_{IS}(2D)$ depends on the complexity of non-manifold connections. In any case, note that the number of singularities is usually limited in a non-manifold simplicial shape, as demonstrated in [BCMA+11]. For instance, this is the key idea exploited in any decomposition-based representation, which we have reviewed in Section 3.4. Broadly speaking, if $\Sigma$ is regular, then there are no top edges, and, thus, there may be a limited number of non-manifold vertices in $\Sigma$, having more than one connected components in their star, like the Cylinders and Pinched-pie shapes in Table 4.3. In this context, $s'_1 = 0$, and, thus, $\|R^*_{0,1}\| = K^2_0$, and $\Delta^D_{IS}(2D) = 2s'_1 - K^2_0$.

Here, the IS(2D) data structure offers a compact encoding, because most of vertices has only one connected component in their star. Our tests show that $\|R_{0,1}\| \approx 6 \times \|R^*_{0,1}\|$ for most of non-manifold and regular shapes in Table 4.3.

In any case, the difference $\Delta^D_{IS}(2D)$ tends to decrease with respect to the complexity of non-manifold connections, especially if at least one top edge exists. For instance, if the simplicial 2-complex $\Sigma$ contains a wire-web, i.e., a maximal set of 0-connected top edges, then the difference $\Delta^D_{IS}(2D)$ tends to reduce further. Recall that a top edge is encoded in the partial co-boundary relation $R^*_0(v)$ of both its vertices. For instance, in Table 4.3, it is clear that $\|R_{0,1}\| \approx 4 \times \|R^*_{0,1}\|$ for the “Frame” shape, and $\|R_{0,1}\| \approx 5 \times \|R^*_{0,1}\|$ for the “Tower-wir” shape. Both these shapes contain wire-webs. In any case, by definition, $\|R_{0,1}\| > \|R^*_{0,1}\|$.

### 4.4.2 Experimental Comparisons for Simplicial 3-complexes

In this section, we present some quantitative comparisons for specializations of the IS, IG, and SIG data structures, restricted to a simplicial 3-complex $\Sigma$, namely the IS(3D), IG(3D), and SIG(3D) representations.

The IS(3D) data structure encodes all the simplices in $\Sigma$, plus boundary relations $R_{1,0}$, $R_{2,1}$, and $R_{3,2}$, partial co-boundary relations $R^*_{0,1}$ and $R^*_{1,2}$, and co-boundary relation $R_{2,3}$. Given a vertex $v$ in $\Sigma$, partial co-boundary relation $R^*_{0,1}(v)$ encodes either a top edge, or one edge for each 1-connected component in $St(v)$, formed by triangles and tetrahedra. Let $C^1_v$ be the number of 1-connected components formed by triangles and tetrahedra in $St(v)$, which are known as vertex-based clusters [DFH03]. In this context, the number $h_v$ of connected components in $Lk(v)$ is $h_v = k^1(v) + C^1_v$. Let $C^1$ be the total number of vertex-based clusters for all the vertices in $\Sigma$.

Thus, $H^0 = K^1_0 + C^1 = 2s'_1 + C^1$, since a top edge is encoded for each of its vertices. Given an edge $e$ in $\Sigma$, partial co-boundary relation $R^*_{1,2}(e)$ encodes either a top triangle, or one triangle
for each 3-cluster in $St(e)$. A 3-cluster in $St(e)$ is also known as an edge-based cluster [DFH03].

As a consequence, $H_1 = K_{S}^2 + K_{S}^3 = 3s_2^2 + K_{S}^3$, since a top triangle is encoded for each of its edges. Furthermore, a tetrahedron is encoded for each of its triangles, and, thus, $H_2 = 4s_3$. From Equation 4.1, we deduce that the storage cost $S_{IS}^{3D}$ of the IS(3D) data structure is:

$$S_{IS}^{3D} = 2s_1 + 3s_2 + 8s_3 + 2s_t^1 + C^1 + 3s_t^2 + K_1^3$$

If $\Sigma$ is regular, then all the top simplices are maximal, and, thus, $s_t^1 = s_t^2 = 0$. As a consequence, $S_{IS}^{3D}$ becomes:

$$S_{IS}^{3D} = 2s_1 + 3s_2 + 8s_3 + C^1 + K_1^3$$

The IS(3D) data structure scales well to manifolds. In this case, all the top simplices are maximal, and the link of each simplex is formed by only one connected component. Thus, $C^1 = s_0$, and $K_1^3 = s_1$. As a consequence, $S_{IS}^{3D}$ becomes:

$$S_{IS}^{3D} = s_0 + 3s_1 + 3s_2 + 8s_3$$

The IG(3D) data structure encodes the same simplices and boundary relations as the IS(3D) data structure. Specifically, it encodes co-boundary relations $R_{0,1}$, $R_{1,2}$, and $R_{2,3}$. From Equation 4.2, we deduce that the storage cost $S_{IG}^{3D}$ of the IG(3D) data structure, is:

$$S_{IG}^{3D} = 4s_1 + 6s_2 + 8s_3$$

We introduce the difference $\Delta_{IG}^{3D} = S_{IG}^{3D} - S_{IS}^{3D} = (2s_1 + 3s_2) - (2s_t^1 + C^1 + 3s_t^2 + K_1^3)$. If $\Sigma$ is regular, then $\Delta_{IG}^{3D} = (2s_1 + 3s_2) - (C^1 + K_1^3)$. Finally, if $\Sigma$ is manifold, then $\Delta_{IG}^{3D} = 3s_2 + s_1 - s_0$. Both the IG(3D) and IS(3D) data structures encode the same boundary relations, plus co-boundary relation $R_{2,3}$. These relations require $\|R_{1,0}\| = 2s_1$, $\|R_{1,2}\| = 3s_2$, and $\|R_{1,3}\| = \|R_{2,3}\| = 4s_3$ indices, respectively, and do not play a key role in the difference between the IG(3D) and IS(3D) representations. Hence, in the IG(3D) data structure, we have $\|R_{0,1}\| = 2s_1$, and $\|R_{1,2}\| = 3s_2$. In the IS(3D) data structure, $\|R_{0,1}\| = 2s_1 + C^1$ and $\|R_{1,2}\| = 3s_2 + K_1^3$. Hence, $\Delta_{IG}^{3D} = (\|R_{1,0}\| - \|R_{0,1}\| + \|R_{1,2}\| - \|R_{1,2}\|) = 0$. As a consequence, $\Sigma$ is regular, then all the top simplices are maximal, and, thus, $s_t^1 = s_t^2 = 0$ and $K_0^2 = 0$. As a consequence, $S_{SIG}^{3D}$ becomes:

$$S_{SIG}^{3D} = 2s_1 + 3s_2 + 8s_3 + K_0^2 + K_1^3$$

If $\Sigma$ is regular, then all the top simplices are maximal, and, thus, $s_t^1 = s_t^2 = 0$ and $K_0^2 = 0$. As a consequence, $S_{SIG}^{3D}$ becomes:

$$S_{SIG}^{3D} = 2s_1 + 3s_2 + 8s_3 + K_0^2 + K_1^3$$
The SIG(3D) data structure scales well to manifolds. In this case, all the top simplices are maximal, and the star of each simplex is formed by only one 3-cluster, and, thus, $K_0^3 = s_0$, and $K_1^3 = s_1$. As a consequence, the storage costs of the SIG(3D) and IS(3D) data structures coincide, if restricted to manifold simplicial 3-complexes.

We introduce the difference $\Delta IS(3D) = S_{SIG}^{3D} - S_{IS}^{3D} = (K_0^3 + K_0^3) - C^1$. The SIG(3D) and IS(3D) data structures encode the same boundary relations, plus co-boundary relation $R_{2,3}$. These relations require $\|R_{1,0}\| = 2s_1$, $\|R_{1,2}\| = 3s_2$, and $\|R_{2,3}\| = 4s_3$ indices, respectively, and do not play a key role in the difference between the SIG(3D) and IS(3D) representations. For each vertex $v$, both data structures encode all the top edges in $St(v)$, and, for each edge $e$, all the top triangles in $St(e)$. In other words, the difference $\Delta IS^{SIG}(3D)$ depends on the number $C_v^1$ of vertex-based clusters and on the number $k_0^1(v) + k_0^2(v)$ of 2-clusters and 3-clusters incident at a vertex $v$. Note that a vertex-based cluster can be formed by several 2-clusters and 3-clusters, as shown in Figure 4.2(a). In this example, the star of vertex $v$ is formed by only one vertex-based cluster, but there are one 2-cluster (corresponding to the top triangle $d_1$) and two 3-clusters (corresponding to the tetrahedra $t_0$ and $t_1$) in $St(v)$. If $\Sigma$ is regular, then $\Delta IS^{SIG}(3D) = K_0^3 - C^1$.

Now, we compare the IS(3D) and IG(3D) data structures with several data structures reviewed in Section 3.3, which represent tetrahedral grids, namely manifold simplicial 3-complexes. In these tests, reported in Table 4.4, we analyze the FE data structure [DL89] (restricted to simplicial 3-complexes), briefly discussed in Section 3.3.1, and the CHF data structure [LLLV05], briefly reviewed in Section 3.3.2. We also analyze a specialization to tetrahedral grids of the X-Maps data structure [CK10], discussed in Section 3.3.3. Note that $S_{SIG}^{3D} = S_{SIG}^{3D}$ for manifolds.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$s_0$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$S_{XM}$</th>
<th>$S_{FE}$</th>
<th>$S_{IG}^{3D}$</th>
<th>$S_{IS}^{3D}$</th>
<th>$S_{CHF}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>1.2k</td>
<td>6.4k</td>
<td>9.2k</td>
<td>4k</td>
<td>192k</td>
<td>139.2k</td>
<td>112.8k</td>
<td>80k</td>
<td>48.9k</td>
</tr>
<tr>
<td>Cylinder</td>
<td>1.3k</td>
<td>7.8k</td>
<td>11.6k</td>
<td>5.2k</td>
<td>249.6k</td>
<td>176k</td>
<td>142.4k</td>
<td>101.5k</td>
<td>62.1k</td>
</tr>
<tr>
<td>Gargoyle</td>
<td>2.7k</td>
<td>14.7k</td>
<td>22k</td>
<td>10k</td>
<td>480k</td>
<td>333k</td>
<td>270.8k</td>
<td>192.8k</td>
<td>119.4k</td>
</tr>
<tr>
<td>Rings</td>
<td>2.5k</td>
<td>13.2k</td>
<td>18.8k</td>
<td>8.1k</td>
<td>388.8k</td>
<td>284k</td>
<td>230.4k</td>
<td>163.3k</td>
<td>99.3k</td>
</tr>
<tr>
<td>Torus 3D</td>
<td>2.3k</td>
<td>15.4k</td>
<td>24k</td>
<td>10.9k</td>
<td>523.2k</td>
<td>362.4k</td>
<td>292.8k</td>
<td>206.5k</td>
<td>128.9k</td>
</tr>
</tbody>
</table>

Table 4.4: Storage costs of the IS(3D) ($S_{IS}^{3D}$), IG(3D) ($S_{IG}^{3D}$), and some data structures discussed in Section 3.3, which represent tetrahedral grids. In particular, we also analyze storage costs of the X-Maps($S_{XM}$), FE ($S_{FE}$), and CHF ($S_{CHF}$) data structures. Recall that $s_j$ is the number of $j$-simplices in a tetrahedral grid, with $0 \leq j \leq 3$.

These tests demonstrate that the IS(3D) data structure scales well for manifolds. For instance, the FE data structure, specific for tetrahedral grids, results in a verbose representation, and it is about 1.7 times more expensive than the IS(3D) data structure. Similarly, the X-Maps data structure provides a verbose representation, and it is about 2.4 times more expensive than the IS(3D) data structure. Here, the IG(3D) data structure is about 1.4 times more expensive than the IS(3D) representation. In any case, in the literature, there are representations, like the CHF data structure, which are more compact than the IS(3D) data structure. For instance, the IS(3D) representation is about 1.6 times more expensive than the CHF data structure.
Now, we compare the IS(3D), IG(3D), and SIG(3D) data structures, used for representing arbitrary shapes. To the best of our experience, there are very few data structures, which represent arbitrary simplicial 3-complexes. Table 4.5 summarizes storage costs of the IS(3D), SIG(3D), and IG(3D) data structures for simplicial 3D shapes used in our tests. The IG(3D) and SIG(3D) representations are about 1.38 times and 1% more expensive than the IS(3D) data structure, respectively.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$s_0$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$s_1^I$</th>
<th>$s_2^I$</th>
<th>$S_{IS}^{3D}$</th>
<th>$S_{SIG}^{3D}$</th>
<th>$S_{IG}^{3D}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc</td>
<td>252</td>
<td>261</td>
<td>354</td>
<td>158</td>
<td>13</td>
<td>–</td>
<td>4.4k</td>
<td>3.1k</td>
<td>3.1k</td>
</tr>
<tr>
<td>Balloon</td>
<td>1.1k</td>
<td>3.9k</td>
<td>3.6k</td>
<td>856</td>
<td>64</td>
<td>1.6k</td>
<td>44k</td>
<td>32.8k</td>
<td>32.8k</td>
</tr>
<tr>
<td>Bucket</td>
<td>53</td>
<td>167</td>
<td>160</td>
<td>48</td>
<td>6</td>
<td>32</td>
<td>2k</td>
<td>1.46k</td>
<td>1.44k</td>
</tr>
<tr>
<td>Chime</td>
<td>246</td>
<td>833</td>
<td>948</td>
<td>360</td>
<td>7</td>
<td>9</td>
<td>11.9k</td>
<td>8.51k</td>
<td>8.49k</td>
</tr>
<tr>
<td>Flasks</td>
<td>1.3k</td>
<td>6.3k</td>
<td>8.5k</td>
<td>3.5k</td>
<td>–</td>
<td>460</td>
<td>104.2k</td>
<td>74.9k</td>
<td>74.94k</td>
</tr>
<tr>
<td>Halves</td>
<td>252</td>
<td>1.3k</td>
<td>1.9k</td>
<td>0.8k</td>
<td>–</td>
<td>–</td>
<td>23k</td>
<td>16.31k</td>
<td>16.3k</td>
</tr>
<tr>
<td>Sierpinski</td>
<td>32.8k</td>
<td>98.3k</td>
<td>65.5k</td>
<td>16.4k</td>
<td>–</td>
<td>–</td>
<td>917.4k</td>
<td>688.1k</td>
<td>688.1k</td>
</tr>
<tr>
<td>Teapot</td>
<td>4.6k</td>
<td>17.9k</td>
<td>17k</td>
<td>5.7k</td>
<td>2.9k</td>
<td>3.9k</td>
<td>219.2k</td>
<td>162.9k</td>
<td>162.8k</td>
</tr>
<tr>
<td>Wheel</td>
<td>402</td>
<td>2.1k</td>
<td>2.7k</td>
<td>1.1k</td>
<td>96</td>
<td>32</td>
<td>31.4k</td>
<td>23.75k</td>
<td>23.7k</td>
</tr>
</tbody>
</table>

Table 4.5: Storage costs of the IS(3D) ($S_{IS}^{3D}$), IG(3D) ($S_{IG}^{3D}$), and SIG(3D) ($S_{SIG}^{3D}$) data structures with all the arbitrary simplicial 3-complexes used in our tests. Note that $s_j$ is the number of $j$-simplices in a simplicial 3-complex, with $0 \leq j \leq 3$. Moreover, we denote the number of top edges and triangles as $s_1^I$ and $s_2^I$, respectively. The “Sierpinski” shape is a courtesy of Harish Doraiswamy and Vijay Natarajan [DN12].

Now, we compare the IS(3D), IG(3D), and SIS(3D) representations by evaluating their storage costs in terms of which topological relations they encode in a simplicial 3-complex $\Sigma$. Table 4.6 shows all the contributions to $S_{IS}^{3D}$, $S_{SIG}$, and $S_{IG}^{3D}$, for all the simplicial 3-complexes analyzed in Table 4.4 and Table 4.5. Here, the storage cost of boundary relations in the IG(3D), the SIG(3D), and the IS(3D) data structures is $B = 2s_1 + 3s_2 + 4s_3$. Then, we also consider the total number $K_0^2 + K_0^3$ of 2- and 3-clusters and the number $C^1$ of vertex-based clusters incident at vertices. We also consider the number $K_1^3$ of 3-clusters incident at edges. Finally, we analyze the maximum numbers $M_{0,1}$, $M_{0,1}^*$, $M_{1,2}$, $M_{1,2}^*$ of simplices in co-boundary relations $R_{0,1}$, $R_{0,1}^*$, $R_{1,2}$, and $R_{1,2}^*$, respectively.

As just discussed, the difference $\Delta S_{IS}^{SIG}(3D)$ between the SIG(3D) and IS(3D) representations depends only on how many vertex-based clusters and 2-clusters and 3-clusters belong to the star of vertices in $\Sigma$. As shown in Table 4.6, if $\Sigma$ is manifold, then the SIG(3D) and IS(3D) representations coincide. Otherwise, $\Delta S_{IS}^{SIG}(3D)$ linearly depends on $K_0^3$ and $K_0^3 + C^1$.

In our tests, $\Delta S_{IS}^{SIG}(3D)$ does not exceed 1% of the IS(3D) representation. This difference may be larger with more complex shapes than ours, since, for any vertex $v$, the number $k_2^b(v) + k_3^b(v)$ of 2- and 3-clusters in $St(v)$ may be extremely large, if compared with the number $C_1^v$ of connected components in $Lk(v)$. For instance, in the arbitrary simplicial 3-complex in Figure 4.2(a), the star of each vertex is formed by only one connected component, and, thus, $C_1^v = 7$. Conversely, the star of vertex 1 is formed by one 2-cluster and two 3-clusters, while star of vertices 4 and 5 is formed by one 2-cluster and one 3-cluster. The star of remaining vertices is formed by only one

76
Table 4.6: Contributions to storage costs

![Table](image)

Table 4.6: Contributions to storage costs $S_{IG}^3$, $S_{SIG}^3$, and $S_{IS}^3$, for all the simplicial 3-complexes analyzed in Table 4.4 and Table 4.5. We denote the storage cost of boundary relations in the IG(3D), SIG(3D), and IS(3D) data structures as $B = 2s_1 + 3s_2 + 4s_3$. Then, we also consider the total number $K_0^3 + K_1^3$ of 2- and 3-clusters and the number $C^3$ of vertex-based clusters incident at vertices. We also consider the number $K_1^3$ of 3-clusters incident at edges. Finally, we analyze the maximum numbers $M_{0,1}$, $M_0^3$, $M_{1,2}$, $M_1^3$, and $M_2^3$ of simplices in co-boundary relations $R_{0,1}$, $R_{0,1}^*$, $R_{1,2}$, and $R_{1,2}^*$, respectively. The “Sierpinski” shape is a courtesy of Harish Doraiswamy and Vijay Natarajan [DN12].

3-cluster, and, thus, $K_0^3 = 3$ and $K_1^3 = 8$. As a consequence, $\Delta_{IG}^3(3D) = 4$. In any case, it is quite clear that the SIG(3D) data structure is more expensive than the IS(3D) representation, if the star of a vertex is formed by pieces of different dimensions.

![Diagram](image)

Figure 4.7: A comparison between co-boundary relations $R_{0,1}$ and $R_{0,1}^*$ for a manifold simplicial 3-complex. (a) The IG(3D) data structure encode all the edges incident at a vertex $v$, while (b) the IS(3D) data structure encodes only edge $e$.

The difference $\Delta_{IS}^3(3D)$ is maximum when the input simplicial 3-complex $\Sigma$ is manifold. In this
case, all the top simplices in Σ are maximal, and the star of vertices and edges is formed by only one connected component. Hence, the IS(3D) data structure encodes, for any vertex \( v \) and edge \( e \), only one edge and only one triangle in partial co-boundary relations \( R^*_{0,1}(v) \) and \( R^*_{1,2}(e) \), respectively. Thus, \( M_{0,1} = M_{1,2} = 1 \). Conversely, the IG(3D) representation encodes co-boundary relations \( R_{0,1}(v) \) and \( R_{1,2}(e) \). In this case, \( \| R_{0,1} \| \gg 1 \), and \( \| R_{1,2} \| \gg 1 \). For instance, Figure 4.7 provides a simplicial 3-complex, where the IG(3D) data structure encodes five edges in the star of a vertex \( v \), while the IS(3D) data structure encodes only edge \( e \). Furthermore, other similar shapes are shown in Table 4.6: for instance, in the “Cylinder” shape, \( M_{0,1} = 128 \), and \( M_{1,2} = 15 \).

In the IS(3D) data structure, the storage cost of co-boundary relations requires about 30\% of \( S^{3D}_{IS} \). Conversely, the storage cost of boundary relations requires about 70\% of \( S^{3D}_{IS} \), which is equivalent to about 50\% of \( S^{3D}_{IG} \).
Chapter 5

The IA* Data Structure and Adjacency-based Representations

In this chapter, we introduce the Generalized Indexed data structure with Adjacencies (IA*) [CDFW11], one of our contributions in the context of topological data structures. The IA* data structure is a dimension-independent and adjacency-based data structure for representing abstract simplicial complexes. It directly encodes only vertices and top simplices, plus a subset of adjacency relations, restricted to top simplices.

In Section 5.1, we propose the complete design of the IA* data structure, and introduce a graph-based representation for the IA* data structure. We evaluate its storage cost, and provide algorithms for retrieving all the topological relations from the IA* data structure. We prove that the information encoded in the IA* data structure is sufficient to retrieve all the topological relations.

Following [DFH05], adjacency-based data structures have been shown to be the most compact representations, and they have been used in applications, which do not require an explicit encoding of all the simplices. The most widely used data structure for manifold simplicial complexes is the Extended Indexed data structure with Adjacencies (EIA) [DF03], which we briefly discussed in Section 3.1.3. The EIA data structure requires about 60% of 2D incidence-based representations, and about 40% of 3D incidence-based representations [DFH05]. If restricted to manifolds, the IA* data structure reduces to the EIA representation, and, thus, it scales very well.

However, the EIA data structure represents only manifold shapes, thus compact variants for non-manifold shapes are needed. Moreover, an efficient encoding for the star of a simplex is required in order to easily recognize non-manifold singularities. In any adjacency-based data structure, we exploit a compact encoding for the non-manifold adjacency along a simplex, which gives an efficient characterization of non-manifold singularities. To the best of our experience, the IA* data structure can be considered as the extension of the EIA data structure to non-manifold shapes, and it offers an efficient encoding for non-manifold adjacencies.
Other dimension-specific extensions of the EIA data structure have been proposed, namely the Triangle-Segment (TS) data structure [DFMPS04], and the Non-Manifold Indexed data structure with Adjacencies (NMIA) [DFH03]. In Section 5.2, we analyze the TS data structure, which represents arbitrary simplicial 2-complexes. In Section 5.3, we briefly discuss the NMIA data structure, which represents arbitrary simplicial 3-complexes. Both the TS and NMIA data structures are based on the assumption that the input complex is embedded in the Euclidean space $E^3$. These definitions are slightly different than the original proposals, due to the efficient encoding of the non-manifold adjacency, which we propose in this thesis.

Finally, in Section 5.4, we perform experimental comparisons for the TS, NMIA, and IA* data structures in terms of their storage cost and encoded relations. Our tests show that the IA* data structure is more compact than any incidence-based representation, and is even more compact than dimension-specific ones, namely than the TS and NMIA data structures. It is an interesting property, since the latter ones are able to exploit dimension-specific properties of their embedding Euclidean space, like the radial ordering of triangles and tetrahedra around an edge, to reduce their storage requirements. Conversely, the IA* data structure does not require an embedding in any Euclidean space.

In Chapter 6, we propose an implementation of the IA* data structure in our Mangrove Topological Data Structure (Mangrove TDS) framework. We provide a pseudo-code description of algorithms used for retrieving topological relations, and for constructing the IA* data structure. Furthermore, in Chapter 7, we propose a quantitative analysis for performances of several data structures, including the IA*, TS, and NMIA representations, regarding the efficiency of topological relations.

5.1 The Generalized Indexed data structure with Adjacencies

In this section, we introduce one of our contributions in the context of topological data structures, namely the Generalized Indexed data structure with Adjacencies (IA*) [CDFW11]. The IA* data structure is an explicit, dimension-independent, and adjacency-based data structure for representing abstract simplicial complexes, not necessarily embedded in any Euclidean space. The IA* data structure can be considered as the extension to non-manifold shapes of the EIA data structure [DF03], which we discussed in Section 3.1.3. In the remainder of this section, we provide a complete analysis of the IA* data structure.

In Section 5.1.1, we propose the complete design of the IA* data structure. We also define a graph-based representation for the IA* data structure, which we call the IA*-graph. In Section 5.1.2, we provide basic ideas for implementing the IA* data structure, and evaluate its storage cost. We prove that the IA* data structure reduces to the EIA data structure, if restricted to manifolds. In Section 5.1.3, we demonstrate that the information encoded in the IA* data structure is sufficient to retrieve topological relations for simplices in a simplicial complex.
5.1.1 Design of the Data Structure

In this section, we describe the design of the IA* data structure. We also define a graph-based representation for the IA* data structure, which we call the IA*-graph.

The IA* data structure is based on a decomposition of the star of vertices. Given any arbitrary simplicial d-complex Σ, the star of a vertex \( v \) in Σ is decomposed into maximal \( p \)-clusters, with \( 0 < p \leq d \), in the same way as the SIG data structure, discussed in Section 4.3. Recall that a \( p \)-cluster is a \((p−1)\)-connected component of Σ, formed by top \( p \)-simplices. Each \( p \)-cluster in \( St(v) \) is represented by one of its top \( p \)-simplices, arbitrarily selected, which is known as the representative simplex. We denote the list of all the representative \( p \)-simplices, one for each \( p \)-cluster in \( St(v) \), as partial co-boundary relation \( R^*_{0,p}(v) \). For instance, four triangles \( f_1, f_2, f_3, f_4 \) in Figure 5.1 form a 2-cluster incident at vertex 1.

For \( 1 < p \leq d \), we define adjacency relation \( R^*_{p,p}(\sigma) \) of any top \( p \)-simplex \( \sigma \), which consists of all the top \( p \)-simplices adjacent to \( \sigma \). Similarly, for \( 1 < p \leq d \), we define partial co-boundary relation \( R^*_{p−1,p}(\tau) \), for any \((p−1)\)-simplex \( \tau \), which consists of all the top \( p \)-simplices in \( St(\tau) \).

Given any arbitrary simplicial d-complex Σ, the IA* data structure encodes vertices and top \( p \)-simplices in Σ, with \( 0 < p \leq d \), plus the following relations:

- partial boundary relation \( R^*_{p,0}(\sigma) \), with \( 0 < p \leq d \), for each top \( p \)-simplex \( \sigma \): it consists of all the vertices bounding \( \sigma \);
- partial co-boundary relation \( R^*_{0,p}(v) \), with \( 0 < p \leq d \), for any vertex \( v \): it consists of one representative \( p \)-simplex, arbitrarily selected, for each \( p \)-cluster in \( St(v) \);
- adjacency relation \( R^*_{p,p}(\sigma) \), with \( 1 < p \leq d \), for each top \( p \)-simplex \( \sigma \): it consists of all the top \( p \)-simplices sharing a \((p−1)\)-simplex \( \tau \) with \( \sigma \);
- partial co-boundary relation \( R^*_{p−1,p}(\tau) \), with \( 1 < p \leq d \), for any non-manifold \((p−1)\)-simplex \( \tau \): it consists of all the top \( p \)-simplices in \( St(\tau) \).

Figure 5.1 shows an arbitrary shape, discretized by a simplicial 3-complex, which may be represented through the IA* data structure. Each top simplex in this shape is directly represented through its vertices. For instance, tetrahedron \( t_1 \) is formed by vertices \((1, 11, 12, 14)\), and, thus, \( R^*_{4,0}(t_1) = \{1, 12, 13, 14\} \). Top edge \( w \) is incident at vertex \( v \), and, thus, \( R^*_{0,1}(v) = \{w\} \). Again, there are two 2-clusters in \( St(v) \), respectively formed by top triangles \( f_1, f_2, f_3, f_4 \), and of \( f_5, f_6 \), thus, \( R^*_{0,2}(v) = \{f_1, f_2, f_3, f_4\} \). There is also a 3-cluster in \( St(v) \), formed by tetrahedra \( t_1 \) and \( t_2 \), and, thus, \( R^*_{0,3}(v) = \{t_1, t_2\} \). Again, top triangles \( f_1, f_2, f_3, f_4 \), and \( f_5 \) are adjacent along edge \( e \), and, thus, \( R^*_{1,2}(e) = \{f_1, f_2, f_3, f_4\} \). Finally, we can state that \( R^*_{2,2}(f_1) = \{f_2, f_3, f_4\} \), \( R^*_{2,2}(f_2) = \{f_1, f_3, f_4\} \), \( R^*_{2,2}(f_3) = \{f_1, f_2, f_4\} \), and \( R^*_{2,2}(f_4) = \{f_1, f_2, f_3\} \).

Partial co-boundary relation \( R^*_{p−1,p} \), with \( 1 < p \leq d \), simplifies the encoding of partial adjacency relation \( R^*_{p,p} \), especially when several top \( p \)-simplices are adjacent along a common \((p−1)\)-simplex.
A manifold edge, and, thus, $R < p$. Hence, for $1 \leq p \leq d$, adjacency relation $R^*_{p,p}(\sigma)$ along a $(p-1)$-face $\tau$ of $\sigma$ may be expressed, either as a single top $p$-simplex, if $\tau$ is manifold, or through partial co-boundary relation $R_{p-1,p}(\tau)$, if $\tau$ is non-manifold. For instance, in Figure 5.1, top triangles $f_5$ and $f_6$ share a manifold edge, and, thus, $R^*_{2,2}(f_5) = \{f_6\}$, and $R^*_{2,2}(f_6) = \{f_5\}$. Similarly, edge $e$ is shared by four top triangles, namely, $f_1$, $f_2$, $f_3$, and $f_4$: here, $Lk(e)$ is formed by four vertices, one for each top triangle in $St(e)$, and, thus, it is non-manifold. Hence, partial adjacency relation $R^*_{2,2}(f_k)$ of a top triangle $f_k$, with $1 \leq k \leq 4$, is expressed through partial co-boundary relation $R^*_{1,2}(e) = \{f_1, f_2, f_3, f_4\}$, since $R^*_{2,2}(f_k) = R^*_{1,2}(e) \setminus \{f_k\}$.

This solution is very efficient, especially when there are several top $p$-simplices in the star of a $(p-1)$-simplex $\tau$, because partial relation $R^*_{p-1,p}(\tau)$ is encoded only once, regardless the number of top $p$-simplices in $St(\tau)$. Thus, for any top $p$-simplex $\sigma$ in $St(\tau)$, partial adjacency relation $R^*_{p,p}(\sigma)$ along $\tau$ is retrieved by removing $\sigma$ from the list of simplices in $R^*_{p-1,p}(\tau)$.

Hence, for $1 < p \leq d$, we characterize non-manifold $(p-1)$-faces $\tau$ of any top $p$-simplex $\sigma$, by encoding partial adjacency relation $R^*_{p,p}(\sigma)$ along $\tau$ through partial co-boundary relation $R^*_{1,p}(\tau)$. In any case, we can exploit this solution if and only if the star of a non-manifold $(p-1)$-simplex contains at least one top $p$-simplex. For instance, in Figure 5.2(a), edge $e$ is surely non-manifold, because its star is formed by several clusters of different dimensions. Here, only top triangle $df$ belongs to $St(e)$, and, thus, $R^*_{2,2}(df)$ is empty. In any case, we can characterize non-manifold edge $e$ through partial co-boundary relation $R^*_{1,2}(e) = \{df\}$. Conversely, in Figure 5.2(b), there are no

$\tau$. In this case, $\tau$ is non-manifold, because $Lk(\tau)$ contains a vertex, for each top $p$-simplex in $St(\tau)$. Conversely, if $\tau$ is manifold, then at most two top $p$-simplices belong to $St(\tau)$. As a consequence, given a top $p$-simplex $\sigma$, with $1 < p \leq d$, adjacency relation $R^*_{p,p}(\sigma)$ along a $(p-1)$-face $\tau$ of $\sigma$ may be expressed, either as a single top $p$-simplex, if $\tau$ is manifold, or through partial co-boundary relation $R^*_{p-1,p}(\tau)$, if $\tau$ is non-manifold. For instance, in Figure 5.1, top triangles $f_5$ and $f_6$ share a manifold edge, and, thus, $R^*_{2,2}(f_5) = \{f_6\}$, and $R^*_{2,2}(f_6) = \{f_5\}$. Similarly, edge $e$ is shared by four top triangles, namely, $f_1$, $f_2$, $f_3$, and $f_4$: here, $Lk(e)$ is formed by four vertices, one for each top triangle in $St(e)$, and, thus, it is non-manifold. Hence, partial adjacency relation $R^*_{2,2}(f_k)$ of a top triangle $f_k$, with $1 \leq k \leq 4$, is expressed through partial co-boundary relation $R^*_{1,2}(e) = \{f_1, f_2, f_3, f_4\}$, since $R^*_{2,2}(f_k) = R^*_{1,2}(e) \setminus \{f_k\}$.

This solution is very efficient, especially when there are several top $p$-simplices in the star of a $(p-1)$-simplex $\tau$, because partial relation $R^*_{p-1,p}(\tau)$ is encoded only once, regardless the number of top $p$-simplices in $St(\tau)$. Thus, for any top $p$-simplex $\sigma$ in $St(\tau)$, partial adjacency relation $R^*_{p,p}(\sigma)$ along $\tau$ is retrieved by removing $\sigma$ from the list of simplices in $R^*_{p-1,p}(\tau)$.

Hence, for $1 < p \leq d$, we characterize non-manifold $(p-1)$-faces $\tau$ of any top $p$-simplex $\sigma$, by encoding partial adjacency relation $R^*_{p,p}(\sigma)$ along $\tau$ through partial co-boundary relation $R^*_{1,p}(\tau)$. In any case, we can exploit this solution if and only if the star of a non-manifold $(p-1)$-simplex contains at least one top $p$-simplex. For instance, in Figure 5.2(a), edge $e$ is surely non-manifold, because its star is formed by several clusters of different dimensions. Here, only top triangle $df$ belongs to $St(e)$, and, thus, $R^*_{2,2}(df)$ is empty. In any case, we can characterize non-manifold edge $e$ through partial co-boundary relation $R^*_{1,2}(e) = \{df\}$. Conversely, in Figure 5.2(b), there are no

$\tau$. In this case, $\tau$ is non-manifold, because $Lk(\tau)$ contains a vertex, for each top $p$-simplex in $St(\tau)$. Conversely, if $\tau$ is manifold, then at most two top $p$-simplices belong to $St(\tau)$. As a consequence, given a top $p$-simplex $\sigma$, with $1 < p \leq d$, adjacency relation $R^*_{p,p}(\sigma)$ along a $(p-1)$-face $\tau$ of $\sigma$ may be expressed, either as a single top $p$-simplex, if $\tau$ is manifold, or through partial co-boundary relation $R^*_{p-1,p}(\tau)$, if $\tau$ is non-manifold. For instance, in Figure 5.1, top triangles $f_5$ and $f_6$ share a manifold edge, and, thus, $R^*_{2,2}(f_5) = \{f_6\}$, and $R^*_{2,2}(f_6) = \{f_5\}$. Similarly, edge $e$ is shared by four top triangles, namely, $f_1$, $f_2$, $f_3$, and $f_4$: here, $Lk(e)$ is formed by four vertices, one for each top triangle in $St(e)$, and, thus, it is non-manifold. Hence, partial adjacency relation $R^*_{2,2}(f_k)$ of a top triangle $f_k$, with $1 \leq k \leq 4$, is expressed through partial co-boundary relation $R^*_{1,2}(e) = \{f_1, f_2, f_3, f_4\}$, since $R^*_{2,2}(f_k) = R^*_{1,2}(e) \setminus \{f_k\}$.

This solution is very efficient, especially when there are several top $p$-simplices in the star of a $(p-1)$-simplex $\tau$, because partial relation $R^*_{p-1,p}(\tau)$ is encoded only once, regardless the number of top $p$-simplices in $St(\tau)$. Thus, for any top $p$-simplex $\sigma$ in $St(\tau)$, partial adjacency relation $R^*_{p,p}(\sigma)$ along $\tau$ is retrieved by removing $\sigma$ from the list of simplices in $R^*_{p-1,p}(\tau)$.

Hence, for $1 < p \leq d$, we characterize non-manifold $(p-1)$-faces $\tau$ of any top $p$-simplex $\sigma$, by encoding partial adjacency relation $R^*_{p,p}(\sigma)$ along $\tau$ through partial co-boundary relation $R^*_{1,p}(\tau)$. In any case, we can exploit this solution if and only if the star of a non-manifold $(p-1)$-simplex contains at least one top $p$-simplex. For instance, in Figure 5.2(a), edge $e$ is surely non-manifold, because its star is formed by several clusters of different dimensions. Here, only top triangle $df$ belongs to $St(e)$, and, thus, $R^*_{2,2}(df)$ is empty. In any case, we can characterize non-manifold edge $e$ through partial co-boundary relation $R^*_{1,2}(e) = \{df\}$. Conversely, in Figure 5.2(b), there are no

\[
\begin{align*}
v &= 1 \\
w &= (1, 2) \\
e &= (1, 3) \\
f_1 &= (1, 3, 4) \\
f_2 &= (1, 3, 5) \\
f_3 &= (1, 3, 6) \\
f_4 &= (1, 3, 7) \\
f_5 &= (1, 8, 9) \\
f_6 &= (1, 9, 10) \\
t_1 &= (1, 11, 12, 14) \\
t_2 &= (1, 12, 13, 14)
\end{align*}
\]
top triangles incident at non-manifold edge \( e \). In this case, partial co-boundary relation \( R_{1,2}^*(e) \) is empty, and we do not encode any information about edge \( e \).

![Figure 5.2](image-url)

Figure 5.2: Non-manifold edges in simplicial 3-complexes, described through the IA* data structure. These edges may be characterized through their partial co-boundary relation \( R_{1,2}^* \). (a) Here, only a top triangle \( df \) is incident at a non-manifold edge \( e = (v_1, v_2) \), and, thus, we characterize \( e \) as \( R_{1,2}^*(e) = \{ df \} \). (b) If there are no top triangles in the star of a non-manifold edge \( e \), then we cannot exploit partial co-boundary relation \( R_{1,2}^* \) to characterize \( e \).

The IA* data structure can be described through a graph-based representation \( G = (\mathcal{N}, \mathcal{A}) \), which we call the IA*-graph. Each node in \( \mathcal{N} \) corresponds either to a vertex, or, to a top simplex, or, for \( 1 < p \leq d \), to a non-manifold \((p-1)\)-simplex \( \tau \), shared by at least one top \( p \)-simplex. Each arc in \( \mathcal{A} \) corresponds to a topological relation for the simplices described by nodes in the IA*-graph. Figure 5.3 shows the IA*-graph corresponding to the simplicial 3-complex in Figure 5.1.

An arc \((\sigma, v)\) is said to be a boundary arc, if it connects the nodes related to a top \( p \)-simplex \( \sigma \), with \( 0 < p \leq d \), and to a vertex \( v \), such that \( v \in R_{p,0}^*(\sigma) \). Note that relations \( R_{p,0}^* \) and \( R_{0,p}^* \) are not symmetric. We define the IA* boundary graph as the spanning subgraph of the IA*-graph \( G \), formed by nodes related to vertices and top simplices, plus boundary arcs. Figure 5.3(b) shows the IA* boundary graph for the simplicial 3-complex in Figure 5.1. Here, node and arcs in red represent, respectively, top triangle \( f_1 = (1, 3, 4) \), and its boundary relation \( R_{2,0}^*(f_1) \). Then, node and arcs in blue represent, respectively, top triangle \( f_5 = (1, 8, 9) \), and its boundary relation \( R_{2,0}^*(f_5) \). Finally, node and arcs in green represent, respectively, tetrahedron \( t_1 = (1, 11, 12, 14) \), and its boundary relation \( R_{3,0}^*(t_1) \).

An arc \((v, \sigma)\) is said to be a co-boundary arc, if it connects the nodes related to a vertex \( v \), and to a top \( p \)-simplex, with \( 0 < p \leq d \), such that \( \sigma \in R_{0,p}^*(v) \). We define the IA* co-boundary graph as the spanning subgraph of the IA*-graph \( G \), formed by nodes related to vertices and top simplices, plus co-boundary arcs. Figure 5.3(c) shows the IA* co-boundary graph for vertices of the simplicial 3-complex in Figure 5.1. Here, nodes shown as dotted boxes correspond to representative simplices for clusters in the star of vertex \( v = 1 \). There are four co-boundary arcs, outgoing from \( v \), one for each cluster in \( St(v) \). Specifically, a co-boundary arc connects \( v \) with top edge \( w = (1, 2) \), namely a 1-cluster, in \( St(v) \). Co-boundary arc in red denotes a 2-cluster in \( St(v) \), represented by top triangle \( f_1 = (1, 3, 4) \). Then, co-boundary arc in blue denotes another 2-cluster in \( St(v) \), represented by top triangle \( f_5 = (1, 8, 9) \). Finally, co-boundary arc in green denotes a 3-cluster in \( St(v) \), represented by tetrahedron \( t_1 = (1, 11, 12, 14) \).
An arc \((\sigma, \sigma')\) is said to be a *manifold adjacency arc*, if it connects nodes related, for \(1 < p \leq d\), to two top \(p\)-simplices \(\sigma\) and \(\sigma'\), which share a manifold \((p-1)\)-simplex. Note that also the symmetric arc \((\sigma', \sigma)\) exists. Finally, an arc \((\tau, \sigma)\) is said to be a *non-manifold adjacency arc*, if it connects nodes related, for \(1 < p \leq d\), to a non-manifold \((p-1)\)-simplex \(\tau\), and a top \(p\)-simplex \(\sigma\), such that \(\sigma \in R^\star_{p-1,p}(\tau)\). We define the IA* adjacency graph as the spanning subgraph of the IA* graph \(G\), formed by nodes related, for \(1 < p \leq d\), to top \(p\)-simplices, and to non-manifold \((p-1)\)-simplices \(\tau\), such that partial co-boundary relation \(R^\star_{p-1,p}(\tau)\) is not empty, plus manifold and non-manifold adjacency arcs. Figure 5.3(d) shows the IA* adjacency graph for simplicial 3-complex in Figure 5.1. Here, two pairs of manifold adjacency arcs are shown. The first pair of arcs connects top triangles \(f_5\) and \(f_6\) in one of the 2-clusters in \(St(v)\), corresponding to the blue box. The second pair of arcs connects tetrahedra \(t_1\) and \(t_2\) in the 3-cluster in \(St(v)\), corresponding to the green box. Moreover, there are four non-manifold adjacency arcs, which connect non-manifold edge \(e\) and top triangles \(f_1\), \(f_2\), \(f_3\), and \(f_4\). These top triangles form an other 2-cluster in \(St(v)\), corresponding to the red box. In this way, we can recognize non-manifold edge \(e\) in constant time.

Generally speaking, if a vertex is manifold, then its star is formed by only one cluster. The reverse is not true: even if the star of a vertex may be formed by only one cluster, it might be a non-manifold simplex. For instance, vertex 3 in Figure 5.1 is non-manifold, still its star is formed by only one cluster.

Figure 5.3: (a) The simplicial 3-complex already shown in Figure 5.1, and its corresponding IA*-graph. For the sake of clarity, the (b) IA* boundary graph, (c) the IA* co-boundary graph, and (d) the IA* adjacency graph are separately shown.

Generally speaking, if a vertex is manifold, then its star is formed by only one cluster. The reverse is not true: even if the star of a vertex may be formed by only one cluster, it might be a non-manifold simplex. For instance, vertex 3 in Figure 5.1 is non-manifold, still its star is formed by only one cluster.
5.1.2 Implementation and Storage Cost

In this section, we provide basic ideas for implementing the IA* data structure, and we evaluate its storage cost. Specifically, we demonstrate that the IA* data structure, if restricted to manifolds, reduces to the EIA representation [DF03], discussed in Section 3.1.3.

In order to represent a simplicial d-complex $\Sigma$ through the IA* data structure, we exploit the same approach described in [AJ05]. Here, we encode a topological relation $R$ as a table with one record for each simplex involved in $R$. We assign a unique index to each simplex $\sigma$ in $\Sigma$, namely an integer value, to be used for accessing records related to $\sigma$ in each table.

The IA* data structure encodes, for $0 < p \leq d$, each partial boundary relation $R^*_p,0$ as a table. Each record in these tables describes one top $p$-simplex $\sigma$. Here, $\deg(\sigma) = p + 1$. Thus, partial boundary relation $R^*_{p,0}(\sigma)$ is formed by $p + 1$ vertices bounding $\sigma$. Hence, we need $p + 1$ indices for encoding boundary relation $R^*_{p,0}(\sigma)$, and, thus, the storage cost of partial boundary relations for top simplices in $\Sigma$ is equal to:

$$\sum_{p=1}^{d} (p + 1)s^t_p$$

where $s^t_p$ is the number of top $p$-simplices in $\Sigma$.

The IA* data structure encodes, for $0 < p \leq d$, each partial co-boundary relation $R^*_0,p$ in a table. Each record in these tables corresponds to a vertex $v$ in $\Sigma$, and contains $k^p(v)$ indices for all the representative simplices in partial co-boundary relation $R^*_{0,p}(v)$. Thus, the total number $K^p_0$ of $p$-clusters for vertices in $\Sigma$, with $0 < p \leq d$, is equal to:

$$K^p_0 = \sum_{v \in \Sigma^0} k^p(v)$$

where $\Sigma^0$ is the collection of vertices in $\Sigma$.

The IA* data structure also encodes, for $1 < p \leq d$, partial co-boundary relation $R^*_{p-1,p}$ as a table. Each record in these tables corresponds to a non-manifold $(p-1)$-simplex $\tau$, such that at least one top $p$-simplex is incident in $\tau$, as discussed in Section 5.1.1. Hence, each record stores several indices for top $p$-simplices in $St(\tau)$. Note that we store only one record for each non-manifold simplex $\tau$. Let $N^p_\tau$ be the number of top $p$-simplices incident at a non-manifold $(p-1)$-simplex $\tau$ in $\Sigma$, then the storage cost of partial co-boundary relation $R^*_{p-1,p}$ is equal to:

$$\sum_{p=2}^{d} \sum_{\tau \in \Sigma^{p-1}} N^p_\tau$$

where $\Sigma^{p-1}$ is the collection of $(p-1)$-simplices in $\Sigma$.

Finally, the IA* data structure also encodes, for $1 < p \leq d$, each partial adjacency relation $R^*_{p,p}$ as a table. Each record in these tables corresponds to one top $p$-simplex $\sigma$ in $\Sigma$, and contains one index for each $(p-1)$-face of $\sigma$, as discussed in Section 5.1.1. Specifically, for a non-manifold
(p - 1)-face \( \tau \), such that \( R_{p-1,p}^*(\tau) \) is not empty, we store the index of \( \tau \) in the table describing partial co-boundary relation \( R_{p-1,p}(\sigma) \). Hence, we need \((p + 1)\) indices for encoding partial adjacency relation \( R_{p,p}(\sigma) \), and, thus, the storage cost of partial adjacency relation for all the top simplices in \( \Sigma \) is equal to:

\[
\sum_{p=2}^{d} (p + 1)s_p^d
\]

As a consequence, the storage cost \( S_{IA^*} \) of the IA* data structure is equal to:

\[
S_{IA^*} = 2s_1^d + 2 \sum_{p=2}^{d} (p + 1)s_p^d + \sum_{p=1}^{d} K_p^d + \sum_{p=2}^{d} \sum_{\tau \in \Sigma^{p-1}} N_p^\tau
\]  

(5.1)

If the input simplicial \( d \)-complex \( \Sigma \) is regular, then all the top \( p \)-simplices in \( \Sigma \) are maximal. As a consequence, the storage cost \( S_{IA^*} \) becomes:

\[
S_{IA^*} = 2(d + 1)s_d + K_0^d + \sum_{\tau \in \Sigma^{d-1}} N_\tau^d
\]

The IA* data structure scales well to manifolds. If the input simplicial \( d \)-complex \( \Sigma \) is manifold, then all the top \( p \)-simplices in \( \Sigma \) are maximal, and \( k^d(v) = 1 \), for all the vertices in \( \Sigma \). Clearly, there are not any non-manifold singularities. Thus, the storage cost \( S_{IA^*} \) becomes:

\[
S_{IA^*} = s_0 + 2(d + 1)s_d
\]

Hence, the IA* data structure, restricted to manifolds, reduces to the EIA data structure [DF03], which we discussed in Section 3.1.3.

### 5.1.3 Retrieving Topological Relations

In this section, we demonstrate that the information encoded in the IA* data structure is sufficient to retrieve topological relations for simplices in a simplicial complex. In Section 5.1.3.1, we propose an algorithm for retrieving boundary relations, while, in Section 5.1.3.2, we discuss an algorithm for retrieving co-boundary relations. Finally, in Section 5.1.3.3, we describe an algorithm for retrieving adjacency relations. In these algorithms, we assume to explicitly represent a simplex not directly encoded in the IA* data structure through its vertices.

#### 5.1.3.1 Retrieving Boundary Relations

In this section, given a simplicial \( d \)-complex \( \Sigma \), we propose an algorithm for retrieving boundary relations \( R_{p,q}(\sigma) \), with \( p > q \), for any \( p \)-simplex \( \sigma \) in \( \Sigma \), with \( 0 < p \leq d \).

In this algorithm, we need to know which vertices bound a \( p \)-simplex \( \sigma \). In the IA* data structure, for any top \( p \)-simplex \( \sigma \) in \( \Sigma \), partial boundary relation \( R_{p,0}^*(\sigma) \) is directly encoded. Conversely, a non-top \( p \)-simplex in \( \Sigma \) is explicitly represented through its vertices. Thus, we can generate all the \( q \)-faces of \( \sigma \), expressed in terms of their vertices. For instance, we can recursively exploit
the construction schema proposed in [DFHPC10]. Given a \( p \)-simplex \( \sigma \), generated by vertices \([v_0, \ldots, v_i, \ldots, v_p] \), a \((p-1)\)-face \( \lambda_i \) of \( \sigma \), with \( i = 0, \ldots, p \), is formed by vertices \([v_0, \ldots, \bar{v}_i, \ldots, v_p] \), where we discard vertex \( v_i \).

Time complexity of this algorithm is linear in the number of simplices in boundary relation \( R_{p,q} \), with \( p > q \). Recall that a \( p \)-simplex \( \sigma \) has \( s_k^p = \binom{p+1}{k} \) faces of dimension \( k \), with \( 0 \leq k < p \) [Eden87]. In this algorithm, we visit all the \( k \)-faces of \( \sigma \), for \( q < k < p \), and their number is:

\[
\sum_{k=q}^{p-1} \binom{p+1}{k+1}
\]

This number can be expressed as a constant \( C_{p,q} \), which depends only on \( p \) and \( q \). For instance, a triangle has six faces, namely three vertices and three edges. Therefore, the retrieval of boundary relation \( R_{p,q}(\sigma) \) for a \( p \)-simplex \( \sigma \) has a time complexity in \( \mathcal{O}(1) \), thus it is optimal.

### 5.1.3.2 Retrieving Co-boundary Relations

In this section, given a simplicial \( d \)-complex \( \Sigma \), we describe an algorithm for retrieving co-boundary relations \( R_{p,q}(\sigma) \), with \( p < q \), for any \( p \)-simplex \( \sigma \) in \( \Sigma \), with \( 0 \leq p < d \).

First, we consider the retrieval of all the top \( k \)-simplices, with \( 1 \leq k \leq d \), incident at a vertex \( v \) in \( \Sigma \). Note that top edges in \( St(v) \) are directly encoded in partial co-boundary relation \( R_{0,1}^*(\sigma) \). If \( k \neq 1 \), then we perform a breadth-first traversal of each \( k \)-cluster in \( St(v) \), represented by a top \( k \)-simplex \( \sigma \) in \( R_{0,k}^* \). Top \( k \)-simplices in a \( k \)-cluster, represented by \( \sigma \), can be retrieved through the transitive closure of partial adjacency \( R_{k,k}^* \), restricted to top \( k \)-simplices, starting from \( \sigma \). If the adjacency along a \((k-1)\)-face of a top \( k \)-simplex \( \psi \) is non-manifold, then top \( k \)-simplices adjacent to \( \psi \) can be retrieved through partial co-boundary relation \( R_{p-1,p}(\tau) \). Clearly, the time complexity of this operation is in \( \mathcal{O}(v^*_c) \), where \( v^*_c \) is the number of top simplices in \( St(v) \), since relations \( R_{0,k}^*, R_{0,1}^*, R_{k-1,k}^*, \) and \( R_{k,k}^* \) are directly encoded in the IA* data structure. Thus, this operation is optimal.

Figure 5.4 shows how it is possible to retrieve top simplices incident at vertex \( v = 1 \) in the simplicial 3-complex in Figure 5.1. Note that \( R_{0,1}^* = \{w\}, R_{0,2}^* = \{f_1, f_5\}, \) and \( R_{0,3}^* = \{t_1\} \). First, we retrieve top edge \( w = (1, 2) \), as shown in Figure 5.4(a). Then, we visit the first 2-cluster in \( St(v) \) (in red), represented by top triangle \( f_1 = (1, 3, 4) \). This 2-cluster is formed by top triangles \( f_1, f_2, f_3, \) and \( f_4 \), which share non-manifold edge \( e = (1, 3) \). Thus, we retrieve these top triangles by partial co-boundary relation \( R_{1,2}^* \), as shown in Figure 5.4(b). We visit the second 2-cluster in \( St(v) \) (in blue), represented by top triangle \( f_5 = (1, 8, 9) \), through partial adjacency relation \( R_{2,2}^* \), as shown in Figure 5.4(c). Here, top triangles \((1, 8, 9)\) and \((1, 9, 10)\) share a manifold edge \((1, 9)\), thus adjacency relation \( R_{2,2}^* \) is directly encoded. Finally, we visit the 3-cluster in \( St(v) \) (in green), represented by tetrahedron \( t_1 = (1, 11, 12, 14) \), as shown in Figure 5.4(d). This 3-cluster is formed by tetrahedra \((1, 11, 12, 14)\) and \((1, 12, 13, 14)\), which share a manifold triangle \((1, 12, 14)\).

A \( p \)-simplex, with \( 1 \leq p \leq d \), which is incident at any vertex \( v \) in \( \Sigma \), may be either a top \( p \)-simplex,
or a $p$-face of a top $h$-simplex in $St(v)$, with $h > p$. Thus, co-boundary relation $R_{0,p}(v)$ can be retrieved by identifying top $h$-simplices in $St(v)$, with $p < h \leq d$, and selecting all the $p$-faces in $St(v)$, which belong to the boundary of these $h$-simplices. These faces are expressed in terms of their vertices. As demonstrated in Section 5.1.3.1, the number of $p$-faces for a $h$-simplex is given by $s^h_p = \binom{h+1}{p+1}$, which is a constant value depending only on $h$ and $p$. A $p$-face is incident at vertex $v$, if $v$ belongs to its boundary. Thus, the time complexity of the algorithm, which extracts co-boundary relation $R_{0,p}(v)$, is $O(v^t_v)$, where $v^t_v$ is the number of top simplices in $St(v)$.

Any co-boundary relation $R_{0,k}$, with $0 < k \leq d$, is needed to retrieve the star of a $p$-simplex $\sigma$, not directly encoded in the IA* data structure. Here, we assume that $\sigma$ is explicitly expressed in terms of its vertices $[v_0, \ldots, v_p]$. Thus, co-boundary relation $R_{p,q}(\sigma)$, with $p < q$, is formed by either top $q$-simplices, or $q$-faces of top $h$-simplices, with $h > q$, which are incident at $\sigma$. Thus, given a vertex $v$ on the boundary of $\sigma$, arbitrarily selected, co-boundary relation $R_{p,q}(\sigma)$ can be retrieved by identifying top $h$-simplices, with $h \geq q$, incident at $v$, and by selecting top $q$-simplices and $q$-faces of these simplices, which are incident at $\sigma$. As a consequence, the time complexity for retrieving co-boundary relation $R_{p,q}(\sigma)$ is linear in the number of $h$-simplices, with $q \leq h \leq d$, incident at one of the vertices bounding $\sigma$. Thus, co-boundary relation $R_{p,q}$ is local in the IA* data structure.

In the IA* data structure, most of co-boundary relations $R_{p,q}(\sigma)$ of a $p$-simplex $\sigma$ are local, and their complexity is linear in the number of top simplices incident at any vertex bounding $\sigma$. As a consequence, they are optimal only for simplicial complexes embedded in $\mathbb{E}^3$. 

Figure 5.4: (a-d) Several steps needed for retrieving top simplices incident at vertex $v = 1$ of the simplicial 3-complex in Figure 5.1. The key idea is to expand, for $1 < k \leq 3$, each $k$-cluster in the star of $v$ through partial adjacency relation $R^*_k,k$. 

88
5.1.3.3 Retrieving Adjacency Relations

In this section, we describe an algorithm for retrieving adjacency relation of any simplex from the IA* data structure. In particular, given a simplicial $d$-complex $\Sigma$, the challenge is to retrieve, for any $p$-simplex $\sigma$, with $0 \leq p \leq d$, adjacency relation $R_{p,p}(\sigma)$. Recall that two vertices are adjacent if they are connected by a common edge, and two $p$-simplices, with $0 < p \leq d$, are adjacent if they share a common $(p-1)$-simplex.

The IA* data structure directly encodes adjacency relation $R_{d,d}$ for any $d$-simplex in $\Sigma$. It encodes, for $1 < k < d$, partial adjacency relation $R^{*}_{k,k}(\sigma')$, restricted to top $k$-simplices, adjacent to a top $k$-simplex $\sigma'$. As discussed in Section 5.1.1, given a manifold $(k-1)$-face $\tau$ of $\sigma'$, the top $k$-simplex adjacent to $\sigma'$ along $\tau$ is directly encoded. Otherwise, if $\tau$ is non-manifold, then we can retrieve top $k$-simplices adjacent to $\sigma'$ along $\tau$ through partial co-boundary relation $R^{*}_{k-1,k}(\tau)$. The time complexity for retrieving adjacency relation $R^{*}_{k,k}(\sigma')$ is linear in the number of top $k$-simplices adjacent to $\sigma'$, and, thus, it is optimal.

For any top $p$-simplex $\sigma$ in $\Sigma$, with $1 < p \leq d$, adjacency relation $R_{p,p}(\sigma)$ can be retrieved in two steps. In the first step, we retrieve top $p$-simplices adjacent to $\sigma$ through partial adjacency relation $R^{*}_{p,p}(\sigma)$. The time complexity of this operation is linear in the number of top $p$-simplices adjacent to $\sigma$. In the second step, we retrieve non-top $p$-simplices adjacent to $\sigma$. Note that these simplices are not directly encoded in the IA* data structure, and they are explicitly represented through their vertices. First, we retrieve $(p-1)$-simplices $\tau$ bounding $\sigma$, thus, as discussed in Section 5.1.3.1, the time complexity of this operation is $O(1)$. At this point, we extract co-boundary relation $R_{p-1,p}$, for each $(p-1)$-face $\tau$ bounding $\sigma$. As demonstrated in Section 5.1.3.2, the time complexity of this operation is linear in the number of top simplices incident at a vertex of $\tau$, arbitrarily selected.

As a consequence, for $1 < p \leq d$, adjacency relation $R_{p,p}$ is local in the IA* data structure.

For any $p$-simplex $\sigma$ in $\Sigma$ not directly encoded in the IA* data structure, adjacency relation $R_{p,p}(\sigma)$ can be retrieved by combining relations $R_{p,p-1}$ and $R_{p-1,p}$, as performed for retrieving non-top $p$-simplices adjacent to a top $p$-simplex. As a consequence, the time complexity of this operation is dominated by the retrieval of co-boundary relation $R_{p-1,p}$ for a $(p-1)$-face of $\sigma$.

Adjacency relation $R_{0,0}(\sigma)$ of a vertex $\sigma$ in $\Sigma$ can be retrieved in two steps. In the first step, we retrieve, for $1 \leq h \leq d$, top $h$-simplices incident at $\sigma$, as discussed in Section 5.1.3.2. In the second step, for each top $h$-simplex $\psi$ in $St(\sigma)$, we select vertices of $\psi$, which are different from $\sigma$. The time complexity of this operation is linear in the number of top simplices in $St(\sigma)$, and, thus, adjacency relation $R_{0,0}$ is local in the IA* data structure.

For any top edge $\sigma = (v_1, v_2)$ in $\Sigma$, we do not encode any adjacency relation, and, thus, we need to exploit co-boundary relation $R_{0,1}$. In particular, adjacency relation $R_{1,1}(\sigma)$ can be retrieved as the union of all the edges in $R_{0,1}(v_1)$ and in $R_{0,1}(v_2)$. The time complexity of this operation is linear in the number of top simplices in $St(v_1)$, plus the number of top simplices in $St(v_2)$. Thus, adjacency relation $R_{1,1}$ is local in the IA* data structure.
In the IA* data structure, most of adjacency relations \( R_{p,p}(\sigma) \) of a \( p \)-simplex \( \sigma \) are local, and their time complexity is linear in the number of top simplices incident at a vertex bounding \( \sigma \). As a consequence, they are optimal only for simplicial complexes embedded in \( \mathbb{E}^3 \).

### 5.2 The Triangle-Segment Data Structure

In this section, we define a slight modification of the *Triangle Segment (TS)* data structure, proposed in [DFMPS04]. The TS representation is an explicit, and adjacency-based data structure, specific for arbitrary simplicial 2-complexes embedded in the Euclidean space \( \mathbb{E}^3 \), known as *triangle segment* meshes. It is one of the first attempts to generalize the EIA data structure to arbitrary simplicial 2D complexes. A full implementation for the first proposal of the TS data structure is available in the public domain [AIM04].

The TS data structure can represent any arbitrary simplicial 2-complex embedded in the Euclidean space \( \mathbb{E}^3 \), because it exploits the radial sorting of triangles around a non-manifold edge, in order to reduce the amount of information encoded. Our modification is based on a compact encoding for the *non-manifold adjacency* along an edge, and gives an efficient characterization of non-manifold singularities. We decompose the star of any vertex into maximal 1- and 2-clusters, by exploiting the same approach used in Section 4.4.2 with the SIG(2D) data structure.

Given any arbitrary simplicial 2-complex \( \Sigma \), the TS data structure encodes vertices, top edges (top 1-simplices), and triangles in \( \Sigma \), plus the following topological relations:

- partial boundary relation \( R^*_p,0(\sigma) \), for each top \( p \)-simplex \( \sigma \), with \( p \leq 2 \), which consists of all the vertices bounding \( \sigma \);
- partial co-boundary relation \( R^*_0,p(v) \), for each vertex \( v \), with \( p \leq 2 \), which consists of one representative simplex, arbitrarily selected, for each \( p \)-cluster in \( St(v) \);
- adjacency relation \( R_{2,2}(t) \), for each triangle \( t \), which consists of all the triangles adjacent to \( t \) along its three edges;
- partial co-boundary relation \( R^*_{1,2}(\tau) \), which is defined for each edge \( \tau \) belonging to the boundary of more than two triangles. For each triangle \( t \) in \( St(\tau) \), it encodes the triangles, which immediately precede and follow \( t \) in counter-clockwise order around \( \tau \).

Partial relation \( R^*_{1,2} \) allows for a compact encoding of adjacency relation \( R_{2,2} \), specifically when there are more than two triangles adjacent to a triangle \( t \) along an edge \( \tau \) of \( t \). As discussed in Section 2.3, in a regular 1-connected 2-complex, an edge \( \tau \) is manifold, if there are at most two triangles in \( St(\tau) \). This incidence relation is called the *manifold adjacency* along \( \tau \). Note that a simplicial 2-complex \( \Sigma \) can be decomposed in regular 1-connected subcomplexes \( \Sigma' \), given by maximal 2-clusters in \( \Sigma \). Adjacency relation \( R_{2,2}(t) \) along an edge \( \tau \) is encoded as either one triangle adjacent to the triangle \( t \), if \( \tau \) is manifold, or as a reference to two triangles in \( R^*_{1,2}(\tau) \), if
is non-manifold. Figure 5.5 shows partial co-boundary relation \( R_{1,2}^\ast (\tau) \) of a non-manifold edge \((v, w)\) with respect to a triangle \( t \). In this case \( R_{1,2}^\ast (\tau) = \{ t_2, t_1 \} \), where triangles \( t_1 \) and \( t_2 \) follow and precede \( t \) in counter-clockwise order around edge \((v, w)\), respectively.

![Figure 5.5: An example of partial co-boundary relation \( R_{1,2}^\ast (\tau) \) for a non-manifold edge \( \tau = (v, w) \) bounding a triangle \( t \), and shared among several triangles. In this context, \( R_{1,2}^\ast (\tau) = \{ t_2, t_1 \} \), where \( t_1 \) and \( t_2 \) are the triangles, which follow and precede \( t \) in counter-clockwise order around \( \tau \), respectively. Figure courtesy of [DFMPS04].](image)

As a consequence, we associate a Euclidean point in \( E^3 \) with each vertex in \( \Sigma \) in order to radially sort triangles around a non-manifold edge \( \tau = (v_1, v_2) \). This operation is equivalent to radially sort vertices different than \( v_1 \) and \( v_2 \) of triangles in \( St(\tau) \) around edge \( \tau \), as proposed in [PS85].

In order to represent a simplicial 2-complex \( \Sigma \) through the TS data structure, we exploit the same approach described in [AJ05], used for encoding the IA* data structure. For the sake of clarity, we encode auxiliary records for describing non-manifold edges in \( \Sigma \). We assign a unique index to vertices, top edges, triangles, and non-manifold edges in \( \Sigma \). The unique index of a simplex \( \sigma \), namely an integer value, is used for accessing records related to \( \sigma \) in each table.

We represent the Euclidean points in \( E^3 \) associated with each vertex in \( \Sigma \) through an array \( V \) with \( s_0 \) locations, one for each vertex in the collection \( \Sigma^0 \) of vertices. Its storage cost is \( 3s_0 \) floating point values. In the remainder of this section, we consider only the storage cost related to topological relations.

We represent partial boundary relations \( R_{1,0}^\ast \) and \( R_{2,0}^\ast \) as two tables, one for each relation. Each record in these tables represents, for \( 0 < p \leq 2 \), a top \( p \)-simplex, defined by \( p + 1 \) indices of its vertices. As a consequence, the number of indices needed to encode the partial boundary relations for top simplices in \( \Sigma \) is \( 2s_1^t + 3s_2 \), where \( s_1^t \) and \( s_2 \) are the number of, respectively, top edges and triangles in \( \Sigma \). We do not encode any boundary relation for a non-manifold edge in \( \Sigma \).

We also encode partial co-boundary relations \( R_{0,1}^\ast \) and \( R_{0,2}^\ast \) as two tables, one for each relation. Each record in these tables corresponds to any vertex \( v \) in \( \Sigma \), and contains indices of all the representative simplices in the partial co-boundary relations \( R_{0,1}^\ast (v) \) and \( R_{0,2}^\ast (v) \). Hence, the number of indices needed to encode the partial co-boundary relations for all the vertices in \( \Sigma \) is \( 2s_1^t + K_0^2 \), where \( K_0^2 \) denotes the total number of 2-clusters incident at vertices \( \Sigma \).

We encode partial co-boundary relation \( R_{1,2}^\ast \) as one table, where each record corresponds to a
non-manifold edge $\tau$, and stores indices of two triangles in $R_{1,2}^*(\tau)$, with respect to a triangle $t$ in $St(\tau)$. Note that a record related to a non-manifold edge $\tau$ must be replicated for each triangle in $St(\tau)$. Let $N_1^2\tau$ be the number of non-manifold edges in a triangle $t$ in $\Sigma$, then the total number of indices needed to encode partial co-boundary relation $R_{1,2}^*$ for all the triangles in $\Sigma$ is:

$$2 \sum_{t \in \Sigma^2} N_1^2\tau$$

where $\Sigma^2$ is the collection of triangles in $\Sigma$.

We represent adjacency relation $R_{2,2}$ as a table, where each record corresponds to a triangle $t$ in $\Sigma$, and stores indices of triangles adjacent along edges of $t$. Note that, given an edge $\tau$ bounding $t$, we encode, at most, an index to an other triangle adjacent to $t$ along $\tau$, if $\tau$ is manifold. Otherwise, we encode the index of the record describing $\tau$ in the table related to partial co-boundary relation $R_{1,2}^*$. In other words, we describe non-manifold adjacency along $\tau$ through two triangles in $R_{1,2}^*(\tau)$, with respect to $t$. We need three indices to encode adjacency relation $R_{2,2}(t)$, for any triangle $t$ in $\Sigma$. Hence, the number of indices needed to encode adjacency relation $R_{2,2}$ is $3s_2$. Therefore, the storage cost $S_{TS}$ of the TS data structure is:

$$S_{TS} = 4s_1^t + 6s_2 + K_0^2 + 2 \sum_{t \in \Sigma^2} N_1^2$$

(5.2)

If the input simplicial 2-complex $\Sigma$ is regular, then all the top simplices are maximal, and $s_1^t = 0$. As a consequence, the storage cost $S_{TS}$ of the TS data structure reduces to:

$$S_{TS} = 6s_2 + K_0^2 + 2 \sum_{t \in \Sigma^2} N_1^2$$

It is interesting to notice that the storage cost may become extremely large, if compared with the storage cost $S_{IS}$ of the IS(2D) data structure. For instance, in a particular configuration, it may be $N_1^2 = 3$, for a triangle $t$ in $\Sigma$. As a consequence, for a regular simplicial 2-complex, the storage cost $S_{TS}$ becomes $12s_2 + K_0^2$, and it is larger than the storage cost $S_{IS}^{2D} = 2s_1 + 6s_2 + K_0^2$. As we will see in Section 5.4.1, an example of this behavior for storage costs $S_{TS}$ and $S_{IS}^{2D}$ is given by the “800-Cubes” shape in Table 5.2.

The TS data structure scales well to manifolds. In this case, all the top simplices are maximal, and thus $s_1^t = 0$. Only partial co-boundary relation $R_{0,2}^*$ is not empty, and, thus, $k^2(v) = 1$, for each vertex $v$. Therefore, the storage cost $S_{TS}$ for any manifold simplicial 2-complex is:

$$S_{TS} = s_0 + 6s_2$$

Hence, the TS data structure, restricted to manifolds, reduces to the EIA data structure, which we discussed in Section 3.1.3.

The TS data structure supports a recursive strategy to retrieve topological queries in a simplicial 2-complex $\Sigma$, and most of topological relations are optimal [DFMPS04].

For any top $p$-simplex $\sigma$, with $0 < p \leq 2$, we directly encode partial boundary relation $R_{p,0}^*(\sigma)$. A simplex in $\Sigma$, not directly encoded in the TS data structure, is explicitly described by its vertices.
As a consequence, it is possible to generate all the $k$-faces of a $h$-simplex $\sigma'$, with $0 < h \leq 2$, and $0 \leq k < h$, in terms of all the vertices of $\sigma'$, namely boundary relations $R_{h,k}$.

A triangle adjacent along a manifold edge to any triangle $t$ is encoded in adjacency relation $R_{2,2}(t)$. All the triangles adjacent along any non-manifold edge $\tau$ are traversed, either in counter-clockwise, or clockwise order around $\tau$, through partial co-boundary relation $R_{1,2}^*(\tau)$, with respect to any triangle in $St(\tau)$. We retrieve all the triangles in $R_{0,2}(v)$ by performing a breadth-first traversal of each 2-cluster in $St(v)$. In other words, we expand each 2-cluster in $St(v)$, represented by any triangle $t$ in $R_{0,2}^*(v)$, as the transitive closure of adjacency relation $R_{2,2}$, starting from $t$.

For any vertex $v$, all the top edges in $St(v)$ are directly encoded in partial co-boundary relation $R_{0,1}^*(v)$. Other edges are retrieved by selecting all the edges in $St(v)$ from the edges bounding triangles in $R_{0,2}(v)$. Adjacency relation $R_{0,0}(v)$ is retrieved by selecting all the vertices different from $v$, which belong to the boundary of top edges, and triangles in $St(v)$. We need to retrieve co-boundary relation $R_{0,2}(v)$, and, thus, partial co-boundary relation $R_{0,1}^*$, and adjacency relation $R_{0,0}$, are local.

If an edge $e = (v_1, v_2)$ is manifold, then co-boundary relation $R_{1,2}^*(e)$ consists, at most, of two triangles, but it is local. All the triangles in $St(e)$ are retrieved by selecting triangles in $St(v_1)$ and in $St(v_2)$. As consequence, the time complexity of this operation is linear in the number of triangles incident at one vertex bounding $e$. However, if we know one triangle in $St(e)$, then this relation is constant, because the other triangle adjacent along $e$ is retrieved through adjacency relation $R_{2,2}$.

In [DFMP04] the authors discuss how several editing operators can be applied on the TS data structure. They also define the Non-manifold Multi-Tessellation (NMT) model, a multi-resolution model, restricted to simplicial 2-complexes.

5.3 The Non-Manifold Indexed data structure with Adjacencies

In this section, we define a slight modification of the Non-Manifold Indexed data structure with Adjacencies (NMIA), proposed in [DFH03]. The NMIA data structure is an explicit, and adjacency-based representation, specific for arbitrary simplicial 3-complexes embedded in the Euclidean space $E^3$. It is one of the first attempts to generalize the EIA data structure to arbitrary simplicial 3D complexes.

The NMIA data structure can represent any arbitrary simplicial 3-complex embedded in the Euclidean space $E^3$, because it exploits the radial sorting of triangles and tetrahedra around a non-manifold edge, in order to reduce the amount of information encoded. Our modification is based on a compact encoding for the non-manifold adjacency along an edge, and gives an efficient characterization of non-manifold singularities.
Given an arbitrary simplicial 3-complex $\Sigma$, we decompose the star of a vertex $v$ in $\Sigma$ into maximal connected components in the same way as the IS(3D) data structure, described in Section 4.4.2. Recall that any connected component in $St(v)$ may be either a top edge, or a 1-connected sub-complex $\Sigma'$ of $St(v)$, formed by top triangles and tetrahedra, namely a vertex-based cluster. Any vertex-based cluster $\Sigma'$ of dimension $p$ in $St(v)$, with $1 < p \leq 3$, is described by a top $p$-simplex in $\Sigma'$, arbitrarily selected. This top $p$-simplex is said to be the representative simplex of $\Sigma'$. It is clear that the representative simplex of a top edge $e$ in $St(v)$ is $e$ itself. Given a vertex $v$ in $\Sigma$, for $1 < p \leq 3$, we denote the representative $p$-simplices of all the vertex-based $p$-clusters incident at $v$ as partial co-boundary relation $R^*_0, p(v)$. Top edges incident at $v$ form partial co-boundary relation $R^*_0, 1(v)$.

In this context, we decompose the star of any edge $\tau$ in $\Sigma$ into maximal connected components, like in the IS(3D) data structure, described in Section 4.4.2. Recall that any connected component in $St(\tau)$ may be either a top triangle, or a 3-cluster, namely an edge-based cluster. Any 3-cluster $\Sigma'$ in $St(\tau)$ is described by a tetrahedron, arbitrarily selected. This tetrahedron is said to be the representative simplex of $\Sigma'$. It is clear that the representative simplex of a top triangle $f$ in $St(\tau)$ is $f$ itself. Top triangles and tetrahedra in $St(\tau)$ are 1-adjacent along $\tau$. Specifically, a top face $f$ is 1-adjacent along an edge $\tau$ to an $h$-cluster $C$, for $1 < h \leq 3$, if $f$ and the representative simplex of $C$ are 1-adjacent along $\tau$. A 3-cluster $C_1$ is 1-adjacent to any $h$-cluster $C_2$, for $1 < h \leq 3$, along an edge $\tau$, if the representative simplices of $C_1$ and $C_2$ are 1-adjacent along $\tau$.

Given a top $p$-simplex $\sigma$ in a $p$-cluster $C_\sigma$, for $1 < p \leq 3$, the representative simplices of clusters 1-adjacent to $C_\sigma$ along any edge $\tau$ of $\sigma$ form relation $R_{p, cl}(\sigma)$.

Figure 5.6(a) shows four edge-based clusters of different dimensions, belonging to the star of edge $e$. Two 2-clusters are formed by top triangles $f_1$ and $f_2$, respectively. Two 3-clusters are formed by tetrahedra $t_1$ and $t_2$, and by tetrahedron $t_3$, respectively. As a consequence, $R_{2, cl}(f_1) = \{f_2, t_1, t_3\}$, $R_{3, cl}(t_1) = R_{3, cl}(t_2) = \{f_1, t_3, f_2\}$, and $R_{3, cl}(t_3) = \{f_1, f_2, t_1\}$. Figure 5.6(b) shows three 3-clusters belonging to the star of edge $e$: a 3-cluster is formed by tetrahedra $t_1$ and $t_2$, while the remaining 3-clusters are formed by tetrahedra $t_3$ and $t_4$. Here, $R_{3, cl}(t_1) = R_{3, cl}(t_2) = \{t_3, t_4\}$, $R_{3, cl}(t_3) = \{t_1, t_4\}$, and $R_{3, cl}(t_4) = \{t_1, t_3\}$.

Given any arbitrary simplicial 3-complex $\Sigma$, the NMIA data structure encodes all the vertices, top edges (top 1-simplices), top triangles (top 2-simplices), and tetrahedra (3-simplices) in $\Sigma$, plus the following relations:

- partial boundary relation $R^*_{p, 0}(\sigma)$, with $0 < p \leq 3$, for each top $p$-simplex $\sigma$, which consists of vertices bounding $\sigma$;

- partial co-boundary relation $R^*_0, p(v)$, with $0 < p \leq 3$, for each vertex $v$, which consists of one representative simplex, arbitrarily selected, for each top edge and vertex-based cluster of dimension $p$ in $St(v)$;

- adjacency relation $R_{3, 3}(t)$, for each tetrahedron $t$, which consists of four tetrahedra adjacent to $t$ along its four triangles;
we reduce this problem to the radial sorting of several triangles around an edge. This operation

- relation $R_{p,cl}(\sigma)$, with $1 < p \leq 3$, for a top $p$-simplex $\sigma$ belonging to a $p$-cluster $C_\sigma$, which consists, for $p \leq h \leq 3$, of one representative simplex for each $h$-cluster 1-adjacent to $C_\sigma$;

- partial relation $R_{1,cl}^*(\tau)$, for an edge $\tau$ bounding a top simplex $\sigma$ belonging to a cluster $C_\sigma$ in $St(\tau)$, which consists of the representative simplices of the two clusters in $St(\tau)$ preceding and following $C_\sigma$ in counter-clockwise order around $\tau$ with respect to $\sigma$, respectively.

We exploit relation $R_{2,cl}$ in order to describe the two top triangles $f_1$ and $f_2$, which are adjacent along a manifold edge $\tau$. In this context, $R_{2,cl}(f_1) = \{f_2\}$, and $R_{2,cl}(f_2) = \{f_1\}$.

Relations $R_{2,cl}$ and $R_{3,cl}$ contain clusters 1-adjacent along an edge $\tau$. These relations can be simplified by using partial relation $R_{1,cl}^*(\tau)$, if $\tau$ is a non-manifold edge. As discussed in Section 5.1, the star of any non-manifold edge $\tau$ may be formed by clusters of different dimensions, as shown in Figure 5.6(a). Conversely, an edge $\tau$ may be non-manifold, although its star is formed by clusters of the same dimension. For instance, in Figure 5.6(b), edge $e$ is shared only by 3-clusters, and it is non-manifold. Similarly, a non-manifold edge $\tau$ may be shared by more than two top triangles.

Given a top $p$-simplex $\sigma$ in a $p$-cluster $C_\sigma$, for $1 < p \leq 3$, we encode relation $R_{p,cl}(\sigma)$ along a non-manifold edge $\tau$ of $\sigma$, by replicating partial relation $R_{1,cl}^*(\tau)$ with respect to each top simplex in $C_\sigma$. In Figure 5.6(a), edge $e$ is shared by two top triangles $f_1$ and $f_2$, plus tetrahedra $t_1$, $t_2$, and $t_3$: as a consequence, $R_{1,cl}^*(e) = \{t_1, t_3\}$ for $f_1$, $R_{1,cl}^*(e) = \{t_3, t_1\}$ for $f_2$, and $R_{1,cl}^*(e) = \{f_2, f_1\}$ for $t_1$ and $t_2$. Finally, $R_{1,cl}^*(e) = \{f_1, f_2\}$ for $t_3$. Similarly, in Figure 5.6(b), edge $e$ is shared by tetrahedra $t_1$, $t_2$, $t_3$, and $t_4$: as a consequence, $R_{1,cl}^*(e) = \{t_3, t_4\}$ for $t_1$ and $t_2$, $R_{1,cl}^*(e) = \{t_4, t_1\}$ for $t_3$, and $R_{1,cl}^*(e) = \{t_1, t_3\}$ for $t_4$.

We associate a Euclidean point in $\mathbb{E}^3$ with each vertex in $\Sigma$ in order to radially sort triangles and tetrahedra around a non-manifold edge $\tau = (v_1, v_2)$. For each tetrahedron $\sigma$ representing a cluster $C_\sigma$ incident at $\tau$, we can consider one of its two triangles, which is incident at $\tau$. In this way, we reduce this problem to the radial sorting of several triangles around an edge. This operation

Figure 5.6: Edge-based clusters incident at a non-manifold edge $e$. Here, the star of a non-manifold edge $e$ may be composed either of (a) edge-based clusters of different dimensions, or (b) of the same dimension.
is equivalent to radially sort all the vertices different than \( v_1 \) and \( v_2 \) of triangles in \( St(\tau) \) around edge \( \tau \), as proposed in [PS85]. At the end of this operation, we replace any triangle bounding a tetrahedron \( \sigma \) with the tetrahedron \( \sigma \) itself.

In order to represent a simplicial 3-complex \( \Sigma \) through the NMIA data structure, we exploit the same approach described in [AJ05], used for encoding the \( IA^* \) data structure. For the sake of clarity, we encode auxiliary records for describing non-manifold edges in \( \Sigma \). We assign a unique index to vertices, top edges and triangles, tetrahedra, and non-manifold edges in \( \Sigma \). The unique index of a simplex \( \sigma \), namely an integer value, is used for accessing records related to \( \sigma \) in each table.

We represent the Euclidean points in \( \mathbb{E}^3 \) associated with each vertex in \( \Sigma \) through an array \( V \) with \( s_0 \) locations, one for each vertex in the collection \( \Sigma^0 \) of vertices. Its storage cost is \( 3s_0 \) floating point values. In the remainder of this section, we consider only the storage cost related to topological relations.

We encode, for \( 0 < p \leq 3 \), partial boundary relations \( R^*_{p,0} \) as three tables, one for each relation. A record of these tables represents, for \( 0 < p \leq 3 \), a top \( p \)-simplex, defined by \( p + 1 \) indices of its vertices. Thus, the number of indices needed to encode partial boundary relations of all the top simplices in \( \Sigma \) is \( 2s_1^p + 3s_2^p + 4s_3 \), where \( s_1^p \) and \( s_2^p \) are the number of top edges and triangles in \( \Sigma \), respectively, while \( s_3 \) is the number of tetrahedra. We do not encode any boundary relation for a non-manifold edge in \( \Sigma \).

We encode, for \( 1 \leq p \leq 3 \), partial co-boundary relations \( R^*_{0,p} \) as three tables, one for each relation. A record of these tables corresponds to a vertex \( v \) in \( \Sigma \), and contains indices of representative \( p \)-simplices of top edges and vertex-based clusters in \( R^*_{0,p}(v) \). Thus, the number of indices needed to encode partial co-boundary relations for vertices is \( 2s_1^1 + C^1 \), where \( C^1 \) is the number of vertex-based clusters in \( St(v) \), like in the IS(3D) data structure, discussed in Section 4.4.2.

We represent adjacency relation \( R_{3,3} \) as a table, where each record corresponds to a tetrahedron \( t \) in \( \Sigma \). In each record, we store indices of tetrahedra adjacent along faces of tetrahedron \( t \). Thus, the number of indices needed to encode adjacency relations for tetrahedra in \( \Sigma \) is \( 4s_3 \).

We encode partial relation \( R^*_{1,cl} \) as a table, where each record corresponds to a non-manifold edge \( \tau \), which bounds a top \( p \)-simplex \( \sigma \) in a \( p \)-cluster \( C_\sigma \). We store indices of the representative simplices for the two clusters in \( R^*_{1,cl}(\tau) \) with respect to \( \sigma \). Let \( N_2^f \) and \( N_3^t \) be the numbers of non-manifold edges, respectively in a top triangle \( f \), and in a tetrahedron \( t \) in \( \Sigma \). Thus, the number of indices needed to encode the partial relation \( R^*_{1,cl} \) is equal to:

\[
2 \sum_{f \in \Sigma^2_2} N_2^f + 2 \sum_{t \in \Sigma^3} N_3^t
\]

where \( \Sigma^2_2 \) and \( \Sigma^3 \) are the collections of top triangles and tetrahedra in \( \Sigma \), respectively.

We represent relation \( R_{2,cl} \) as a table, where each record corresponds to a top triangle \( f \), which is 1-adjacent either to a top triangle, or to a 3-cluster, along an edge \( \tau \) in \( f \). If \( \tau \) is manifold, then
we encode the index of one top triangle adjacent to \( f \) along \( \tau \). Otherwise, we encode an index of the record describing \( R_{1,cl}^*(\tau) \) with respect to top triangle \( f \). Thus, the number of indices needed to encode relation \( R_{2,cl} \) is \( 3s_2^2 \).

Finally, we represent relation \( R_{3,cl} \) as a table, where each record corresponds to a tetrahedron \( t \) in a 3-cluster \( C_t \), which is 1-adjacent either to a top triangle, or to a 3-cluster, along an edge \( \tau \) of \( t \). If \( \tau \) is manifold, then we do not encode any information, because an other tetrahedron is adjacent along one of the two faces of \( t \) in \( St(\tau) \). Otherwise, we encode an index of the record describing \( R_{1,cl}^*(\tau) \) with respect to \( t \). Thus, the number of indices needed to encode relation \( R_{3,cl} \) for a tetrahedron \( t \) is \( N_3^t \).

As a consequence, the storage cost \( S_{NM} \) of the NMIA data structure is:

\[
S_{NM} = 4s_1^4 + 6s_2^4 + 8s_3 + C_1 + 2 \sum_{f \in \Sigma_2} N_f^2 + 3 \sum_{t \in \Sigma^3} N_t^3
\]  

(5.3)

If the input simplicial 3-complex \( \Sigma \) is regular, then all the top simplices are maximal, \( s_1^4 = s_2^4 = 0 \), and \( C_1 = K_3^0 \), where \( K_3^0 \) is the total number of 3-clusters incident at vertices of \( \Sigma \). As a consequence, the storage cost \( S_{NM} \) of the NMIA data structure becomes:

\[
S_{NM} = 8s_3 + K_3^0 + 3 \sum_{t \in \Sigma^3} N_t^3
\]

The NMIA data structure scales well to manifolds. In this case, all the top simplices are maximal, and there are no non-manifold singularities. Only partial co-boundary relation \( R_{0,3}^* \) is not empty, and, thus, there is only one vertex-based incident in the star of each vertex \( v \), namely \( h_v = 1 \).

Therefore, the storage cost \( S_{NM} \) for a manifold simplicial 3-complex reduces to:

\[
S_{NM} = s_0 + 8s_3
\]

As a consequence, the NMIA data structure, restricted to manifolds, reduces to the EIA data structure, which we discussed in Section 3.1.3.

The NMIA data structure supports a recursive strategy to retrieve topological queries in a simplicial 3-complex \( \Sigma \), and most of topological relations are optimal, as discussed in [DFH03].

For a top \( p \)-simplex \( \sigma \), with \( 0 < p \leq 3 \), we encode partial boundary relation \( R_{p,0}^*(\sigma) \). Any other simplex in \( \Sigma \), not directly encoded in the NMIA data structure, is implicitly described by its vertices. As a consequence, it is possible to generate \( k \)-faces of a \( h \)-simplex \( \sigma' \), with \( 0 < h \leq 3 \), and \( 0 \leq k < h \), in terms of vertices of \( \sigma' \), namely boundary relations \( R_{h,k} \).

Adjacency relation \( R_{3,3}(t) \) of a tetrahedron \( t \) is directly encoded. It is possible to retrieve tetrahedra of a 3-cluster, represented by a tetrahedron \( t \), as the transitive closure of adjacency relation \( R_{3,3} \), starting from \( t \). Edge-based clusters incident at a non-manifold edge \( \tau \) are traversed either in counter-clockwise, or in clockwise order around \( \tau \), through partial co-boundary relation \( R_{1,cl}^*(\tau) \) with respect to each top simplex in \( St(\tau) \).

Top simplices incident at a vertex \( v \) are retrieved through a breadth-first traversal of top edges and vertex-based clusters in \( St(v) \). Top edges in \( St(v) \) are directly encoded in partial co-boundary
relation $R_{0,1}(v)$. Top triangles and tetrahedra in $St(v)$ are retrieved by expanding each vertex-based cluster in $R_{0,2}(v)$ and in $R_{0,3}(v)$ through relation $R^{*}_{1,cl}$, and adjacency relation $R_{3,3}$. These operations are optimal.

For $0 < k \leq 3$, co-boundary relations $R_{0,k}(v)$ for a vertex $v$ in $\Sigma$ are retrieved by selecting $k$-faces in $St(v)$, which bound a top simplex in $St(v)$. Their time complexity is linear in the number of top simplices in $St(v)$, and, thus, they are local. Co-boundary relation $R_{2,3}(f)$ of a triangle $f$ is retrieved in constant time through adjacency relation $R_{3,3}(t)$, if we know a tetrahedron $t$ in $St(f)$. Otherwise, we must select tetrahedra incident at $f$ from those incident at $f$. In this case, co-boundary relation $R_{2,3}$ is local.

For an edge $\tau$, we retrieve co-boundary relation $R_{1,2}(\tau)$ by selecting top triangles, and triangles bounding tetrahedra in $St(\tau)$. The time complexity of this operation is linear in $|R_{1,2}(\tau)|$, and, thus, it is optimal. For a triangle $f$, adjacency relation $R_{2,2}(f)$ is retrieved by selecting, for each edge $e$ bounding $f$, co-boundary relation $R_{1,2}(e)$, and, thus, it is optimal.

Remaining topological relations are local, because they are based on the retrieval of top simplices incident at one of their vertices. For instance, adjacency relation $R_{0,0}(v)$ is retrieved by removing $v$ from vertices bounding edges in $St(v)$. Co-boundary relation $R_{1,2}(e)$, for a manifold edge $e = (v_1, v_2)$, is retrieved by selecting triangles in $St(v_1)$ and in $St(v_2)$.

In [DFH04] the authors discuss how editing operators can be applied on the NMIA data structure.

5.4 Experimental Comparisons

In this section, we present a quantitative comparison of adjacency-based representations, discussed in this chapter, namely the IA*, NMIA, and TS data structures. Recall that the TS and NMIA data structures are specific, respectively, for simplicial 2-complexes and 3-complexes embedded in the Euclidean space $E^3$. Conversely, the IA* data structure is a dimension-independent representation for abstract simplicial complexes, not necessarily embedded in any Euclidean space. Our tests are performed in terms of their storage cost and encoded relations. In Section 5.4.1, we compare the TS data structure, a specialization of the IA* data structure for arbitrary simplicial 2-complexes, plus other data structures specific for simplicial 2-complexes. In Section 5.4.2, we compare the NMIA data structure, a specialization of the IA* data structure for arbitrary simplicial 3-complexes, plus other representations specific for manifold volumetric shapes. We have performed our tests on all the digital shapes freely available from [GGG09]. For the sake of brevity, we present results only on a subset of these shapes.

Our tests show that the IA* data structure is more compact than any incidence-based representation, and is even more compact than the TS and NMIA data structures. It is an interesting property, since the latter ones are able to exploit dimension-specific properties of their embedding space, like the radial ordering of triangles and tetrahedra around an edge, to reduce their storage requirements. To the best of our experience, the IA* data structure is one of the most compact
explicit, dimension-independent, and adjacency-based representations for non-manifold shapes.

### 5.4.1 Experimental Comparisons for Simplicial 2-Complexes

In this section, we present quantitative comparisons between the TS data structure, discussed in Section 5.2, and a specialization of the IA\(^*\) data structure, restricted to simplicial 2-complexes, plus other representations, specific for simplicial 2-complexes.

First, we introduce a specialization of the IA\(^*\) data structure for describing any arbitrary simplicial 2-complex \(\Sigma\), namely the IA\(^*\)(2D) representation. The IA\(^*\)(2D) representation encodes vertices and top simplices in \(\Sigma\), plus information about non-manifold edges in \(\Sigma\), which are shared by more than two triangles. It also encodes partial boundary relation \(R^*_1(e)\) and \(R^*_2(f)\) for any top edge \(e\) and any triangle \(f\) in \(\Sigma\), respectively. It encodes partial co-boundary relations \(R^*_2(v)\) and \(R^*_0(f)\) for any vertex \(v\) in \(\Sigma\). Furthermore, it encodes partial co-boundary relation \(R^*_1(\tau)\), for any non-manifold edge \(\tau\). Note that this relation is stored only once, for each non-manifold edge \(\tau\). Finally, it encodes adjacency relation \(R^*_2(\tau)\) along each edge \(e\) of any triangle \(f\) in \(\Sigma\), as discussed in Section 5.1. In this context, a top edge is encoded for each of its vertices, and, thus, \(K^1_A = 2s^1\). From Equation 5.1, we deduce that the storage cost \(S^2_{IA^*}\) of the IA\(^*\)(2D) data structure is equal to:

\[
S^2_{IA^*} = 4s^1_t + 6s^2_t + K^2_0 + \sum_{\tau \in \Sigma^2} N^2_{\tau}
\]

where \(\Sigma^1\) is the collection of 1-simplices in \(\Sigma\). If the input simplicial 2-complex \(\Sigma\) is regular, then there are not any top triangles, thus the storage cost \(S^2_{IA^*}\) becomes:

\[
S^2_{IA^*} = 6s^2_t + K^2_0 + \sum_{\tau \in \Sigma^1} N^2_{\tau}
\]

If the input simplicial 2-complex \(\Sigma\) is manifold, then the IA\(^*\)(2D) data structure reduces to the EIA data structure, and its storage cost becomes \(S^2_{IA^*} = s_0 + 6s^2\), as demonstrated in Section 5.1.2.

Let \(S_{TS}\) be the storage cost of the TS data structure, provided by Equation 5.2, then we can define the difference \(\Delta^T_{IA^*}(2D) = S_{TS} - S^2_{IA^*}\):

\[
\Delta^T_{IA^*}(2D) = 2 \sum_{t \in \Sigma^1} N^1_t - \sum_{\tau \in \Sigma^2} N^2_{\tau}
\]

where \(\Sigma^2\) is the collection of 2-simplices in \(\Sigma\). It is clear that \(\Delta^T_{IA^*}(2D)\) depends only on the different encoding of the non-manifold adjacency along any non-manifold edge \(\tau\). Note that, if there are no non-manifold edges in \(\Sigma\), then the IA\(^*\)(2D) and the TS data structures are the same.

In the TS data structure, we store triangles in \(St(\tau)\) in a circular list, sorted either in counterclockwise, or in clockwise order around \(\tau\), as discussed in Section 5.2. Basically, we store two indices for each triangle in the star of a non-manifold edge \(\tau\). Conversely, the IA\(^*\)(2D) data structure stores triangles incident at a non-manifold edge \(\tau\) in \(\Sigma\) only once, and, then, we refer to these triangles by a reference, as discussed in Section 5.1.1. Thus, if there are non-manifold edges
in $\Sigma$, then the IA$^*$ (2D) data structure is more compact than the TS representation. Our tests provides an experimental assessment of this property.

In Section 4.4.1, we have compared storage costs of several incidence-based data structures with a subset of manifold shapes, which we have also reused for evaluating storage cost of the IA$^*$ data structure. In Table 5.1, we evaluate storage costs of the IA$^*$ (2D) data structure, and of other representations specific for manifolds, like the Corner Table representation [RSS01], the Lath data structure [JLMC02], and the SQuad representation [GLLR11a], briefly discussed in Section 3.2.1.1. Recall that the storage cost of the Corner Table data structure coincides with the storage cost of the Star-Vertex representation [KT01]. Note that, in this case, the SIG(2D) data structure coincides with the IS(2D) representation. Here, we approximate storage costs of all the data structures, by using the Euler-Poincaré formula [Ago05]. Thus, $s_1 \approx 3s_0$ and $s_2 \approx 2s_0$ [Ede87].

Recall that, in this case, the TS, the EIA, and the IA$^*$ (2D) data structures coincide, and their storage cost is $13s_0$ indices. The storage cost of the IS(2D) data structure is $19s_0$, while the Lath representation requires $18s_0$ indices. The Corner Table data structure requires $12s_0$ indices. The SQuad data structure requires 2 references per triangle, and, thus, $4s_0$ indices.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$s_0$</th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$S_{IS}^D$</th>
<th>$S_L$</th>
<th>$S_{IA}^D$</th>
<th>$S_C$</th>
<th>$S_Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cone</td>
<td>0.6k</td>
<td>1.8k</td>
<td>1.2k</td>
<td>11.4k</td>
<td>10.8k</td>
<td>7.8k</td>
<td>7.2k</td>
<td>2.4k</td>
</tr>
<tr>
<td>Crumb</td>
<td>0.3k</td>
<td>0.9k</td>
<td>0.6k</td>
<td>5.7k</td>
<td>5.4k</td>
<td>3.9k</td>
<td>3.6k</td>
<td>1.2k</td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>0.08k</td>
<td>0.24k</td>
<td>0.16k</td>
<td>1.52k</td>
<td>1.44k</td>
<td>1.04k</td>
<td>0.96k</td>
<td>0.32k</td>
</tr>
<tr>
<td>Football 1</td>
<td>1.2k</td>
<td>3.6k</td>
<td>2.4k</td>
<td>22.8k</td>
<td>21.6k</td>
<td>15.6k</td>
<td>14.4k</td>
<td>4.8k</td>
</tr>
<tr>
<td>Football 2</td>
<td>0.9k</td>
<td>2.7k</td>
<td>1.8k</td>
<td>17.1k</td>
<td>16.2k</td>
<td>11.7k</td>
<td>10.8k</td>
<td>3.6k</td>
</tr>
<tr>
<td>Torus</td>
<td>10.2k</td>
<td>30.6k</td>
<td>20.4k</td>
<td>193.8k</td>
<td>183.6k</td>
<td>132.6k</td>
<td>122.4k</td>
<td>40.8k</td>
</tr>
</tbody>
</table>

Table 5.1: Storage costs of several data structures, which describe manifold simplicial 2-complexes. In particular, we analyze storage costs of the Lath ($S_L$), Corner Table ($S_C$), and SQuad ($S_Q$) data structures, plus storage costs of the IS(2D) ($S_{IS}^D$) and IA$^*$ (2D) ($S_{IA}^D$) representations. Recall that $s_j$ is the number of $j$-simplices in a triangulation, with $0 \leq j \leq 2$.

These tests show that the IA$^*$ (2D) data structure is a valuable tool for representing manifold shapes. Here, the IS(2D) data structure is about 1.46 times more expensive than the IA$^*$ (2D) representation. As a consequence, the IA$^*$ (2D) data structure is also more compact than all the data structures, specific for manifolds, analyzed in Section 4.4.1. For instance, the HE and WE data structures are, respectively, 2.5 and 2 times more expensive than the IA$^*$ (2D) representation. The X-Maps data structure is equivalent to the IG(2D) representation, and, thus, it is about 2 times more expensive than the IA$^*$ (2D) representation. In any case, the Corner Table, Star-Vertex, and SQuad representations are more compact than the IA$^*$ (2D) data structure. The Corner Table representation is slightly more compact than the IA$^*$ (2D) data structure, and the difference between their storage costs is only $s_0$. The IA$^*$ (2D) data structure is only 1.08 times more expensive than the Corner Table and Star-Vertex representations. Conversely, the SQuad data structure is about 3 times more compact than the IA$^*$ (2D) representation. In any case, these data structures are implicit, and represent only manifold simplicial 2-complexes. The Star-Vertex data structure has not the full navigation capability, as the other data structures. Compressed representations,
like the SQuad data structure, do not support editing operators, as discussed in Section 3.2.1.1. Conversely, the IA\(^+\) data structure is a dimension-independent, explicit, and adjacency-based data structure, which offers a good compromise regarding its storage cost, navigation capabilities, and recognition of non-manifold singularities. The IA\(^+\) data structure represents non-manifold shapes of arbitrary dimension.

Now, we compare storage costs of several data structures for arbitrary simplicial 2-complexes, namely the TS, IA\(^{(2D)}\) and IS\((2D)\) representations, plus the Vertex-Face (VF) [VL97] data structure, discussed in Section 3.2.4. Note that the VF data structure represents only regular simplicial 2-complexes. Table 5.2 summarizes our results.

<table>
<thead>
<tr>
<th>Shape</th>
<th>(s_0)</th>
<th>(s_1)</th>
<th>(s_2)</th>
<th>(s_3)</th>
<th>(S_{1S}^{2D})</th>
<th>(S_{VF})</th>
<th>(S_{TS})</th>
<th>(S_{IA}^{2D})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armchair</td>
<td>5.3k</td>
<td>15.9k</td>
<td>10.6k</td>
<td>–</td>
<td>100.7k</td>
<td>95.4k</td>
<td>69.3k</td>
<td>69.1k</td>
</tr>
<tr>
<td>800-Cubes</td>
<td>2.2k</td>
<td>10.7k</td>
<td>9.6k</td>
<td>–</td>
<td>81.1k</td>
<td>78.6k</td>
<td>89.9k</td>
<td>75k</td>
</tr>
<tr>
<td>Cylinders</td>
<td>90</td>
<td>300</td>
<td>200</td>
<td>–</td>
<td>1.9k</td>
<td>1.8k</td>
<td>1.33k</td>
<td>1.31k</td>
</tr>
<tr>
<td>Pinched-pie</td>
<td>696</td>
<td>2.9k</td>
<td>2.3k</td>
<td>–</td>
<td>20.5k</td>
<td>19.6k</td>
<td>18.5k</td>
<td>16.6k</td>
</tr>
<tr>
<td>Twist</td>
<td>1.2k</td>
<td>3.5k</td>
<td>2.4k</td>
<td>–</td>
<td>22.5k</td>
<td>21.3k</td>
<td>15.7k</td>
<td>15.6k</td>
</tr>
<tr>
<td>Robot</td>
<td>3.2k</td>
<td>10k</td>
<td>6.7k</td>
<td>–</td>
<td>63.5k</td>
<td>60.2k</td>
<td>46.4k</td>
<td>44.9k</td>
</tr>
<tr>
<td>Balance</td>
<td>4k</td>
<td>12k</td>
<td>8k</td>
<td>34</td>
<td>75.9k</td>
<td>–</td>
<td>51.9k</td>
<td>51.9k</td>
</tr>
<tr>
<td>Carter</td>
<td>4k</td>
<td>11.9k</td>
<td>7.9k</td>
<td>2</td>
<td>75.3k</td>
<td>–</td>
<td>53.4k</td>
<td>52.4k</td>
</tr>
<tr>
<td>Chandelier</td>
<td>4k</td>
<td>12k</td>
<td>8k</td>
<td>136</td>
<td>174.5k</td>
<td>–</td>
<td>121.1k</td>
<td>120.3k</td>
</tr>
<tr>
<td>Frame</td>
<td>987</td>
<td>2.2k</td>
<td>1.1k</td>
<td>216</td>
<td>12.1k</td>
<td>–</td>
<td>8.1k</td>
<td>8.1k</td>
</tr>
<tr>
<td>Tower</td>
<td>9.1k</td>
<td>27.7k</td>
<td>18.4k</td>
<td>160</td>
<td>175.2k</td>
<td>–</td>
<td>123.8k</td>
<td>121.9k</td>
</tr>
<tr>
<td>Tower-wir</td>
<td>8.3k</td>
<td>24.6k</td>
<td>15.9k</td>
<td>896</td>
<td>154.3k</td>
<td>–</td>
<td>111.7k</td>
<td>109.3k</td>
</tr>
</tbody>
</table>

Table 5.2: Storage costs of several data structures, describing arbitrary simplicial 2-complexes. We analyze storage costs of the Vertex-Face (\(S_{VF}\)), IS\((2D)\) (\(S_{1S}^{2D}\)), TS (\(S_{TS}\)), and IA\(^{(2D)}\) (\(S_{IA}^{2D}\)) representations. Note that the Vertex-Face data structure represents only regular simplicial 2-complexes. Recall that, given any simplicial 2-complex \(\Sigma\), \(s_j\) is the number of \(j\)-simplices in \(\Sigma\), with \(0 \leq j \leq 2\), while \(s_3\) is the number of top edges in \(\Sigma\).

These tests show that the IA\(^{(2D)}\) representation is one of the most compact data structure for arbitrary shapes. The IS\((2D)\) representation, and the VF data structure are, respectively, about 1.45 and 1.28 times more expensive than the IA\(^{(2D)}\) data structure. Note that, in this context, the VF data structure is also only 5\% more compact than the IS\((2D)\) representation. The TS data structure is, on average, only 4\% more expensive than the IA\(^{(2D)}\) data structure. In most of tests, the IS\((2D)\) representation is, on average, 1.36 times more expensive than the TS data structure. In any case, as discussed in Section 5.2, the storage cost of the TS data structure may become quite large, if compared with the storage cost of the IS\((2D)\) data structure. For instance, in Table 5.2, the storage cost \(S_{TS}\) of the TS data structure, representing the “800-Cubes” shape, is larger than \(S_{1S}^{2D}\). In this case, \(S_{TS} \approx 1.1 \times S_{1S}^{2D}\), and \(S_{TS} \approx 1.2 \times S_{IA}^{2D}\).

Furthermore, several data structures discussed in Section 4.4.1 are extremely more expensive than the IA\(^{(2D)}\) representation. Specifically, we have compared the RE data structure [Wei88b], discussed in Section 3.2.2, the PE data structure [LL01], discussed in Section 3.2.3, the DE data structure [CKS98], discussed in Section 3.2.5, and the IG\((2D)\) data structure, introduced in Sec-
tion 4.4.1. The RE data structure is about 11.2 times more expensive than the IA\(^\ast\)(2D) representation. The PE and DE representations are, respectively, 3.9 and 2.46 times more expensive than the IA\(^\ast\)(2D) data structure. Finally, the IG(2D) representation is only 1.8 times more expensive than the IA\(^\ast\)(2D) data structure.

Now, we compare the TS and IA\(^\ast\)(2D) representations, by evaluating their storage costs in terms of which topological relations they encode. Table 5.3 summarizes contributions to storage costs \(S_{TS}\) and \(S_{IA}^{2D}\), for all the simplicial 2-complexes, analyzed in Table 5.1 and Table 5.2. Here, we denote the number of non-manifold vertices and edges as, respectively, \(s_n^0\) and \(s_n^1\). Then, we denote the total storage cost of boundary relations as \(2s_1^t + 3s_2\). The total number of 2-clusters incident at vertices is denoted as \(K^2\). We denote the total number of indices needed for encoding the non-manifold adjacency along any edge in the IA\(^\ast\)(2D) data structure as \(N^1\). In the TS data structure, we denote the total number of indices needed for encoding non-manifold adjacencies along an edge as \(N^2\). Finally, we denote the minimum and maximum number of triangles incident at a non-manifold edge \(\tau\) as, respectively, \(min(N^2)\) and \(max(N^2)\).

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\text{Shape} & s_n^0 & s_n^1 & 2s_1^t + 3s_2 & K^2 & \min(N^1) & \max(N^1) & \Delta^2_{IA} (2D) \\
\hline
\text{Cone} & - & - & 3.6k & 0.6k & - & - & - \\
\text{Crumb} & - & - & 1.8k & 0.3k & - & - & - \\
\text{Dodecahedron} & - & - & 0.48k & 0.08k & - & - & - \\
\text{Football 1} & - & - & 7.2k & 1.2k & - & - & - \\
\text{Football 2} & - & - & 5.4k & 0.9k & - & - & - \\
\text{Torus} & - & - & 64.2k & 10.2k & - & - & - \\
\hline
\text{Armchair} & 57 & 56 & 31.6k & 5.3k & 0.2k & 0.4k & 3 & 4 & 0.2k \\
\text{800-Cubes} & 2k & 3.7k & 28.8k & 2.5k & 14.9k & 29.8k & 4 & 4 & 14.9k \\
\text{Cylinders} & 3 & 2 & 0.6k & 0.1k & 0.01k & 0.03k & 8 & 8 & 0.02k \\
\text{Pinched-pie} & 456 & 480 & 6.9k & 0.9k & 1.9k & 3.84k & 4 & 4 & 1.94k \\
\text{Twist} & 28 & 28 & 7.2k & 1.1k & 0.09k & 0.19k & 3 & 4 & 0.1k \\
\text{Robot} & 508 & 408 & 20.1k & 3.3k & 1.4k & 2.9k & 1 & 3 & 1.5k \\
\hline
\text{Balance} & 8 & - & 24k & 3.8k & - & - & - & - \\
\text{Carter} & 328 & 313 & 23.7k & 4.1k & 0.9k & 1.9k & 3 & 4 & 1k \\
\text{Chandelier} & 352 & 264 & 55.5k & 9.2k & 0.8k & 1.6k & 3 & 3 & 0.8k \\
\text{Frame} & 165 & - & 3.7k & 668 & - & - & - & - \\
\text{Tower} & 801 & 640 & 55.5k & 9k & 1.9k & 3.8k & 3 & 3 & 1.9k \\
\text{Tower-wir} & 1.3k & 795 & 49.5k & 7.9k & 2.4k & 4.8k & 3 & 3 & 2.4k \\
\hline
\end{array}
\]

Table 5.3: Contributions to \(S_{TS}\) and \(S_{IA}^{2D}\), for all the simplicial 2-complexes analyzed in Table 5.1 and Table 5.2. Here, we denote the number of non-manifold vertices and edges as, respectively, \(s_n^0\) and \(s_n^1\). Then, we denote the total storage cost of boundary relations as \(2s_1^t + 3s_2\). The total number of 2-clusters incident at vertices is denoted as \(K^2\). We denote the total number of indices needed for encoding the non-manifold adjacency along any edge in the IA\(^\ast\)(2D) data structure as \(N^1\). In the TS data structure, we denote the total number of indices needed for encoding non-manifold adjacencies along an edge as \(N^2\). Finally, we denote the minimum and maximum number of triangles incident at a non-manifold edge \(\tau\) as, respectively, \(\min(N^2)\) and \(\max(N^2)\).

These tests experimentally justify several theoretical properties of the TS and IA\(^\ast\)(2D) data structures. In the TS data structure, the total storage cost of boundary relations, namely \(2s_1^t + 3s_2\),
is about 42% of $S_{TS}$. In the IA$^*$ (2D) data structure, this storage cost is about 46% of $S_{IA}^{2D}$. Remaining relations require, on average, about 58% and 54% of, respectively, $S_{TS}$ and $S_{IA}^{2D}$. Note that boundary relations are encoded in both the TS and IA$^*$ (2D) data structures. In any case, for the “800-Cubes” shape, it is clear that $N_{IA}^2 > 2s_1^4 + 3s_2$, namely $N_{IA}^2 \approx 3 \times (2s_1^4 + 3s_2^2)$. Recall that, as just discussed above, the TS data structure may become more expensive than the IS(2D) data structure, with respect to the complexity of non-manifold configurations.

Note that $\Delta_{IA}^2, (2D)$ depends only on the different encoding of the non-manifold adjacency along non-manifold edges in the input simplicial 2-complex $\Sigma$. For instance, if there are no non-manifold edges in $\Sigma$, then the TS and IA$^*$ (2D) data structures are the same, for instance with the “Balance” and “Frame” shapes in Table 5.3. It is also clear that $N_{IA}^2 > N_{IA}^1$. The IA$^*$ (2D) data structure encodes only one record for each non-manifold edge $\tau$ in $\Sigma$, and stores all the triangles in $St(\tau)$ only once. Conversely, the TS data structure encodes one record related to $\tau$ for each triangle in $St(\tau)$, as discussed in Section 5.2. Hence, the number of records, encoded in the TS data structure, may be extremely large. For instance, with the “800-Cubes” shape in Table 5.3, the IA$^*$ (2D) data structure encodes 3.7$k$ records, one for each non-manifold edge $\tau$. Here, the star of each non-manifold edge $\tau$ contains four triangles. Conversely, the TS data structure encodes partial co-boundary relation $R_{IA}^1, (3D)$ for each of the four triangles in $St(\tau)$. In this case, $N_{IA}^1 \approx 4 \times 3.7k \approx 14.9k$, and $N_{IA}^2 = 2 \times N_{IA}^1 \approx 29.8k$.

### 5.4.2 Experimental Comparisons for Simplicial 3-Complexes

In this section, we present quantitative comparisons between the NMIA data structure, discussed in Section 5.3, and a specialization of the IA$^*$ data structure, restricted to simplicial 3-complexes, plus other representations, specific for simplicial 3-complexes.

First, we introduce a specialization of the IA$^*$ data structure for describing any arbitrary simplicial 3-complex $\Sigma$, namely the IA$^*$ (3D) representation. The IA$^*$ (3D) representation encodes vertices and top simplices in $\Sigma$, plus information about non-manifold edges in $\Sigma$, shared by at least one top triangle. All the non-top 2-simplices are shared by at most two tetrahedra, thus they are manifold. The IA$^*$ (3D) representation also encodes, for $0 < p \leq 3$, partial boundary relation $R_{p,0}^v(\sigma)$, for any top $p$-simplex $\sigma$. It also encodes, for $0 < p \leq 3$, partial co-boundary relations $R_{IA}^1, p(v)$, for any vertex $v$, and partial co-boundary relation $R_{IA}^2, (\tau)$, for any non-manifold edge $\tau$, shared by at least one top triangle. Note that this relation is stored only once, for each non-manifold edge $\tau$. Finally, it encodes partial adjacency relation $R_{IA}^2, (f)$ along each edge of any top triangle $f$, as discussed in Section 5.1, plus adjacency relation $R_{3,3}^t(t)$, for each tetrahedron $t$ in $\Sigma$. A top edge is encoded for each of its vertices, and, thus, $K_0^1 = 2s_1^4$. From Equation 5.1, we deduce that the storage cost $S_{IA}^{3D}$ of the IA$^*$ (3D) data structure is equal to:

$$S_{IA}^{3D} = 4s_1^4 + 6s_2^4 + 8s_3 + K_0^2 + K_0^3 + \sum_{\tau \in \Sigma^1} N_\tau^2$$

If the input simplicial 3-complex $\Sigma$ is regular, then all the top simplices are maximal, and, thus, $s_1^4 = s_2^4 = 0$ and $K_0^3 = 0$. Non-manifold edges are shared by several tetrahedra, and, thus, $N_\tau^2 = 0$,
for any non-manifold edge $\tau$. As a consequence, $S_{IA^*}^{3D} = 8s_3 + K_0^3$.

If the input simplicial 3-complex $\Sigma$ is manifold, then the $IA^*(3D)$ data structure reduces to the EIA data structure, and its storage cost becomes $S_{IA^*}^{3D} = s_0 + 8s_3$, as demonstrated in Section 5.1.2.

Let $S_{NM}$ be the storage cost of the NMIA data structure, provided by Equation 5.3, then we can define the difference $\Delta_{NM}^{IA^*}(3D) = S_{NM} - S_{IA^*}^{3D}$ as:

$$\Delta_{NM}^{IA^*}(3D) = (C_1 - K_0^2 - K_0^3) + \left(2 \sum_{f \in \Sigma^2} N_f^2 + 3 \sum_{t \in \Sigma^3} N_t^3 - \sum_{\tau \in \Sigma^2} N_\tau^2 \right)$$

where $\Sigma^1$ and $\Sigma^3$ are, respectively, the collections of 1- and 3-simplices in $\Sigma$, while $\Sigma^2$ is the collection of top 2-simplices in $\Sigma$. It is clear that $\Delta_{NM}^{IA^*}(3D)$ depends on the numbers of connected components (namely $C_1$), and of clusters (namely $K_0^2 + K_0^3$) in the star of all the vertices in $\Sigma$, and on the different encoding of non-manifold adjacencies along any non-manifold edge in the NMIA and in the $IA^*(3D)$ data structures. Note that both the $IA^*(3D)$ and NMIA data structures reduce to the EIA representation, if restricted to manifolds. If the input simplicial 3-complex is regular, then the difference $\Delta_{NM}^{IA^*}(3D)$ becomes:

$$\Delta_{NM}^{IA^*}(3D) = 3 \sum_{t \in \Sigma^3} N_t^3$$

The $IA^*(3D)$ data structure tends to be more compact than the NMIA representation. As discussed in Section 4.4.2, the number of clusters in the star of a vertex is larger than the number of vertex-based clusters. In other words, $K_0^2 + K_0^3 > C_1$. The $IA^*(3D)$ data structure stores top triangles incident at a non-manifold edge $\tau$ in $\Sigma$ only once, and, then, we refer to these triangles through a reference, as discussed in Section 5.1.1. In the NMIA data structure, we store top triangles and 3-clusters in $St(\tau)$ in a circular list, sorted either in counter-clockwise, or in clockwise order around $\tau$, as discussed in Section 5.3. Informally, the NMIA data structure encodes more records related to non-manifold edges than the $IA^*(3D)$ representation. For instance, in this latter, we do not store any record related to non-manifold edges, shared only by tetrahedra, like edge $e$ in Figure 5.2(b).

In Section 4.4.2, we have compared storage costs of several incidence-based data structures with a subset of manifold shapes, which we also reuse in these tests. In Table 5.4, we evaluate storage costs of the IS(3D) and $IA^*(3D)$ data structures. Note that the EIA, NMIA, and $IA^*(3D)$ data structures are the same, if restricted to manifolds.

Our tests show that the $IA^*(3D)$ data structure scales well for manifolds, and it is a valuable tool for their representation. As demonstrated by our tests, the $IA^*(3D)$ representation is always more compact than all the data structures analyzed in Section 4.4.2. The IS(3D) data structure is about 2.4 times more expensive than the $IA^*(3D)$ data structure. Note that the IS(3D) and SIG(3D) data structures coincide, if restricted to manifolds. The specialization of the X-Maps data structure [CK10] to simplicial 3-complexes, discussed in Section 3.3.3, is about 5.8 times more expensive than the $IA^*(3D)$ data structure. The FE data structure [DL89], which we discussed
Table 5.5: Storage costs of the IS(3D) ($S_{IS}^{3D}$) and IA$^*$ (3D) ($S_{IA}^{3D}$) data structures, which represent manifold simplicial 3-complexes. Recall that, given a simplicial 3-complex $\Sigma$, $s_j$ is the number of $j$-simplices in $\Sigma$, with $0 \leq j \leq 3$.

In Section 3.3.1, is about 4.1 times more expensive than the IA$^*$ (3D) data structure. The IG(3D) data structure is about 3.4 times more expensive than the IA$^*$ (3D) representation. Finally, the CHF data structure [LLLV05] is about 1.5 times more expensive than the IA$^*$ (3D) representation. Recall that the CHF data structure is equivalent to the Corner Table [RSS01] and SVOT [GR09] representations. To the best of our experience, the IA$^*$ (3D) data structure is one of the most compact data structure for representing manifold volumetric shapes, although it can represent non-manifold simplicial shapes.

Now, we compare the NMIA and IA$^*$ data structures. To the best of our experience, they are the unique adjacency-based data structures for representing non-manifold simplicial 3-complexes. In [HVDF06] the authors demonstrate that the storage cost of the DLD data structure, discussed in Section 3.4.3, is comparable with the storage cost of the NMIA representation. Table 5.5 summarizes storage costs of the IS(3D), NMIA, and IA$^*$ (3D) data structures with all the arbitrary simplicial 3-complexes used in our tests. Recall that, given a simplicial 3-complex $\Sigma$, $s_j$ is the number of $j$-simplices in $\Sigma$, with $0 \leq j \leq 3$. Then, $s_1^1$ and $s_2^2$ are, respectively, the number of top edges and top triangles in $\Sigma$. Finally, $s_0^0$ and $s_1^1$ are, respectively, the number of non-manifold vertices and edges in $\Sigma$.

Table 5.5: Storage costs of the IS(3D) ($S_{IS}^{3D}$), NMIA ($S_{NM}$), and IA$^*$ (3D) ($S_{IA}^{3D}$) data structures with all the arbitrary simplicial 3-complexes used in our tests. Recall that, given a simplicial 3-complex $\Sigma$, $s_j$ is the number of $j$-simplices in $\Sigma$, with $0 \leq j \leq 3$. Then, $s_1^1$ and $s_2^2$ are, respectively, the number of top edges and top triangles in $\Sigma$. Finally, $s_0^0$ and $s_1^1$ are, respectively, the number of non-manifold vertices and edges in $\Sigma$. The “Sierpinski” shape is a courtesy of Harish Doraiswamy and Vijay Natarajan [DN12].
These tests show that the IA* (3D) data structure is more compact than the IS(3D) and NMIA data structures, and also than all the data structures analyzed in Section 4.4.2. Specifically, the IS(3D) representation is about 2.4 times more expensive than the IA* (3D) data structure. As a consequence, the IG(3D) and SIG(3D) representations are, respectively, about 3.3 and 2.42 times more expensive than the IA* (3D) data structure. Finally, the NMIA representation is only 4% more expensive than the IA* (3D) data structure.

We compare the NMIA and IA* (3D) representations, by evaluating their storage costs in terms of which topological relations they encode. Table 5.6 summarizes contributions to storage costs $S_{NM}$ and $S_{IA}^{3D}$, for all the simplicial 3-complexes analyzed in Table 5.4 and Table 5.5. Here, we denote the storage cost of boundary relations in the NMIA and IA* (3D) data structures as $B = 2s_1^3 + 3s_2^3 + 4s_3$. We denote the total number of 2- and 3-clusters incident at all the vertices as $K_0^3 + K_0^3$, and the number of vertex-based clusters incident at all the vertices as $C^1$. Then, we denote the total number of indices needed for encoding non-manifold adjacencies along any edge in the IA* (3D) data structure as $N_{IA}^3$. In the NMIA data structure, we denote the total number of records, related to a non-manifold edge, respectively, in top triangles and tetrahedra as $N_{NM}^3$ and $N_{NM}^3$. Finally, we denote the maximum and minimum number of triangles incident at a non-manifold edge $\tau$, respectively, as $\min N_{\tau}^3$ and $\max N_{\tau}^3$.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$B$</th>
<th>$K_0^3 + K_0^3$</th>
<th>$C^1$</th>
<th>$N_{IA}^3$</th>
<th>$N_{NM}^3$</th>
<th>$N_{NM}^3$</th>
<th>$\min N_{\tau}^3$</th>
<th>$\max N_{\tau}^3$</th>
<th>$\Delta N_{IA}^3$ (3D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>16k</td>
<td>1.2k</td>
<td>1.2k</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Cylinder</td>
<td>20.8k</td>
<td>1.3k</td>
<td>1.3k</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Gargoyle</td>
<td>40k</td>
<td>2.7k</td>
<td>2.7k</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Rings</td>
<td>32.4k</td>
<td>2.5k</td>
<td>2.5k</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Torus 3D</td>
<td>43.6k</td>
<td>2.3k</td>
<td>2.3k</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Arc</td>
<td>0.68k</td>
<td>53</td>
<td>53</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Balloon</td>
<td>8.35k</td>
<td>1.1k</td>
<td>1.1k</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Bucket</td>
<td>300</td>
<td>66</td>
<td>50</td>
<td>16</td>
<td>16</td>
<td>32</td>
<td>1</td>
<td>1</td>
<td>0.1k</td>
</tr>
<tr>
<td>Chime</td>
<td>1.48k</td>
<td>265</td>
<td>247</td>
<td>9</td>
<td>9</td>
<td>12</td>
<td>1</td>
<td>3</td>
<td>0.3k</td>
</tr>
<tr>
<td>Flasks</td>
<td>14k</td>
<td>134k</td>
<td>1.3k</td>
<td>68</td>
<td>68</td>
<td>72</td>
<td>1</td>
<td>2</td>
<td>0.2k</td>
</tr>
<tr>
<td>Halves</td>
<td>3.2k</td>
<td>262</td>
<td>253</td>
<td>-</td>
<td>-</td>
<td>18</td>
<td>-</td>
<td>-</td>
<td>0.05k</td>
</tr>
<tr>
<td>Sierpinsk</td>
<td>65.5k</td>
<td>65.5k</td>
<td>65.5k</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Teapot</td>
<td>40.3k</td>
<td>3.9k</td>
<td>3.8k</td>
<td>75</td>
<td>75</td>
<td>94</td>
<td>1</td>
<td>6</td>
<td>0.2k</td>
</tr>
<tr>
<td>Wheel</td>
<td>4.69k</td>
<td>0.45k</td>
<td>0.4k</td>
<td>32</td>
<td>32</td>
<td>80</td>
<td>1</td>
<td>2</td>
<td>0.24k</td>
</tr>
</tbody>
</table>

Table 5.6: Contributions to $S_{NM}$ and $S_{IA}^{3D}$, for all the simplicial 3-complexes analyzed in Table 5.4 and Table 5.5. Here, we denote the storage cost of boundary relations in the NMIA and IA* (3D) data structures as $B = 2s_1^3 + 3s_2^3 + 4s_3$. We denote the total number of 2- and 3-clusters incident at all the vertices as $K_0^3 + K_0^3$, and the number of vertex-based clusters incident at all the vertices as $C^1$. Then, we denote the total number of indices needed for encoding non-manifold adjacencies along any edge in the IA* (3D) data structure as $N_{IA}^3$. In the NMIA data structure, we denote the total number of records, related to a non-manifold edge, respectively, in top triangles and tetrahedra as $N_{NM}^3$ and $N_{NM}^3$. Finally, we denote the maximum and minimum number of triangles incident at a non-manifold edge $\tau$, respectively, as $\min N_{\tau}^3$ and $\max N_{\tau}^3$. The “Sierpinski” shape is a courtesy of Harish Doraiswamy and Vijay Natarajan [DN12].
These tests experimentally justify theoretical properties of the NMIA and IA* (3D) data structures, which we discussed in this chapter.

First, we evaluate the storage cost $B = 2s_1^t + 3s_2^t + 4s_3$ of boundary relations, encoded in both the NMIA and IA* (3D) data structures. In the NMIA data structure, $B$ is about 45.6% of $S_{NM}$, while, in the IA* (3D) data structure, $B$ is about 46.5% of $S_{IA^*}^0$. Remaining relations require, on average, about 54.4% and 53.5% of $S_{NM}$ and $S_{IA^*}^{3D}$, respectively.

The difference $\Delta_{IA^*}^{NM} (3D)$ depends on two aspects, namely the difference between the number of connected components and clusters, for all the vertices in $\Sigma$, plus the different encoding of non-manifold adjacencies along any non-manifold edge in $\Sigma$.

As demonstrated in Section 4.4.2, in the star of any vertex $v$ in $\Sigma$, the total number $k^2(v) + k^3(v)$ of clusters is usually larger than the number $C^0$ of connected components of $Lk(v)$. As a consequence, $K_0^1 + K_0^2 \geq C^1$. In our tests, the difference $\|C^1 - K_0^2 - K_0^3\|$ is, on average, only 4% of $S_{IA^*}^0$. In any case, this difference may be larger with more complex 3D shapes than ours. If $\Sigma$ is regular, like the “Halves” and “Sierpinski” shapes, then $\|C^1 - K_0^2 - K_0^3\| = 0$. Finally, if $\Sigma$ is manifold, then $K_0^3 = 0$ and $C^1 = K_0^2 = s_0$. As a consequence, $\|C^1 - K_0^2 - K_0^3\| = 0$.

The different encoding for non-manifold edges play a key role in $\Delta_{IA^*}^{NM} (3D)$. Recall that the IA* (3D) data structure encodes only one record for each non-manifold edge $\tau$ shared by at least one top triangle, namely partial co-boundary relation $R^*_{1,2}(\tau)$. For instance, the IA* (3D) data structure does not encode any information for non-manifold edge $e$ in Figure 5.2, which is shared only by two tetrahedra. Conversely, the NMIA data structure encodes partial relation $R^*_{1,cl}(\tau)$, for each non-manifold edge $\tau$, shared by several top triangles and 3-clusters. Thus, it is interesting to highlight the behavior of the NMIA and IA* (3D) data structures, according to the number of non-manifold edges in the star of top triangles.

If the reference simplicial $d$-complex $\Sigma$ is regular, then all the top simplices are maximal, and, thus, $K_0^2 = 0$ and $C^1 = K_0^3$. Non-manifold edges in $\Sigma$ are shared only by 3-clusters, and, thus, $N^t_{IA^*} = N^t_{NM} = 0$, like in the “Halves” shape. Hence, $\Delta_{IA^*}^{NM} (3D) = 3N^3_{NM}$, and, thus, the NMIA data structure is more expensive than the IA* (3D) data structure.

Similarly, if there are no top triangles in $\Sigma$, then partial relations $R^*_{1,2}$ and $R^*_{2,cl}$ are empty, like in the “Arc” shape. As a consequence, $\Delta_{IA^*}^{NM} (3D) = 3N^3_{NM}$, and, thus, the NMIA data structure is more expensive than the IA* (3D) data structure. Clearly, if there are no non-manifold edges in $\Sigma$, then $N^t_{IA^*} = N^t_{NM} = N^3_{NM} = 0$, like in the “Balloon” and “Sierpinski” shapes. Since there are only non-manifold vertices in $\Sigma$, any connected component in the star of a non-manifold vertex $v$ corresponds either to a 2-cluster or to a 3-cluster in $S_{IA^*}$ (3D). As a consequence, $K_0^3 + K_0^2 = C^1$, and, thus, the NMIA and IA* (3D) data structures coincide.

In general, it is clear that $2N^3_{NM} + 3N^3_{NM} > N^t_{IA^*}$. For instance, in the “Bucket” shape, there are 16 non-manifold edges: here, the star of each non-manifold edge $\tau$ contains one top triangle $f_\tau$ and one tetrahedron $t_\tau$. Hence, the IA* (3D) data structure encodes partial co-boundary relation $R^*_{1,2}(\tau)$, consisting only of the top triangle $f_\tau$. Conversely, the NMIA data structure encodes two
records, describing, respectively, partial relation $R^i_{1,c,t}(\tau)$ with respect to $f_r$ and $t_r$. In this case, $N_{IA^*}^i = 16$, and $N_{NM}^i = N_{NM}^3 = 2 \times N_{IA^*}^i = 32$. As a consequence, $2N_{NM}^2 + 3N_{NM}^3 \approx 10 \times N_{IA^*}^i$. Finally, it is interesting to compare contributions in $\Delta_{IA^*}(3D)$, namely $C^i_1 - K_0^i + K_0^3$ and $2N_{NM}^2 + 3N_{NM}^3 - N_{IA^*}^i$. Note that $K_0^3 + K_0^i \geq C_i$ and $N_{IA^*}^i \leq 2N_{NM}^2 + 3N_{NM}^3$. We can give an informal characterization of this comparison, which show that the $IA^*(3D)$ data structure is more compact than the NMIA data structure.

Suppose to represent the simplicial 3-complex in Figure 5.7 through the NMIA and $IA^*(3D)$ data structures. Here, the star of each vertex is formed by only one connected component, and, thus, $C^i_1 = 7$. Conversely, the star of vertex 1 is formed by one 2-cluster and two 3-clusters, while the star of vertices 4 and 5 is formed by one 2-cluster and one 3-cluster. The star of the remaining vertices consists of only one 3-cluster. Hence, $K_0^2 = 3$ and $K_0^3 = 8$.

![Figure 5.7](image)

Figure 5.7: In this simplicial 3-complex, the star of each vertex is formed by only one connected component, corresponding to several clusters of different dimension. We have already shown this simplicial 3-complex in Figure 4.3.

In the $IA^*(3D)$ data structure, we encode partial relations $R^i_{1,2}(e_3) = \{df_1\}$ and $R^i_{1,2}(e_4) = \{df_1\}$, hence $N_{IA^*}^i = 2$. Conversely, in the NMIA data structure, we encode partial relations $R^i_{1,cl}(e_3) = \{t_1, t_1\}$ and $R^i_{1,cl}(e_4) = \{t_2, t_2\}$ for the 2-cluster formed by the top triangle $df_1$. We encode partial relations $R^i_{1,cl}(e_3) = \{df_1, df_1\}$ and $R^i_{1,cl}(e_4) = \{df_1, df_1\}$, respectively, for 3-clusters formed by tetrahedra $t_1$ and $t_2$. As a consequence, $N_{NM}^2 = 4 = N_{NM}^3$. Therefore, it is simple to state that $\Delta_{IA^*}(3D) = 2$, and, thus, the $IA^*(3D)$ data structure is more compact than the NMIA representation.

We can generalize this example, by increasing complexity of the reference simplicial 3-complex $\Sigma$. Note that, for any vertex $v$ in $\Sigma$, $\|h_v - k_0^i(v) - k_0^3(v)\|$ admits the maximum value when the star of $v$ is formed by only one connected component, corresponding to several 2-clusters and 3-clusters, which are connected through non-manifold edges in $St(v)$, like the star of vertex $v = 1$ in Figure 5.7. Here, $h_v = 1$ and $k_0^3(v) + k_0^3(v) > 1$. Each non-manifold edge $\tau$ in $St(v)$ is shared by at least one 2-cluster, represented by a top triangle $f_r$, and one 3-cluster, represented by a tetrahedron $t_r$. As a consequence, $N_{IA^*}^3 \approx k_0^3(v)$, $N_{IA^*}^2 \approx k_0^3(v)$, and $N_{IA^*}^3 \approx k_0^3(v)$. Thus, if we repeat this analysis for each vertex $v$, then it is quite clear that the $IA^*(3D)$ data structure is more compact than the NMIA representation.
Chapter 6

Rapid Prototyping of Topological Data Structures

In this chapter, we describe our dimension-independent and extensible Mangrove Topological Data Structure (Mangrove TDS) framework, targeted to the fast prototyping of topological data structures. As discussed in [DFH05], several topological data structures have been introduced in the literature. A framework which supports a wide number of topological data structures under a common application interface is a valuable tool, especially from the applicative point of view.

The Mangrove TDS framework exploits a graph-based representation of a topological data structure, which allows designing a wide number of representations. A topological data structure can be described through a graph-based representation, which we call a mangrove. A mangrove captures connectivity information, provided by topological relations in a simplicial complex [DF03]. A topological data structure can be described as a specific mangrove: for instance, the IS-graph, discussed in Section 4.1, is a graph-based representation of the Incidence Simplicial (IS) data structure [DFHPC10]. The Mangrove TDS framework is based on mangroves in order to design a topological data structure.

Several applications require to access all the elements in a simplicial shape, thus it is important to support a random access to all the simplices, although we are using a topological data structure, like the IA* data structure, which encodes only vertices and top simplices. The Mangrove TDS framework offers an implicit description for a simplex not directly encoded in a topological data structure, which we call a ghost simplex.

In Section 6.1, we introduce the Mangrove TDS framework and discuss its properties.

In order to prove the validity of our approach, we have designed and implemented several data structures, including the Incidence Simplicial (IS) data structure [DFHPC10], and the Generalized Indexed data structure with Adjacencies (IA*) [CDFW11], which we have introduced in Sections 4.1 and 5.1, respectively. Specifically, in Section 6.2, we prove that the IS data structure can be
implemented in our framework, while, in Section 6.3, we describe how the IA* data structure can be defined in the Mangrove TDS framework. We describe all the navigation and construction algorithms for these topological data structures.

The Mangrove TDS framework is also a common basis for comparing efficiency of topological data structures. In Chapter 7, we present comparisons about performances of all the topological data structures discussed in Chapters 4 and 5. For each data structure, we describe navigation and construction algorithms.

The complete implementation of the Mangrove TDS framework, including all the data structures we have discussed, is contained in the Mangrove TDS Library, which we plan to release in public domain for the community.

6.1 The Mangrove Topological Data Structure Framework

In this section, we introduce our Mangrove Topological Data Structure (Mangrove TDS) framework, and discuss its properties.

In Section 6.1.1, we describe a graph-based representation of any topological data structure, which we call a mangrove. In Section 6.1.2, we propose a compact encoding for this graph-based representation, where the key information is associated with nodes of a mangrove. In Section 6.1.3, we introduce an implicit description for any simplex not directly encoded in a topological data structure, which we call a ghost simplex. Finally, in Section 6.1.4, we describe operations which are currently supported by the Mangrove TDS framework.

Note that our design choices fully satisfy requirements recently introduced in [SB11] regarding properties of a framework for representing simplicial shapes.

6.1.1 Mangroves

In this section, we describe a graph-based representation of a topological data structure, which we call a mangrove. The Mangrove TDS framework exploits this representation for designing a topological data structure. Here, we restrict our attention to simplicial complexes, but a mangrove can be also defined on cell complexes with minor modifications.

Recall that a simplicial shape is discretized by a simplicial d-complex Σ. As discussed in Section 2.4, we exploit their connectivity information through topological relations. These latter provide an effective framework for topological data structures, described in terms of the entities and relations they encode [DF03].

In this context, a topological data structure, which describe a simplicial d-complex Σ can be described by a mangrove, which is a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$, where:
• a node $n_{\sigma}$ in $\mathcal{N}$ describes a $p$-simplex $\sigma$ in $\Sigma$, with $0 \leq p \leq d$, and it is called a node of dimension $p$;

• each arc $(n_{\sigma}, n_{\sigma'})$ in $\mathcal{N}$ connects two nodes which correspond, respectively, to a $k$-simplex $\sigma$ and to a $m$-simplex $\sigma'$ such that $\sigma \sim_{k,m} \sigma'$.

We can classify arcs with respect to which topological relations they represent. An arc $(n_{\sigma}, n_{\sigma'})$ between a node $n_{\sigma}$ of dimension $k$ and a node $n_{\sigma'}$ of dimension $m$ can be:

• a boundary arc if and only if it represents a boundary relation $\sigma \sim_{k,m} \sigma'$, with $k > m$;

• a co-boundary arc if and only if it represents a co-boundary relation $\sigma \sim_{k,m} \sigma'$, with $k < m$;

• an adjacency arc if and only if it represents an adjacency relation $\sigma \sim_{k,k} \sigma'$.

We say that a mangrove $G$ is global if it encodes all the simplices in $\Sigma$, otherwise $G$ is a local mangrove. For instance, the IS data structure [DFHPC10] can be represented by a global mangrove, while the IA* data structure [CDFW11] can be described by a local mangrove.

In this way, it is possible to represent a topological data structure for simplicial complexes, overcoming several restrictions regarding representation of non-manifold simplicial shapes in the most common frameworks in the literature, like the Computational Geometry Algorithms Library (CGAL) [Ket99, CGA11], the OpenMesh Library [BSBK02, OMS11], the Half-edge Mesh Library (MeshLib) [Gu11], and the Visual Computing Graphics (VCG) Library [VCG04]. Specifically, most of frameworks offer a wide range of operators and tools for handling cellular shapes, but only on a specific data structure. For instance, the CGAL, MeshLib, and OpenMesh libraries exploit the HE data structure for cell complexes [Man87], while the VCG Library exploits a face-based representation. Moreover, few frameworks support regular shapes with non-manifold singularities, like the OpenMesh Library, but restricted only on non-manifold vertices. The MeshLib Library supports only triangulations, while the VCG Library supports only triangulations and quads.

### 6.1.2 Explicit Representation of Simplices

In this section, we describe a compact encoding of a mangrove, which we exploit in the Mangrove TDS framework. Here, the key information is associated only with nodes of a mangrove, and we do not explicitly encode any arc.

Here, we focus our attention only on simplices directly encoded in a mangrove $G$, which describes connectivity information in a simplicial $d$-complex $\Sigma$. Specifically, if a mangrove is global, then we consider all the simplices in $\Sigma$. These $p$-simplices (with $0 \leq p \leq d$) are stored in a dynamic array, which we call a SimplicesContainer array. In this context, there is a SimplicesContainer array for each collection of $p$-simplices which we encode in $G$. These arrays support a garbage collector mechanism, and thus they support update operations, like the insertions and removals of simplices. Each location of these arrays stores a simplex in $\Sigma$. Note that it is possible to store simplices
only in a SimplicesContainer array. In this way, we provide an elegant and uniform solution for storing simplices. As experimentally demonstrated in [SB11], an array-based storage scheme for topological entities is surely more compact and faster than the list-based scheme exploited in [Ket99, CGA11]. This design choice allows accessing a simplex in constant time.

In order to sequentially access simplices, we provide safe iterators on a SimplicesContainer array. Each iterator stores a reference to the current simplex and to the current mangrove. This latter is used to automatically detect and skip locations marked as “deleted” by the garbage collector mechanism. This solution is also adopted in most of frameworks in the literature [CGA11, OMS11, SB11], just to mention few.

Each $p$-simplex $\sigma$ directly encoded in a mangrove $G$ has an unique identifier, formed by the pair $(p,i)$, where $p$ is the dimension of $\sigma$, and $i$ is the position of $\sigma$ in the SimplicesContainer array used for storing $p$-simplices. This identifier is called a SimplexPointer reference. It allows accessing a simplex in constant time due to our array-based storage scheme. In fact, $p$ identifies the SimplicesContainer of interest, which contains the required simplex in position $i$. In other words, a SimplexPointer reference is a pair of integer values, which are sufficient to refer to simplices in the most common applications.

The key information is stored for each simplex $\sigma$: specifically, we encode a subset of topological relations in which $\sigma$ is involved. For the sake of simplicity, we do not encode explicitly any arc of $G$, but only endpoints different from the node $n_\sigma$ associated with $\sigma$. In other words, a $p$-simplex $\sigma$ can be represented as a record Simplex, defined as follows:

```plaintext
class Simplex
{
    array< SimplexPointer > [p+1] bnd;
    hash< int, int > [p+1] aux_bnd;
    list< SimplexPointer > cob;
    array< list< SimplexPointer > >[p+1] adj;
}
```

The fixed-length array $bnd$ contains endpoints of $p+1$ boundary arcs outgoing from $n_\sigma$. In other words, it contains $p+1$ SimplexPointer references to $k$-simplices (with $0 \leq k < p$) belonging to the combinatorial boundary of $\sigma$. Recall that a $p$-simplex $\sigma$ is completely generated by its $p+1$ vertices, and its boundary can be expressed by $p+1$ subfaces of dimension $p-1$. We can also encode oriented boundary of $\sigma$ (see Section 9.1) by sorting the array $bnd$. The content of this array depends on the specific topological data structure we are using: for instance, the array $bnd$ may contain either references to vertices in partial boundary relation $R_{p,0}^*(\sigma)$ with the IA* data structure [CDFW11], or references to simplices in boundary relation $R_{p,p-1}(\sigma)$ with the IS data structure [DFHPC10].

In any case, this information is not always enough for all the purposes. In some applications, we need to access simplices bounding a $p$-simplex $\sigma$, but not directly encoded in $bnd$. Note
that, in the Mangrove TDS framework, simplices can be stored only in any Simplex record in a SimplicesContainer array. For instance, in the NMIA data structure (see Section 5.3), we also encode information about non-manifold edges of a tetrahedron \( \sigma \), namely the partial relation \( R^*_{\sigma} \). Thus, we need to store auxiliary information about simplices bounding \( \sigma \). In order to reduce overhead, we use a fixed-length array \( aux_{bnd} \) containing \( p + 1 \) hash tables, which allow encoding SimplexPointer references to auxiliary boundary simplices for \( \sigma \). Hast table \( aux_{bnd}[k] \) encodes information on the \( \binom{p+1}{k+1} \) subfaces of dimension \( k \) (with \( k < p \)) for a \( p \)-simplex \( \sigma \). Let \((p, j)\) be the SimplexPointer reference which represents \( \sigma \), then any information about the \( i \)-th \( k \)-face of \( \sigma \) can be stored as the pair \((i, j)\) in \( aux_{bnd}[k] \). Thus, retrieving information on a \( k \)-face of \( \sigma \) has a time complexity in \( O \left( \log \binom{p+1}{k+1} \right) \), in the worst case. This schema does not depend on which order we exploit for storing and identifying faces of a \( p \)-simplex \( \sigma \). In Section 6.1.3, we propose a hierarchy of faces to be enforced while building a specific mangrove.

The variable-length array \( cob \) contains endpoints of some co-boundary arcs outgoing from \( n_\sigma \). In other words, it encodes several SimplexPointer references to \( k \)-simplices incident at \( \sigma \), with \( k > p \). Also in this case, the content of the array \( cob \) depends on the specific topological data structure we are using; for instance, it may contain either references to all the \( (p+1) \)-simplices incident in \( \sigma \) with the IG data structure [Ede87], or references to representative simplices for each cluster incident in \( \sigma \) with the SIG data structure [DFGH04].

Finally, the fixed-length array \( adj \) contains a description of adjacency arcs connecting \( n_\sigma \). Recall that two \( p \)-simplices (with \( p \neq 0 \)) are adjacent if they share a \( (p-1) \)-simplex, and two vertices are adjacent if they are connected by a common edge. Any location \( adj[k] \), with \( 0 \leq k \leq p \), corresponds to a \( (p-1) \)-face \( \sigma_k \) of \( \sigma \), and contains SimplexPointer references to simplices related to the adjacent relation of \( \sigma \) along \( \sigma_k \). As discussed in Chapter 5, it is not mandatory that they refer to \( p \)-simplices, since adjacency relation implicitly provide information about non-manifold singularities. Specifically, \( adj[k] \) contains at most one reference to a \( p \)-simplex \( \sigma' \) if \( \sigma' \) and \( \sigma \) are adjacent along a manifold face \( \sigma_k \) of \( \sigma \). Otherwise, \( adj[k] \) contains a SimplexPointer reference to a \( (p-1) \)-simplex \( \sigma_k \) whose co-boundary is not empty. This solution has been already discussed in Chapter 5 regarding non-manifold adjacency for the IA*, TS, and NMIA data structures.

### 6.1.3 Implicit Representation of Simplices

In this section, we propose an implicit representation of simplices not directly encoded in a mangrove, which we call ghost simplices. Specifically, we can exploit this representation only with local mangroves. Recall that a local mangrove encodes only of a subset of simplices in a simplicial \( d \)-complex \( \Sigma \). For instance, the TS, NMIA, and IA* data structures [DFH03, DFMPS04, CDFW11] encode only vertices and top simplices, and they can be represented through local mangroves.

A trivial description of a simplex \( \sigma \) consists of its list of vertices \( V_\sigma \) which generates \( \sigma \). In any case, this representation, which we call explicit representation of \( \sigma \), introduces several drawbacks. Specifically, it requires a lot of knowledge about the simplices in a simplicial shape. In fact, we need to know what vertices effectively belong to the boundary of a simplex \( \sigma \), and this is almost
difficult and impossible for huge shapes, formed by millions of simplices [ABA06]. Also, it does not allow designing a representation with a constant number of elements, more simple to manage, since the list of vertices of $\sigma$ is variable according to its dimension. In other words, edges are described by two vertices, a triangle by three vertices, and so on. Furthermore, this representation does not provide any structural and topological information about $\sigma$. For instance, it is impossible to understand if $\sigma$ is a top simplex, or to retrieve a top simplex incident at $\sigma$ without executing a query. This may be a problem with several data structures: for instance, in the NMIA data structure, co-boundary relation $R_{2,3}(f)$ for a triangle $f$ is local if $f$ is represented through its vertices, otherwise it is optimal (see Section 5.3).

Here, we propose an implicit encoding of a simplex $\sigma$ not directly encoded in a mangrove. The key idea consists of considering a simplex $\sigma$ as either a top simplex, or a face of a top simplex $\sigma'$. Following [HVDF06], a simplicial $d$-complex $\Sigma$ is completely described by its top simplices, thus this representation is valid for any simplex in $\Sigma$. We call this representation as a ghost simplex. We identify a simplex $\sigma$ by a tuple $(t,i,ct,ci)$, which we call GhostSimplexPointer reference, where:

- $t$ is the dimension of the reference top simplex $\sigma'$, i.e., $t = \text{dim}(\sigma')$, which can be retrieved by the FATHER_TYPE function, defined as $\text{FATHER_TYPE}((t,i,ct,ci)) = t$;
- $i$ is the unique identifier of $\sigma'$ in the collection $\Sigma'$ of top simplices, which can be retrieved by the FATHER_ID function, defined as $\text{FATHER_ID}((t,i,ct,ci)) = i$;
- $ct$ is the dimension of $\sigma$, with $ct = \text{dim}(\sigma) \leq t$, which can be retrieved by the CHILD_TYPE function, defined as $\text{CHILD_TYPE}((t,i,ct,ci)) = ct$;
- $ci$ is the unique identifier of $\sigma$, which is considered as a face of dimension $ct$ of $\sigma'$. As a consequence, $ci < \binom{t+1}{ct+1}$ [Ede87]. It can be retrieved by the CHILD_ID function, defined as $\text{CHILD_ID}((t,i,ct,ci)) = ci$.

It is clear that the reference top simplex, which must be directly encoded, can be identified by the SimplexPointer $(t,i)$. For instance, the GhostSimplexPointer reference $(3,1,1,5)$ identifies the fifth edge of the top tetrahedron identified by the SimplexPointer reference $(3,1)$. A GhostSimplexPointer reference directly represent also top simplex $\sigma'$, e.g., when $ct = t$ and $ci = 0$.

This implicit representation of a simplex $\sigma$ is not unique, since $\sigma$ may belong, in general, to several top simplices, and, thus there are several GhostSimplexPointer references which refer to $\sigma$. In any case, it provides a fast identification of simplices, and it requires less information to be known than the list of its vertices of $\sigma$. In fact, it requires only to know how many top simplices belong to a simplicial $d$-complex $\Sigma$, and their dimension. In other words, we need only $s_k$, for $0 \leq k \leq d$.

Our experimental results, provided in Section 8.4, show that the number of top simplices does not exceed 30% of the total number of simplices in $\Sigma$.

The order of faces bounding a simplex $\sigma$ is application-dependent, and it must be enforced when building the related mangrove. We can exploit a slight modification of the approach used in [DFHPC10]. In this approach, we exploit the lexicographic order of vertices in $\Sigma$. Let $v_i$ and $v_j$
be two vertices identified, respectively, by the SimplexPointer references \((0, i)\) and \((0, j)\), then we say that \(v_i < v_j\) if and only if \(i < j\). Hence, a \(p\)-simplex \(\sigma\) is completely and uniquely generated by the ordered set of its vertices \(V = [v_0, \ldots, v_p]\). In this context, a \((p - 1)\)-face \(\lambda_i\) of \(\sigma\) is expressed as \(\lambda_i = [v_0, \ldots, \hat{v_i}, \ldots, v_p]\), where we discard vertex \(v_i\), with \(i = 0, \ldots, p\).

It is reasonable to exploit this approach, since the most common exchange format for a simplicial \(d\)-complex \(\Sigma\) consists of a collection of top simplices, described by their vertices. This representation is known as a *soup of top simplices*. Note that this construction schema is mandatory for computing incidence matrices of a simplicial complex \(\Sigma\), as discussed in Chapter 9.

However, it is not simple to retrieve the list of vertices of a simplex \(\sigma\) identified by a GhostSimplexPointer reference as a face of a top \(t\)-simplex \(\sigma'\), since we have to recompute the hierarchy of faces each time. Our objective consists of defining a lookup table, which relates vertices of \(\sigma'\) and \(\sigma\) according to the GhostSimplexPointer reference of \(\sigma\). If we build the hierarchy of faces in \(\sigma'\) by exploiting positions of vertices in \(V_{\sigma'}\) instead of indices of their vertices, then it is possible to solve this problem with a small overhead.

Here, we consider the reference top \(t\)-simplex \(\sigma'\), which is completely generated by the set \(V_{\sigma'} = [v_0, \ldots, v_t]\) of its vertices. Thus, we can describe \(\sigma'\) by vertices in positions \(0, \ldots, t\) in its boundary. As a consequence, we can reduce \(\sigma'\) to positions \([0, \ldots, t]\). Given a \(p\)-face \(\eta\) of \(\sigma'\) (with \(0 < p \leq t\)) described by positions \([a_0, \ldots, a_i, \ldots, a_p]\) of vertices in \(\sigma'\), we can describe a \((p - 1)\)-face \(\lambda_i\) of \(\eta\) by discarding position \(a_i\). Hence, \(\lambda_i = [a_0, \ldots, a_{i-1}, a_{i+1}, \ldots, a_p]\).

Thus, we can recursively generate \(p\)-faces of \(\sigma'\) (for each \(0 \leq p \leq t\)) in terms of the positions of vertices in \(V_{\sigma'}\). All the \(p\)-faces are stored in the same order as they are generated. Clearly, we must discard duplicates of the same \(p\)-face. Moreover, we do not apply this schema in order to generate vertices, since they are already sorted in \(V_{\sigma'}\). Figure 6.1(a) shows complete hierarchy of all the faces for a tetrahedron \(\sigma'\), expressed in terms of positions of vertices in \(V_{\sigma'}\). Each node refers to a face of \(\sigma'\) and contains positions of vertices in \(V_{\sigma'}\) to be considered. In this way, we can easily relate a simplex described by a GhostSimplexPointer reference and its vertices. Given a GhostSimplexPointer reference \((t, i, ct, ci)\), the LOOKUP function returns positions of vertices to be considered for the \(ci\)-th simplex of dimension \(ct\) from the lookup table related to top \(t\)-simplices. For instance, given the GhostSimplexPointer reference \((3, 0, 1, 5)\), the LOOKUP function returns positions \([0, 1]\) in \(V_{\sigma'}\), as shown in Figure 6.1(a).

During this process, a unique identifier is assigned to each \(p\)-face \(\eta\) of \(\sigma'\), which is unique with respect to the list of \(p\)-faces of \(\sigma'\). Hence, we can also encode boundary of a \(p\)-face \(\eta\) in terms of identifiers of all the \((p - 1)\)-faces of \(\eta\). Figure 6.1(b) shows complete hierarchy of faces of \(\sigma'\), expressed in terms of identifiers of their immediate subfaces. Each node refers to a face \(\eta\) of \(\sigma'\) and contains indices of faces of \(\sigma'\) in the immediate boundary of \(\eta\).

These two representations are equivalent, and simulate an *Incidence Graph* [Ede87] built, respectively, on the positions of vertices in \(V_{\sigma'}\) and indices of subfaces for a top \(t\)-simplex \(\sigma'\). As a consequence, we can also retrieve incident and adjacency faces to a given subface of \(\sigma'\). Given
a \textit{GhostSimplexPointer} reference \((t, i, ct, ci)\), the \textsc{FACES} function returns indices of immediate subfaces of the simplex described by \((t, i, ct, ci)\). For instance, given the \textit{GhostSimplexPointer} reference \((3, 0, 2, 3)\), which correspond to a triangle formed by vertices in positions \([0, 1, 2]\), the \textsc{FACES} function returns edges in positions \([2, 4, 5]\) in \(v_{\sigma'}\), as shown in Figure 6.1(b).

![Diagram of tetrahedron with faces](image)

Figure 6.1: (a) The hierarchy of faces bounding a tetrahedron \(\sigma'\) expressed in terms of position of vertices in \(V_{\sigma'}\). (b) The same hierarchy of faces expressed in terms of immediate subfaces.

These two hierarchies can be computed as a pre-processing step. Also, they can be stored for each collection \(\Sigma_p^t\) of top \(p\)-simplices in a mangrove describing a simplicial \(d\)-complex \(\Sigma\). Recall that a top \(p\)-simplex has \(p + 1\) vertices and \(\binom{p + 1}{k + 1}\) faces of dimension \(k\). Hence, their storage cost, expressed in terms of integer values, is equal to:

\[
2 \sum_{p=1}^{d} \sum_{k=0}^{p} (k + 1)\binom{p + 1}{k + 1}
\]

In the remainder of this thesis, we confuse a simplex not directly encoded in a mangrove \(G\) with one of its \textit{GhostSimplexPointer} references.

### 6.1.4 Operations on Mangroves

In this section, we describe operations which are currently supported by the Mangrove TDS framework. Specifically, we give a set of operations which can be performed on a mangrove \(G\). For the sake of simplicity, in these operations, we access a simplex by a \textit{SimplexPointer} reference only with a global mangrove, otherwise we exploit a \textit{GhostSimplexPointer} reference.

In this context, the behavior of a mangrove \(G\) can be defined in terms of the following primitives, which operate on a \(p\)-simplex \(\sigma\) in \(G\):

- \textsc{BOUNDARY}(\(\sigma\)), which retrieves all the simplices bounding \(\sigma\), namely the combinatorial boundary of \(\sigma\):
• STAR(\(\sigma\)), which retrieves all the simplices which are incident at \(\sigma\), namely the star of \(\sigma\);

• ADJACENCY(\(\sigma\)), which retrieves all the \(p\)-simplices adjacent to \(\sigma\), namely the adjacency relation \(R_{p,p}(\sigma)\);

• LINK(\(\sigma\)), which retrieves all the simplices belonging to the link of \(\sigma\);

• IS\_MANIFOLD(\(\sigma\)), which checks if \(\sigma\) is manifold. Note that this operation is not decidable for simplicial complexes of dimension \(d \geq 6\), as discussed in Section 2.3. In our implementations, we limit our attention to simplicial complexes of dimension up to 3, thus this operation is decidable.

In several applications, we can associate auxiliary information with a simplicial shape. Any auxiliary information, which we call property, can be associated with a subset of simplices belonging to any mangrove \(G\). Here, we can recognize four types of properties:

• global properties, which represent information associated with all the simplices in \(G\) like, for instance, Boolean flags or semantic information;

• local properties, which represent information associated with a class of simplices in \(G\) with respect to their dimension like, for instance, Euclidean coordinates associated with each vertex in \(G\);

• sparse properties, which represent information associated with a random set of simplices in \(G\), for instance, information associated with top or non-manifold simplices in \(G\);

• ghost properties, which represent information associated with ghost simplices in a local mangrove \(G\).

Global and local properties are stored in arrays which are synchronized with simplices in a mangrove \(G\) after any update operation, as proposed in [BSBK02, OMS11]. Sparse properties are stored through a hash table build on SimplexPointer references of involved simplices. Ghost properties are associated with any ghost simplex \(\sigma\) by a hash table, built on the list of vertices of \(\sigma\), since this representation is unique. As demonstrated in Section 6.1.3, we can obtain the explicit representation of a ghost simplex in constant time.

In any case, in our implementation, all the properties can be dynamically allocated at run-time, in such a way that they are just allocated when needed and deleted afterwards. Clearly, there is a reduced overhead for accessing sparse and ghost properties, since we use a hash-table.

### 6.2 Implementing the Incidence Simplicial Data Structure

In this section, we describe a complete implementation of the Incidence Simplicial (IS) data structure [DFHPC10], introduced in Section 4.1, in the context of our Mangrove TDS framework.
Recall that the IS representation is an explicit, incidence-based, and dimension-independent data structure, which encodes all the simplices in an abstract simplicial complex, plus a subset of boundary and co-boundary relations for each simplex. As a consequence, the IS data structure is represented through a global mangrove.

In Section 6.2.1, we propose a complete description of the IS data structure in terms of internal data structures offered by the Mangrove TDS framework. In Sections 6.2.2 and 6.2.3, we discuss algorithms for executing the BOUNDARY and STAR queries, respectively. In Sections 6.2.4 and 6.2.5, we define algorithms for performing, respectively, the ADJACENCY and LINK queries. In Section 6.2.6, we describe an algorithm for executing IS_MANIFOLD query. Finally, in Section 6.2.7, we propose an algorithm for building the IS data structure from a soup of top simplices directly expressed in terms of their vertices.

6.2.1 Implementation of the Data Structure

In this section, we propose a complete description of the IS data structure in terms of internal data structures offered by the Mangrove TDS framework.

The IS data structure encodes all the simplices in any abstract simplicial $d$-complex $\Sigma$, thus we need $d+1$ SimplicesContainer arrays, one for each collection $\Sigma^p$ of $p$-simplices in $\Sigma$, with $0 \leq p \leq d$.

Each $p$-simplex $\sigma$ in $\Sigma$ is encoded through a Simplex record $r_\sigma$ in the SimplicesContainer array related to $\Sigma^p$. Each record $r_\sigma$ is assigned to a unique SimplexPointer reference. We exploit several SimplexPointer references for storing all the simplices in boundary and partial co-boundary relations $R_{p,p-1}(\sigma)$ and $R^*_{p,p+1}(\sigma)$. Specifically, in the bnd array of $r_\sigma$, we store $p+1$ SimplexPointer references, one for each $(p-1)$-simplex in $R_{p,p-1}(\sigma)$. Conversely, in the cob array of $r_\sigma$, we store several SimplexPointer references, one for each $(p+1)$-simplex in $R^*_{p,p+1}(\sigma)$.

6.2.2 Retrieving Boundary of a Simplex

In this section, we propose the a dimension-independent algorithm for retrieving all the simplices bounding a $p$-simplex $\sigma$ belonging to any abstract simplicial $d$-complex $\Sigma$ (with $0 < p \leq d$) described by the IS data structure. In other words, we provide an algorithm for the BOUNDARY query in our implementation of the IS data structure.

In Section 4.1.3.1, we have demonstrated that any boundary relation $R_{p,q}(\sigma)$, with $p > q$, is optimal. Also, it can be retrieved through a breadth-first traversal of the IS boundary graph. Specifically, any boundary relation $R_{p,q}(\sigma)$ can be extracted by encoded boundary relation $R_{p,p-1}(\sigma)$, and, then, by recursively combining boundary relation $R_{i,i-1}(\sigma')$, for all the $i$-faces $\sigma'$ of $\sigma$, for all $q < i < p$. This traversal ends if we reach a $q$-simplex.

In order to retrieve all the simplices bounding any $p$-simplex $\sigma$, we recursively combine boundary relation $R_{i,i-1}(\sigma')$, for all the $i$-faces $\sigma'$ of $\sigma$ until we reach all the vertices bounding $\sigma$, as proposed
in Algorithm 6.1.

Algorithm 6.1 BOUNDARY(σ) - IS data structure

Input: a SimplexPointer reference for a p-simplex σ in a simplicial d-complex Σ
Output: the set S of SimplexPointer references for all the simplices bounding σ

1: let \( p := \text{dim}(σ) \), \( S := \emptyset \)
2: if \( p \neq 0 \) then
3:     for all \( \sigma' \in R_{p,p-1}(σ) \) do
4:         if \( \sigma' \) is not visited then
5:             mark \( \sigma' \) as visited
6:             \( S := S \cup \{\sigma'\} \cup \text{BOUNDARY}(\sigma') \)
7:     end if
8: end for
9: end if
10: return \( S \)

Recall that a \( p \)-simplex \( σ \) has \( \sigma_k^p = \binom{p+1}{k+1} \) faces of dimension \( k \) (with \( 0 \leq k < p \)) [Ede87]. In Algorithm 6.1, we visit \( b_σ \) faces of dimension \( k \) bounding \( σ \), where:

\[
b_σ = \sum_{k=0}^{p-1} \binom{p+1}{k+1}
\]

which is a constant value, depending only on \( p \). The time complexity of the BOUNDARY query is \( O(1) \). Thus, the BOUNDARY query is optimal in the IS data structure.

### 6.2.3 Retrieving Star of a Simplex

In this section, we describe a dimension-independent algorithm for retrieving all the simplices incident at a \( p \)-simplex \( σ \) belonging to any abstract simplicial \( d \)-complex \( Σ \) (with \( 0 \leq p < d \)) described by the IS data structure. In other words, we provide an algorithm for the STAR query in our implementation of the IS data structure.

In Section 4.1.3.2, we have demonstrated that any co-boundary relation \( R_{p,q}(σ) \), with \( p < q \), can be retrieved by a breadth-first visit of the IS star-graph \( G^T_σ \).

All the simplices incident at \( σ \) are, for any \( h > p \), either top \( h \)-simplices incident at \( σ \), or faces bounding top \( h \)-simplices in \( St(σ) \). As a consequence, we must visit all the top simplices in \( St(σ) \), and select their faces, which are incident at \( σ \). We can perform this traversal using relations \( R_{k+1,k} \) and \( R^*_{k+1,k+2} \), for \( k > p \), directly encoded in the IS data structure. The basic step of this traversal consists of visiting a \( k \)-simplex \( σ' \) in \( St(σ) \). Here, we visit partial co-boundary relation \( R^*_k(σ') \): if \( k > p + 1 \), then we visit all the \((k-1)\)-faces bounding \( σ' \) directly encoded in boundary relation \( R_{k,k-1}(σ') \). Algorithm 6.2 provides a possible implementation of the STAR query for any \( p \)-simplex \( σ \).
Algorithm 6.2 STAR($\sigma$) - IS data structure

**Input:** a SimplexPointer reference for a $p$-simplex $\sigma$ in a simplicial $d$-complex $\Sigma$

**Output:** the set $S$ of the SimplexPointer references for all the simplices incident at $\sigma$

1: let $S := \emptyset$, $q$ be an empty queue, $p := \dim(\sigma)$
2: enqueue $\sigma$ in $q$
3: while $q$ is not empty do
4: dequeue $\sigma'$ from $q$
5: let $k := \dim(\sigma')$
6: if $k > p$ then
7: $S := S \cup \{\sigma'\}$
8: end if
9: for all $\sigma'' \in R^*_{k,k+1}($ $\sigma'$ $)$ do
10: if $\sigma''$ is not visited then
11: mark $\sigma''$ as visited
12: enqueue $\sigma''$ in $q$
13: end if
14: end for
15: if $k > p + 1$ then
16: for all $\sigma'' \in R_{k,k-1}($ $\sigma'$ $)$ do
17: if $\sigma''$ is in $\text{BOUNDARY}(\sigma'')$ then
18: if $\sigma''$ is not visited then
19: mark $\sigma''$ as visited
20: enqueue $\sigma''$ in $q$
21: end if
22: end if
23: end for
24: end if
25: end while
26: return $S$

Now, we analyze the time complexity of Algorithm 6.2. This traversal is equivalent to a breadth-first visit of the IS star-graph $G^IS_{\sigma}$. We visit each node and each arc in $G^IS_{\sigma}$ exactly once. The number of arcs is linear in the number of nodes in $G^IS_{\sigma}$, since each simplex is bounded by a constant number of faces. This traversal of $G^IS_{\sigma}$ is equivalent to retrieve all the top simplices in $St(\sigma)$ and to select their faces bounded by $\sigma$. Given a $k$-simplex $\sigma'$ incident at $\sigma$, with $k > p$, the number of all the faces of $\sigma'$ which are incident at $\sigma$ is bounded by:

$$\sum_{m=p}^{k} \binom{k+1}{m+1}$$

which is a constant value $C_{k,p}$ depending only on $k$ and $p$. Thus, if $\sigma^t_k$ is the number of top $k$-simplices in $St(\sigma)$, then the number of nodes in $G^IS_{\sigma}$ is linearly proportional to:

$$\sum_{k=p+1}^{d} C_{k,p} \sigma^t_k \approx \sigma^t_k$$

where $\sigma^t_k$ is the number of top simplices in $St(\sigma)$. 

120
As a consequence, the time complexity of Algorithm 6.2 is $O(\sigma_1^*)$. Generally speaking, the STAR query is local in the IS data structure. In any case, the number $\sigma_1^*$ of top simplices in $St(\sigma)$ is linear in $\|St(\sigma)\|$ only for simplicial 2-complexes and 3-complexes embedded in the Euclidean space $E^3$. Thus, in the IS data structure, the STAR query is optimal only for simplicial complexes embedded in $E^3$. Conversely, it is local for simplicial $h$-complexes, with $h \geq 4$.

### 6.2.4 Retrieving Simplices Adjacent to a Simplex

In this section, we describe a dimension-independent algorithm for retrieving all the $p$-simplices adjacent to a $p$-simplex $\sigma$ in any abstract simplicial $d$-complex $\Sigma$, namely the adjacency relation $R_{p,p}(\sigma)$, with $0 \leq p \leq d$. In other words, we provide an algorithm for the ADJACENCY query in our implementation of the IS data structure.

In Section 4.1.3.3, we have provided an algorithm for retrieving all the $p$-simplices adjacent to a given $p$-simplex $\sigma$ in $\Sigma$, with $0 \leq p \leq d$. Algorithm 6.3 provides the pseudo-code needed for performing these operations.

**Algorithm 6.3 ADJACENCY($\sigma$) - IS data structure**

**Input:** a SimplexPointer reference for a $p$-simplex $\sigma$ in a simplicial $d$-complex $\Sigma$

**Output:** the set $S$ of the SimplexPointer references for all the $p$-simplices adjacent to $\sigma$

```
1: let $p := \dim(\sigma)$, $S := \emptyset$
2: if $p=0$ then
3:   \{$R_{0,0}(\sigma)$ is obtained by combining $R_{0,1}$ and $R_{1,0}$\}
4:   for all $\sigma' \in R_{0,1}(\sigma)$ do
5:     for all $\sigma'' \in R_{1,0}(\sigma')$ do
6:       if $\sigma'' \neq \sigma$ then
7:         $S := S \cup \{\sigma''\}$
8:       end if
9:     end for
10:   end for
11: else
12: \{$R_{p,p}(\sigma)$ is obtained by combining $R_{p,p-1}$ and $R_{p-1,p}$\}
13: for all $\sigma' \in R_{p,p-1}(\sigma)$ do
14:   for all $\sigma'' \in R_{p-1,p}(\sigma')$ do
15:     if $\sigma'' \neq \sigma$ then
16:       $S := S \cup \{\sigma''\}$
17:   end if
18: end for
19: end if
20: return $S$
```

The time complexity of Algorithm 6.3 is dominated by the time required to retrieve either co-boundary relation $R_{0,1}$, or co-boundary relation $R_{p-1,p}$. As demonstrated in Section 6.2.3, any co-boundary relation is performed in time linear in the number of top simplices incident at any
simplex \( \sigma \). Conversely, boundary relations \( R_{1,0} \) and \( R_{p,p-1} \) are directly encoded in the IS data structure. As a consequence, the ADJACENCY query is optimal only for simplicial complexes embedded in the Euclidean space \( \mathbb{E}^3 \), otherwise it is local.

### 6.2.5 Retrieving Link of a Simplex

In this section, we describe an algorithm for retrieving link of a \( p \)-simplex \( \sigma \) in any abstract simplicial \( d \)-complex \( \Sigma \) (with \( 0 \leq p < d \)) described through the IS data structure. In other words, we provide an algorithm for the \textsc{Link} query in our implementation of the IS data structure.

Recall that link \( Lk(\sigma) \) is given by all the simplices in \( \Sigma \), which form the combinatorial boundary of simplices in \( St(\sigma) \), and are not incident at \( \sigma \). Algorithm 6.4 provides a possible solution for computing the link of a \( p \)-simplex \( \sigma \).

**Algorithm 6.4** \textsc{Link}(\( \sigma \)) - IS data structure

**Input:** a SimplexPointer reference for a \( p \)-simplex \( \sigma \) in a simplicial \( d \)-complex \( \Sigma \)

**Output:** the set \( S \) of the SimplexPointer references for all the simplices in the link of \( \sigma \)

1: let \( p := \text{dim}(\sigma) \), \( S := \emptyset \)
2: for all \( \sigma' \in \text{STAR}(\sigma) \) do
3: for all \( \sigma'' \in \text{BOUNDARY}(\sigma') \) do
4: if \( \sigma'' \) is not visited then
5: mark \( \sigma'' \) as visited
6: if \( \sigma \notin \text{BOUNDARY}(\sigma'') \) then
7: \( S := S \cup \{\sigma''\} \)
8: end if
9: end if
10: end for
11: end for
12: return \( S \)

The time complexity of Algorithm 6.4 is dominated by the retrieval of \( St(\sigma) \) through the \textsc{Star} query, thus it is \( O(\sigma^*) \), where \( \sigma^* \) is the number of top simplices incident at \( \sigma \). It is clear that \( Lk(\sigma) \) is formed by the opposite faces of \( \sigma \) for each top simplex in \( St(\sigma) \), plus their boundaries. Hence, the \textsc{Link} query is optimal only for simplicial complexes embedded in the Euclidean space \( \mathbb{E}^3 \), otherwise it is local.

### 6.2.6 Recognizing Non-Manifold Singularities

In this section, we propose an algorithm for recognizing non-manifold singularities in any abstract simplicial \( d \)-complex, with \( d \leq 3 \), described through the IS data structure. In other words, we propose an algorithm for the \textsc{IsManifold} query in our implementation of the IS data structure.

The key idea is to exploit the combinatorial characterization of non-manifold simplex, discussed
in Section 2.3. Recall that this operation is not decidable for \( d \geq 6 \) [Nab96]. Hence, we restrict our attention to arbitrary simplicial complexes of dimension up to 3 which can be embedded in the Euclidean space \( \mathbb{E}^3 \). As a consequence, there may be only several non-manifold vertices and edges, since all the 2-simplices may be either top triangles, or shared by at most two tetrahedra. Hence, all the 2-simplices are manifold.

Given a simplicial \( d \)-complex \( \Sigma \), with \( d \leq 3 \), a vertex \( v \) in \( \Sigma \) is non-manifold if its link \( Lk(v) \) is not combinatorially equivalent either to \( B^2_{\Sigma} \) or to \( S^{h-1}_{\Sigma} \), for any \( h \leq d - 1 \). Similarly, an edge \( e \) in \( \Sigma \) is non-manifold if its link \( Lk(e) \) is not combinatorially equivalent either to \( B^1_{\Sigma} \) or to \( S^{h-1}_{\Sigma} \), for any \( h \leq d - 2 \).

In other words, a vertex \( v \) in \( \Sigma \) is non-manifold if its link \( Lk(v) \) is formed by more than one connected component, or it is not formed by at most two vertices. Also, a non-manifold vertex bounds a non-manifold edge. Conversely, an edge \( e \) in \( \Sigma \) is non-manifold if its link \( Lk(e) \) is formed by more than two connected components, or it is not formed by at most two vertices.

Note that the IS data structures encodes, for each \( p \)-simplex \( \sigma \), with \( 0 \leq p < d \), partial co-boundary relation \( R_{p,p+1}(\sigma) \), which consists of a representative \((p + 1)\)-simplex for each connected component in \( Lk(\sigma) \), thus we can exploit this information for solving this problem. As a consequence, Algorithm 6.5 provides a solution for the IS_MANIFOLD query in our implementation of the IS data structure.

First, we analyze the time complexity of the IS_MANIFOLD query for an edge \( \sigma \). Here, we retrieve the number of 2-simplices in partial co-boundary relation \( R_{1,2}^* (\sigma) \), which is directly encoded in the IS data structure. If \( \| R_{1,2}^* (\sigma) \| > 2 \), or \( \| R_{1,2}^* (\sigma) \| < 2 \), then we can immediately complete our analysis, as well if partial co-boundary relation \( R_{1,2}^* (\sigma) \) contains one or two top triangles. In these cases, the time complexity is \( O(1) \). In the remaining cases, it is mandatory to check if \( Lk(\sigma) \) is formed by only two vertices. In this case, the time complexity of this operation is \( O(\sigma_1^* ) \), where \( \sigma_1^* \) is the number of top simplices in \( St(\sigma) \), as demonstrated in Section 6.2.5.

Now, we can analyze the time complexity of the IS_MANIFOLD query for a vertex \( \sigma \). Here, we retrieve the number of edges in partial co-boundary relation \( R_{0,1}^* (\sigma) \), which is directly encoded in the IS data structure. If \( \| R_{0,1}^* (\sigma) \| > 2 \), then we can immediately state the \( \sigma \) is non-manifold, thus the time complexity is \( O(1) \). Also, if \( \| R_{0,1}^* (\sigma) \| = 2 \), then it is mandatory to check if \( Lk(\sigma) \) is formed by only two vertices. In this case, the time complexity of this operation is \( O(\sigma_1^* ) \), where \( \sigma_1^* \) is the number of top simplices in \( St(\sigma) \), as demonstrated in Section 6.2.5. Finally, if \( \| R_{0,1}^* (\sigma) \| = 1 \), then it is mandatory to check if \( \sigma \) bounds any non-manifold edge \( \lambda \) in \( \Sigma \). We can retrieve co-boundary relation \( R_{0,1}(\sigma) \) in \( O(\sigma_1^* ) \), as demonstrated in Section 6.2.3. In the worst case, we can recognize a non-manifold edge \( \lambda \) in \( O(\lambda_1^* ) \), as just discussed. In any case, each top simplex in \( St(\lambda) \) is also incident at \( \sigma \), by definition. As a consequence, the time complexity of the IS_MANIFOLD query for a vertex \( \sigma \) is linear in:

\[
\sigma_1^* + \sum_{\lambda \in R_{0,1}(\sigma)} \lambda_1^* \approx \sigma_1^*
\]

In the worst case, the time complexity of the IS_MANIFOLD query, for any \( p \)-simplex \( \sigma \), is linear
Algorithm 6.5 IS.MANIFOLD(σ) - IS data structure

Input: a SimplexPointer reference for a p-simplex σ in a simplicial d-complex Σ
Output: true, if the p-simplex σ is manifold; false, otherwise.

1: let \( p := \text{dim}(σ) \), \( n = \|R_{p,p+1}(σ)\| \)
2: if \( p=0 \) then
3:   \{ Manifoldness test for a vertex \}
4:   if \( n > 2 \) then
5:     return false
6:   else if \( n=2 \) then
7:     return “LINK(σ) contains only two vertices”
8: else if \( n=1 \) then
9:   for all \( λ \in R_{0,1}(σ) \) do
10:      if not IS.MANIFOLD(λ) then
11:         return false
12:   end if
13: end for
14: return true
15: else
16: return true
17: end if
18: else if \( p=1 \) then
19: \{ Manifoldness test for an edge \}
20: if \( n > 2 \) then
21:   return false
22: else if \( n < 2 \) then
23:   return true
24: else if \( R_{1,2}^i(σ) \) contains two top triangles then
25:   return true
26: else if \( R_{1,2}^i(σ) \) contains only one top triangle then
27:   return false
28: else
29:   return “LINK(σ) contains only two vertices”
30: end if
31: else
32: return true
33: end if

in the number \( σ^t \) of top simplices in \( St(σ) \).

6.2.7 Construction Algorithm

In this section, we describe a dimension-independent algorithm for building the IS data structure from a soup of top simplices directly expressed in terms of their vertices. This representation is the most common exchange format for any simplicial d-complex Σ.

Here, the challenge is to generate all the simplices in Σ and establish topological relations. This
goal can be achieved by two auxiliary data structures, namely \( Inc \) and \( B \). Auxiliary data structure \( Inc \) is an array such that each location \( Inc[\sigma] \) corresponds to a \( p \)-simplex \( \sigma \) in \( \Sigma \), with \( 0 \leq p < d \), and contains all the \( (p+1) \)-simplices incident at \( \sigma \). Conversely, auxiliary data structure \( B \) is an array with \( d-1 \) locations, where each location \( B[i] \), with \( 0 \leq i < d-1 \), is recursively an array which contains a description of \((i+1)\)-faces of all the simplices in \( \Sigma \). Specifically, each \((i+1)\)-face is described through a raw \((i+1)\)-face. Each raw \((i+1)\)-face \( \psi \) in \( B[i] \) is generated according to the schema discussed in Section 6.1.3, and it is explicitly expressed by its vertices. A raw \((i+1)\)-face \( \psi \) is associated with the unique identifier of a \((i+2)\)-simplex \( \sigma \) bounded by \( \psi \). By definition, each location \( B[i] \) may contain duplicates of the same raw \((i+1)\)-face, which bound several raw \((i+2)\)-faces.

The construction of the IS data structure is performed by executing the following steps:

1. create \( d+1 \) \textit{SimplicesContainer} arrays, and, for each vertex \( v \) in the input soup of top simplices, generate a new \textit{Simplex} record \( r_v \) in the \textit{SimplicesContainer} array related to \( \Sigma^0 \);
2. for each top edge \( w = (v_1, v_2) \), sort all the vertices of \( w \) in increasing order, generate a new \textit{Simplex} record in the \textit{SimplicesContainer} array related to \( \Sigma^1 \), and store \( w \) in partial co-boundary relations \( R^*_{0,1}(v_1) \) and \( R^*_{0,1}(v_2) \);
3. for each top \( p \)-simplex \( \tau \), with \( 2 \leq p \leq d \), sort all the vertices of \( \tau \) in increasing order, generate all the \( \tau^p_k = \binom{p+1}{k+1} \) \( k \)-faces of \( \tau \), and store the corresponding raw faces in \( B[k-1] \), for all \( 1 \leq k < p \);
4. execute the following steps, for \( i = 0, \ldots, d-2 \):
   i) sort each array in \( B[i] \) according to the lexicographic order of the vertices in the raw \((i+1)\)-faces: as a consequence, all the unique \((i+1)\)-simplices \( \sigma \) in \( \Sigma \), shared by several \((i+2)\)-simplices \( \lambda \), are stored in consecutive locations of \( B[i] \);
   ii) for each \((i+1)\)-simplex \( \sigma \) identified at the previous step, generate a new \textit{Simplex} record \( r_{\sigma} \) in the \textit{SimplicesContainer} array related to \( \Sigma^{i+1} \), and store the \textit{SimplexPointer} reference of \( r_{\sigma} \) in place of the old identifier of \( \sigma \), assigned in Step 3;
   iii) if \( \sigma \) is an edge \((v_1, v_2)\), add \( v_1 \) and \( v_2 \) to boundary relation \( R_{1,0}(\sigma) \);
   v) store in \textit{Adj} all the top \((i+2)\)-simplices \( \tau \) (identified in Step 4i) which share a \((i+1)\)-simplex \( \sigma \): as a consequence, given a top \((i+2)\)-simplex \( \tau \), \( \textit{Adj}[\tau] \) contains all the top \((i+2)\)-simplices adjacent to \( \tau \).
5. Generate a new \textit{Simplex} record for each top \( p \)-simplex \( \tau \), with \( 2 \leq p \leq d \), and add all the \((p-1)\)-faces \( \sigma \) of \( \tau \) to boundary relation \( R_{p,p-1}(\tau) \). Add \( \tau \) to locations \( Inc[\psi] \) related to all the faces \( \psi \) bounding \( \tau \).
6. For \( 0 \leq p < d \), retrieve the partial co-boundary relation \( R^*_{p,p+1}(\sigma) \), for each \( p \)-simplex \( \sigma \).

In Step 6, the retrieval of partial co-boundary relations \( R^*_{p,p+1}(\sigma) \), for each \( p \)-simplex \( \sigma \) in \( \Sigma \), requires identifying components in \( Inc[\sigma] \) which correspond to the connected components in \( Lk(\sigma) \),
namely the partial co-boundary relation $R^*_{p,p+1}(\sigma)$. Algorithm 6.6 (IDENTIFY_COMPONENTS procedure) solves this problem.

The IDENTIFY_COMPONENTS procedure identifies all the components in $St(\sigma)$, which correspond to the connected components in $Lk(\sigma)$, in order to identify partial co-boundary relation $R^*_{p,p+1}(\sigma)$. This traversal ends when all the simplices in $Inc[\sigma]$ have been visited. During this traversal we have visited boundary of simplices bounded by simplices in $Inc[\sigma]$, thus we have visited $St(\sigma)$. Thus, the time complexity of this operation is $\|St(\sigma)\|$. 

Algorithm 6.6 IDENTIFY_COMPONENTS$(\sigma, Inc)$ - IS data structure

**Input:** a SimplexPointer reference for a $p$-simplex $\sigma$ in a simplicial $d$-complex $\Sigma$
the auxiliary data structure $Inc$ storing the co-boundary relations $R_{k,k+1}$, for $0 \leq k < d$

**Output:** the partial co-boundary relation $R^*_{p,p+1}(\sigma)$

1: let $p := \text{dim}(\sigma)$
2: for all $\sigma' \in Inc[\sigma]$ do
3: if $\sigma'$ is not visited then
4: let $q$ be an empty queue
5: $R^*_{p,p+1} = R^*_{p,p+1} \cup \{\sigma'\}$
6: enqueue $\sigma'$ in $q$
7: while $q$ is not empty do
8: dequeue $\sigma''$ from $q$
9: if $\sigma''$ is not visited then
10: mark $\sigma''$ as visited
11: for all $\lambda \in Inc[\sigma'']$ do
12: if $\sigma \in \text{BOUNDARY}(\lambda)$ then
13: enqueue $\lambda$ in $q$
14: end if
15: end for
16: for all $\lambda \in \text{BOUNDARY}(\sigma'')$ do
17: if $\sigma \in \text{BOUNDARY}(\lambda)$ then
18: enqueue $\lambda$ in $q$
19: end if
20: end for
21: end if
22: end while
23: end if
24: end for

Now, we evaluate the time complexity of the algorithm used for building the IS data structure from a soup of top simplices. We assume to generate new Simplex records, raw faces, and SimplicesContainer array in $O(1)$.

In Step 1, we generate a new Simplex record for each vertex, and $d+1$ SimplicesContainer arrays, thus, the time complexity of this step is $O(s_0 + d + 1)$. In Step 2, we generate a new Simplex record for each top edge, and store its vertices, thus, the time complexity of this step is $O(s_1^*).$
Step 3, given a top \( p \)-simplex \( \tau \), with \( 1 < p \leq d \), we sort all the vertices of \( \tau \) in increasing order, and the time complexity of this operation is \( \mathcal{O}((p + 1) \log(p + 1)) \). We also generate, for \( 1 \leq k < p \), all the \( \tau^k \) faces of \( \tau \), and store them in \( \mathcal{B}[k - 1] \). Thus, the time complexity of this step is linear in:

\[
\sum_{p=2}^{d} s^t_p \left( (p + 1) \log(p + 1) + \sum_{k=1}^{p-1} \left( \frac{p + 1}{k + 1} \right) \right) \approx \sum_{p=2}^{d} s^t_p
\]

since \( (p + 1) \log(p + 1) \) and the total number of faces of a top \( p \)-simplex \( \tau \) can be considered as constant values depending only on \( p \). At the end of this step, each location \( \mathcal{B}[i] \), with \( 0 \leq i \leq d - 2 \), contains \( b_i \) faces of dimension \( i + 1 \), where:

\[
b_i = \sum_{p=2}^{d} \left( \frac{p + 1}{i + 2} \right) s^t_p \approx \sum_{p=2}^{d} s^t_p
\]

since \( \left( \frac{p + 1}{i + 2} \right) \) is a constant value depending only on \( p \) and \( i \). In Step 4, we sort each location \( \mathcal{B}[i] \), and the time complexity of this operation is \( b_i \log(b_i) \). Also, we create \( (s_{i+1} - s^t_{i+1}) \) new Simplex records for the unique and not top \((i + 1)\)-simplices in \( \mathcal{B}[i] \), and, then, we adjust their boundary relations. Hence, the time complexity of this step is linear in:

\[
\sum_{i=0}^{d-2} b_i \log(b_i) + \sum_{i=0}^{d-2} s_{i+1} - s^t_{i+1} \approx \sum_{i=0}^{d-2} b_i \log(b_i)
\]

since the definition of \( b_i \). Note that the time complexity of the IDENTIFY COMPONENTS procedure is \( \mathcal{O}(\|St(\sigma)\|) \). As a consequence, the time complexity of our algorithm is linear in:

\[
s_0 + s^t_1 + N^t \log(N^t) + \sum_{\sigma \in \Sigma} \|St(\sigma)\|
\]

where \( N^t = \sum_{p=2}^{d} s^t_p \) is the total number of top \( p \)-simplices in \( \Sigma \), with \( p > 2 \).

### 6.3 Implementing the Generalized Indexed data structure with Adjacencies

In this section, we describe a complete implementation of the Generalized Indexed data structure with Adjacencies (IA*) [CDFW11], introduced in Section 5.1, in the context of our Mangrove TDS framework. Recall that the IA* representation is an explicit, adjacency-based, and dimension-independent data structure, which encodes all the vertices and top simplices in an abstract simplicial complex, plus a subset of boundary and adjacency relations for each top simplex, and a subset of co-boundary relations for each vertex. As a consequence, the IA* data structure is represented through a local mangrove.

In Section 6.3.1, we provide a complete description of the IA* data structure in terms of internal data structures offered by the Mangrove TDS framework. In Sections 6.3.2 and 6.3.3, we provide algorithms for executing the BOUNDARY and STAR queries, respectively. In Sections 6.3.4
and 6.3.5, we define algorithms for performing, respectively, the ADJACENCY and LINK queries. In Section 6.3.6, we provide an algorithm for executing IS_MANIFOLD query. Finally, in Section 6.3.7, we provide an algorithm for building the IA* data structure from a soup of top simplices directly expressed in terms of their vertices.

6.3.1 Implementation of the Data Structure

In this section, we propose a complete description of the IA* data structure in terms of internal data structures offered by the Mangrove TDS framework.

In order to encode the IA* data structure, we need $d+1$ SimplicesContainer arrays, one for each subset of the collection $\Sigma^p$ of $p$-simplices in $\Sigma$ (with $0 \leq p \leq d$), which we store in the IA* data structure. Specifically, we encode all the vertices in $\Sigma$ and top $p$-simplices in $\Sigma$ (with $1 \leq p \leq d$).

We also encode any information about non-manifold $k$-simplices, with $1 \leq k < d$, which are incident at least one top $(k+1)$-simplex.

Each vertex $v$ in $\Sigma$ is encoded through a Simplex record $r_v$ in the SimplicesContainer array related to collection $\Sigma^0$. Each record $r_v$ is assigned to a unique SimplexPointer reference. In the cob array of $r_v$, we store several SimplexPointer references, one for each representative $p$-simplex of a $p$-cluster incident at $\sigma$, namely in partial co-boundary relation $R^{*}_{0,p}(v)$, with $0 < p \leq d$.

Each non-manifold $(p-1)$-simplex $\tau$, shared by at least one top $p$-simplex, with $2 \leq p \leq d$, is encoded through a Simplex record $r_\tau$ in the SimplicesContainer array related to a subset of collection $\Sigma^{p-1}$. Each record $r_\tau$ is assigned to a unique SimplexPointer reference. In the cob array of $r_\tau$, we store several SimplexPointer references, one for each top $p$-simplex in partial co-boundary relation $R^{*}_{p-1,p}(\tau)$.

Each top $p$-simplex $\sigma$ in $\Sigma$, with $1 \leq p \leq d$, is encoded through a Simplex record $r_\sigma$ in the SimplicesContainer array related to a subset of collection $\Sigma^p$. Each record $r_\sigma$ is assigned to a unique SimplexPointer reference. In the bnd array of $r_\sigma$, we store $p+1$ SimplexPointer references, one for each vertex in boundary relation $R^{*}_{p,0}(\sigma)$. If $p \geq 2$, then we also encode partial adjacency relation $R^{*}_{p,p}(\sigma)$ in the array adj of $r_\sigma$, where each location corresponds to a $(p-1)$-face $\tau$ of $\sigma$. If $\tau$ is manifold, then we encode the SimplexPointer reference for the top $p$-simplex $\sigma'$ adjacent to $\sigma$ along $\tau$, if it exists. If $\tau$ is non-manifold, then we encode a SimplexPointer reference to record $r_\tau$, which contains partial co-boundary relation $R^{*}_{p-1,0}(\tau)$.

6.3.2 Retrieving Boundary of a Simplex

In this section, we propose the a dimension-independent algorithm for retrieving all the simplices bounding a $p$-simplex $\sigma$ belonging to any abstract simplicial $d$-complex $\Sigma$ (with $0 < p \leq d$) described by the IA* data structure. In other words, we provide an algorithm for the BOUNDARY query in our implementation of the IA* data structure.
In Section 5.1.3.1, we have demonstrated that a boundary relation of a simplex $\sigma$ can be retrieved by a traversal of the IA* boundary graph of $\sigma$, formed by $p + 1$ boundary arcs outgoing from the node describing $\sigma$. Specifically, we can retrieve the combinatorial boundary of a top $p$-simplex $\sigma$ by expanding all the boundary arcs $(\sigma, v)$, which connect nodes related to $\sigma$ and to each vertex $v$ in $R^*_{p,0}(\sigma)$. Conversely, we must recursively generate all the faces of a non-top simplex $\sigma$.

Note that we describe combinatorial boundary of a $p$-simplex in terms of GhostSimplexPointer references. We can exploit the hierarchies of ghost simplices we have introduced in Section 6.1.3. Algorithm 6.7 proposes a possible solution for this problem.

**Algorithm 6.7** BOUNDARY($\sigma$) - IA* data structure

**Input:** a GhostSimplexPointer reference for a $p$-simplex $\sigma$ in a simplicial $d$-complex $\Sigma$

**Output:** the set $S$ of the GhostSimplexPointer references for all the simplices bounding $\sigma$

1: let $p :=$ FATHER_TYPE($\sigma$), $S := \emptyset$
2: if $\sigma$ is a top simplex in $\Sigma$ then
3:   {We generate the GhostSimplexPointer references for all the faces of the top simplex $\sigma$}
4:   for all $0 \leq k < p$ do
5:     for all $0 \leq i < \binom{p+1}{k+1}$ do
6:       $S := S \cup \{(p, \text{FATHER_ID}(\sigma), k, i)\}$
7:   end for
8: end for
9: else
10:   {We recursively generate the GhostSimplexPointer references through the FACES function for all the faces of the $p$-simplex $\sigma$}
11: let $s$ be an empty stack
12: push $\sigma$ in $s$
13: while $s$ is not empty do
14:   pop $\sigma'$ from $s$
15:   if CHILD_TYPE($\sigma'$) $\neq 0$ then
16:     for all $j$ in FACES($\sigma'$) do
17:       push $(p, \text{FATHER_ID}(\sigma), \text{CHILD_TYPE}(\sigma'), j)$ in $s$
18:       $S := S \cup \{(p, \text{FATHER_ID}(\sigma), \text{CHILD_TYPE}(\sigma'), j)\}$
19:     end for
20:   end if
21: end while
22: end if
23: return $S$

If $\sigma$ is a top simplex, then we simply generate GhostSimplexPointer references for all the $k$-faces of $\sigma$, with $0 \leq k < d$. By definition, a $p$-simplex $\sigma$ has $\sigma^p_k = \binom{p+1}{k+1}$ faces of dimension $k$ [Ede87]. Note that, in this case, a GhostSimplexPointer reference for $\sigma$ is the same as the SimplexPointer reference of $\sigma$ (see Section 6.1.3), which we can express as $(p, \text{FATHER_ID}(\sigma))$. Hence, we can exploit $\sigma$ as reference simplex for its subfaces, and generate all the GhostSimplexPointer references $(p, \text{FATHER_ID}(\sigma), k, i)$, with $0 \leq k < d$, and $0 \leq i \leq \sigma^p_k$. Note that, in this case, we do not perform an explicit traversal of the IA* boundary graph.
Conversely, if $\sigma$ is not a top simplex, then it is surely a $p$-face of a top $m$-simplex $\tau$, with $m > p$, by definition. Thus, we must generate $GhostSimplexPointer$ references for all the $k$-faces of $\tau$, with $0 \leq k < p$, which bound $\sigma$. Unique identifiers of these faces can be recursively retrieved by the FACES function. In other words, we visit our hierarchy of faces, as discussed in Section 6.1.3. Note that this hierarchy of faces has been already computed as a pre-processing step.

As a consequence, we visit all the $b_\sigma$ simplices which bound $\sigma$ (see Section 6.2.2), and each operation can be performed $O(1)$. Thus, the time complexity of the BOUNDARY query is $O(1)$, since $b_\sigma$ is a constant value which depends only on $p$.

### 6.3.3 Retrieving Star of a Simplex

In this section, we describe a dimension-independent algorithm for retrieving all the simplices incident at a $p$-simplex $\sigma$ belonging to any abstract simplicial $d$-complex $\Sigma$ (with $0 \leq p < d$) described by the IA* data structure. In other words, we provide an algorithm for the STAR query in our implementation of the IA* data structure. In Section 5.1.3.2, we have illustrated basic ideas for retrieving star of a $p$-simplex in $\Sigma$.

First, we retrieve all the top $k$-simplices, with $1 \leq k \leq d$, incident at any vertex $v$ in $\Sigma$. Here, the key idea consists of retrieving all the top edges in $R^*_{0,1}(v)$, and to expand each $k$-cluster, with $k \neq 1$, encoded by its representative $k$-simplex in $R^*_{0,k}(v)$. Each $k$-cluster can be expanded through the transitive closure of partial adjacency relation $R^*_{0,k}$, restricted to top $k$-simplices. Top $k$-simplices, which are adjacent along a non-manifold $(k-1)$-simplex $\tau$, are retrieved through partial co-boundary relation $R^*_{p-1,0}(\tau)$. Algorithm 6.8 (CLUSTER function) retrieves all the top $k$-simplices incident at any vertex $v$ in $\Sigma$, with $0 < k \leq d$. All the topological relations involved in this algorithm are directly encoded in the IA* data structure, thus the time complexity of the CLUSTER function is $O(v^k_*)$, where $v^k_*$ is the number of all the top $k$-simplices in $St(v)$.

Note that all the top simplices retrieved by the CLUSTER function are top simplices directly encoded in the IA* data structure, and they can be accessed by a SimplexPointer reference.

The star of a vertex $\sigma$ is formed by all the top $k$-simplices, with $1 \leq k \leq d$, and by their faces, which are incident at $\sigma$. Thus, we can retrieve all the top $k$-simplices in $St(\sigma)$, and then select their faces, which are incident at $\sigma$, as performed in Algorithm 6.9. The time complexity for the first step of this operation is $O(\sigma^t_1)$, where $\sigma^t_1$ is the number of all the top simplices incident at $\sigma$. Note that we must describe these simplices through a list of $GhostSimplexPointer$ references.

For each top $k$-simplex $\sigma'$ in $St(v)$, with $0 < k \leq d$, we retrieve all the simplices $\sigma''$ bounding $\sigma'$ by the BOUNDARY query, plus the position $i_\sigma$ of $\sigma$ in the ordered list of vertices bounding $\sigma'$, namely $R^*_{k,0}(\sigma')$. We select all the simplices $\sigma''$ bounded by vertex $\sigma$. Note that all the positions of vertices bounding a simplex $\sigma''$ are retrieved by the LOOKUP function, discussed in Section 6.1.3. The time complexity of these operations is $O(1)$, for each top simplex in $St(\sigma)$. Hence, the time complexity for retrieving the star of a vertex $\sigma$ is $O(\sigma^t_1)$. As a consequence, the STAR query is optimal only for simplicial complexes embedded in $E^3$. Conversely, it is local for
Algorithm 6.8 CLUSTER(v, k) - IA* data structure

Input: a SimplexPointer reference for a vertex v in a simplicial d-complex Σ
the dimension k of the required top simplices incident at v

Output: the set S of the SimplexPointer references for all the top p-simplices incident at v

1: let $S := \emptyset$
2: if $k > 1$ then
3:   {We expand all the k-clusters in $R_{0,k}^v(v)$}
4:   for all $\sigma' \in R_{0,k}^v(v)$ do
5:     let q be an empty queue
6:     enqueue $\sigma'$ in q
7:     while q is not empty do
8:       dequeue $\sigma$ from q
9:       if $\sigma$ is not visited then
10:          mark $\sigma$ as visited
11:          $S := S \cup \{\sigma\}$
12:          for all $\tau$ in $R_{k,k-1}(\sigma)$ do
13:            if $R_{k-1,k}(\tau) = \emptyset$ then
14:              {The $(k-1)$-simplex $\tau$ is manifold}
15:              enqueue $R_{k,k}(\sigma)$ along $\tau$ in q
16:            else
17:              {The $(k-1)$-simplex $\tau$ is non-manifold, and we exploit $R_{k-1,k}^v(\tau)$}
18:              for all $\sigma''$ in $R_{k-1,k}^v(\tau)$ do
19:                enqueue $\sigma''$ in q
20:          end for
21:       end if
22:     end for
23:   end if
24: end while
25: end for
26: else
27:   {We retrieve all the top edges in $R_{0,1}^v(v)$}
28:   for all $\sigma' \in R_{0,1}^v(v)$ do
29:     $S := S \cup \{\sigma'\}$
30:   end for
31: end if
32: return $S$

simplicial h-complexes, with $h \geq 4$.

If $\sigma$ is neither a vertex, or a top simplex, then its star is formed by top simplices, and faces of these simplices, which are incident at all the vertices of $\sigma$. Thus, given a vertex $v$ of $\sigma$, the star of $\sigma$ can be retrieved by identifying all the top simplices incident at $v$, and by selecting simplices, which are incident at all the vertices bounding $\sigma$, as performed in Algorithm 6.9.

At the first step, we retrieve the ordered list $v_\sigma = [v_0, \ldots, v_p]$ of vertices bounding $\sigma$, by combining the GHOST and LOOKUP functions (see Section 6.1.3) on $\sigma$, which is referred by a GhostSimplexPointer reference. The time complexity of this operation is $O(1)$. For the sake of simplicity,
Algorithm 6.9 STAR(σ) - IA* data structure

**Input:** a GhostSimplexPointer reference for a p-simplex σ in a simplicial d-complex Σ

**Output:** the set S of the GhostSimplexPointer references for all the simplices incident at σ

1: let $S := \emptyset$, $p := \text{CHILD\_TYPE}(\sigma)$
2: if $p = 0$ then
3:   {We retrieve all the top simplices and their faces incident at a vertex σ}
4:   for all $0 < k \leq d$ do
5:     for all $\sigma'$ in CLUSTER(σ, k) do
6:       $S := S \cup \{\sigma'\}$
7:     end for
8:     let $i_\sigma$ be the position of $\sigma$ in $R^*_{k,0}(\sigma')$
9:     for all $\sigma''$ in BOUNDARY($\sigma'$) do
10:        if $i_\sigma \in \text{LOOKUP}(\sigma'')$ then
11:           $S := S \cup \{\sigma''\}$
12:        end if
13:     end for
14:   end for
15: else
16:   {We retrieve the star of a ghost p-simplex σ}
17:   let $v_\sigma = [v_0, \ldots, v_p] = \text{GHOST}(\text{LOOKUP}(\sigma))$
18:   for all $0 < k \leq d$ do
19:     for all $\sigma'$ in CLUSTER($v_\sigma$, k) do
20:       if $\sigma'$ is incident at all vertices in $v_\sigma$ then
21:         $S := S \cup \{\sigma'\}$
22:       for all $\sigma''$ in BOUNDARY($\sigma'$) do
23:         if $\sigma''$ is incident at all vertices in $v_\sigma$ then
24:           $S := S \cup \{\sigma''\}$
25:         end if
26:       end for
27:     end for
28:   end for
29:   end if
30: return $S$

we retrieve all the top $k$-simplices, with $0 < k \leq p$, which are incident at $v = v_0$, through the CLUSTER function. The time complexity of this operation is $O(v^*_t)$, where $v^*_t$ is the number of top simplices incident at $v$. We select all the top simplices $\sigma'$ and their faces $\sigma''$, which are incident all the vertices in $v_\sigma$. Partial boundary relation $R^*_{k,0}(\sigma')$ is directly encoded in the IA* data structure, and the BOUNDARY query can be retrieved in $O(1)$, as demonstrated in Section 6.3.2.

As a consequence, the time complexity of these operations is $O(1)$, for each top simplex $\sigma'$ in $St(v)$. Hence, the time complexity for retrieving the star of a p-simplex $\sigma$, with $0 < p < d$, is $O(v^*_t)$, where $v^*_t$ is the number of top simplices incident at one vertex (arbitrarily selected), which bounds $\sigma$. Note that $v^*_t$ is surely larger than the number $\sigma^*_t$ of top simplices in $St(\sigma)$.  

132
6.3.4 Retrieving Simplices Adjacent to a Simplex

In this section, we describe a dimension-independent algorithm for retrieving all the $p$-simplices adjacent to a $p$-simplex $\sigma$ in any abstract simplicial $d$-complex $\Sigma$, namely the adjacency relation $R_{p,p}(\sigma)$, with $0 \leq p \leq d$. In other words, we provide an algorithm for the ADJACENCY query in our implementation of the IA* data structure. In Section 5.1.3.3, we have illustrated the basic ideas for retrieving all the simplices adjacent to a $p$-simplex in any simplicial $d$-complex.

The IA* data structure directly encodes adjacency relation $R_{d,d}(\sigma)$, for any $d$-simplex in $\Sigma$. It also encodes, for $1 < k < d$, partial adjacency relation $R_{k,k}^*(\sigma')$, restricted to top $k$-simplices adjacent to a top $k$-simplex $\sigma'$. As discussed in Section 5.1.1, top $k$-simplices, which are adjacent along a non-manifold $(k-1)$-simplex $\tau$, are retrieved by partial co-boundary relation $R_{k-1,k}^*(\tau)$. Algorithm 6.10 (ADJACENT function) retrieves top $k$-simplices adjacent to a top $k$-simplex $\sigma$ in $\Sigma$. Here, $\sigma$ and the resulting top $p$-simplices are referred by a SimplexPointer reference, since they are directly encoded in the IA* data structure. Topological relations involved in the ADJACENT function are encoded in the IA* data structure, hence the time complexity of this algorithm is $O(\sigma_k^a)$, where $\sigma_k^a$ is the number of top $k$-simplices adjacent to $\sigma$.

Algorithm 6.10 ADJACENT($\sigma$) - IA* data structure

Input: a SimplexPointer reference for a top $k$-simplex $\sigma$ in a simplicial $d$-complex $\Sigma$

Output: the set $S$ of the SimplexPointer references for all the top $k$-simplices adjacent to $\sigma$

1: let $S := \emptyset$
2: for all $\tau$ in $R_{k,k-1}(\sigma)$ do
3:   if $R_{k-1,k}^*(\tau) = \emptyset$ then
4:      {The $(k-1)$-simplex $\tau$ is a manifold $(k-1)$-face of $\sigma$}
5:      let $\theta$ be $R_{k,k}^*(\sigma)$ along $\tau$
6:      if $\theta$ is not visited then
7:         mark $\theta$ as visited
8:         $S := S \cup \{\theta\}$
9:      end if
10:   else
11:      {The $(k-1)$-simplex $\tau$ is a non-manifold $(k-1)$-face of $\sigma$, and we exploit $R_{k-1,k}^*(\tau)$}
12:      for all $\sigma''$ in $R_{k-1,k}^*(\tau)$ do
13:         if $\sigma''$ is not visited and $\sigma'' \neq \sigma$ then
14:            $S := S \cup \{\sigma''\}$
15:            mark $\sigma''$ as visited
16:         end if
17:      end for
18:   end if
19: end for
20: return $S$

Adjacency relation $R_{p,p}(\sigma)$ of any top $p$-simplex, with $1 \leq p \leq d$, can be retrieved in two steps, as proposed in Algorithm 6.11 (ADJACENCY query). In the first step, we retrieve partial adjacency relation $R_{p,p}^*(\sigma)$ by Algorithm 6.10 (ADJACENT function). The time complexity of this operation
Algorithm 6.11 ADIACENCY(σ) - IA* data structure

Input: a GhostSimplexPointer reference for a $p$-simplex $σ$ in a simplicial $d$-complex $Σ$

Output: the set $S$ of the GhostSimplexPointer references for all the $p$-simplices adjacent to $σ$

1: let $S := ∅$, $p := \text{CHILD\_TYPE}(σ)$
2: if $σ$ is top then
3:   {We retrieve the adjacency relation $R_{p,p}(σ)$ for a top $p$-simplex $σ$}
4:   $S := \text{ADJACENT}(σ)$
5: if $p \neq d$ then
6:   for all $τ$ in $R_{p,p−1}(σ)$ do
7:     for all $σ'$ in $R_{p−1,p}(τ)$ do
8:       if $σ' \neq σ$ and $σ'$ is not visited then
9:         $S := S ∪ \{σ'\}$
10:        mark $σ'$ as visited
11:       end if
12:     end for
13:   end for
14: else if $p=0$ then
15:   {We retrieve the adjacency relation $R_{0,0}(σ)$ for a vertex $σ$}
16:   for all $0 < k ≤ d$ do
17:     for all $σ' ∈ \text{CLUSTER}(σ,k)$ do
18:       let $i_σ$ be the position of $σ$ in $R_{k,0}(σ')$
19:       for all $0 ≤ j ≤ k$ do
20:         let $η = (k,\text{CHILD\_ID}(σ'), 0, j)$
21:         if $j \neq i_σ$ and $η$ is not visited then
22:           mark $η$ as visited
23:           $S := S ∪ \{η\}$
24:         end if
25:       end for
26:     end for
27:   end for
28: else
29:   {We retrieve the adjacency relation $R_{p,p}(σ)$ for any ghost $p$-simplex $σ$}
30:   for all $τ$ in $R_{p,p−1}(σ)$ do
31:     for all $σ'$ in $R_{p−1,p}(τ)$ do
32:       if $σ' \neq σ$ and $σ'$ is not visited then
33:         $S := S ∪ \{σ'\}$
34:        mark $σ'$ as visited
35:       end if
36:     end for
37:   end for
38: else
39:   end if
40: return $S$

is $O(σ_a^k)$, as discussed above. In the second step, we retrieve non-top $p$-simplices adjacent to $σ$, by combining relations $R_{p,p−1}(σ)$, and $R_{p−1,p}(τ)$, for each $(p−1)$-face $τ$ bounding $σ$. As demonstrated in Section 6.3.3, partial co-boundary relation $R_{p−1,p}(τ)$ is retrieved in $O(v^l)$, where
$v^*_i$ is the number of top simplices incident at a vertex $v$ bounding $\tau$. Note that $v$ bounds also $\sigma$. In this case, if we consider each vertex $v$ in $R^*_{p,0}(\sigma)$ for computing co-boundary relations $R_{p-1,p}$, then the time complexity of the ADJACENCY query is linear in:

$$\sigma^k + \sum_{v \in R^*_{p,0}(\sigma)} v^*_i$$

Adjacency relation $R_{0,0}(\sigma)$ of a vertex $\sigma$ in $\Sigma$ can be retrieved in two steps. In the first step, we retrieve all the top simplices incident at $\sigma$ by the CLUSTER function, discussed in Section 6.3.3. The time complexity of this step is $O(\sigma^k)$, where $\sigma^k$ is the number of top simplices incident at $\sigma$. In the second step, for each top $k$-simplex $\sigma'$ in $St(\sigma)$, with $0 < k \leq d$, we select vertices of $\sigma'$, which are different from $\sigma$. If $i_\sigma$ is the position of $\sigma$ in $R^*_{k,0}(\sigma')$, then we can consider vertices bounding $\sigma'$, stored in position different than $i_\sigma$, and generate their related GhostSimplexPointer references. The time complexity of these operations is $O(1)$, for each top simplex $\sigma'$. In this case, the time complexity of the ADJACENCY query is $O(\sigma^k)$, where $\sigma^k$ is the number of top simplices in $St(\sigma)$.

If $\sigma$ is neither a vertex, or a top $p$-simplex, then adjacency relation $R_{p,p}(\sigma)$ is retrieved by combining relations $R_{p,p-1}(\sigma)$, and $R_{p-1,p}(\tau)$, for each $(p-1)$-face $\tau$ bounding $\sigma$, as performed for retrieving non-top $p$-simplices adjacent to a top $p$-simplex. In this case, if we consider a vertex $v$ in $R^*_{p,0}(\sigma)$ for computing co-boundary relations $R_{p-1,p}$, then the time complexity of the ADJACENCY query is linear in:

$$\sum_{v \in R^*_{p,0}(\sigma)} v^*_i$$

where $v^*_i$ is the number of top simplices in $St(v)$.

6.3.5 Retrieving Link of a Simplex

In this section, we provide an algorithm for retrieving link of a simplex $\sigma$ in any abstract simplicial $d$-complex $\Sigma$ described through the IA* data structure. In other words, we provide an algorithm for the LINK query in our implementation of the IA* data structure. Recall that link $Lk(\sigma)$ is given by all the simplices in $\Sigma$, which form the combinatorial boundary of simplices in $St(\sigma)$, and are not incident at $\sigma$.

As discussed in Section 4.3, each cluster in the star of a $p$-simplex $\sigma$, with $0 \leq p < d$, is bounded by several simplices belonging to $Lk(\sigma)$. Specifically, each top $k$-simplex $\sigma'$ in $St(\sigma)$, with $p < k \leq d$, corresponds to a $(k-p-1)$-face $\theta$ of $\sigma'$, which is the opposite face of $\sigma$ in $\sigma'$. The $(k-p-1)$-face $\theta$ is formed by vertices of $\sigma'$, which do not belong to $\sigma$. As a consequence, $\theta$ and simplices in its boundary belong to the link of $\sigma$.

Thus, we can exploit this idea for retrieving the link of a $p$-simplex $\sigma$ in $\Sigma$ in two steps. In the first step, we retrieve top simplices incident at $\sigma$. In the second step, for each top $k$-simplex $\sigma'$ in $St(\sigma)$, with $k > p$, we retrieve the opposite $(k-p-1)$-face $\theta$ of the $p$-simplex $\sigma$ with respect to $\sigma'$, and all the simplices bounding $\theta$. Algorithm 6.12 provides a possible solution for this problem.
Algorithm 6.12 LINK(σ) - IA* data structure

**Input:** a GhostSimplexPointer reference for a p-simplex σ in a simplicial d-complex Σ

**Output:** the set S of the GhostSimplexPointer references belonging to the link of σ

1: let S := ∅, p := CHILD_TYPE(σ)
2: if p = 0 then
3:   {We retrieve the link of a vertex σ}
4:   for all 0 < k ≤ d do
5:     for all σ′ in CLUSTER(σ, k) do
6:       let iσ be the position of σ in R∗ k,0(σ′), θ = (k, FATHER_TYPE(σ′), k − 1, iσ)
7:       S := S ∪ {θ}
8:       for all η ∈ BOUNDARY(θ) do
9:         if η is not visited then
10:            mark η as visited
11:            S := S ∪ {η}
12:       end if
13:     end for
14:   end for
15: else
16:   {We retrieve the link of a p-simplex σ, with p ≠ 0}
17:   let vσ = [v0, ..., vp] = GHOST(LOOKUP(σ))
18:   for all 0 < k ≤ d do
19:     for all σ′ in CLUSTER(vσ, k) do
20:       if σ′ is incident at all vertices in vσ then
21:         let iσ be the position of σ in the p-faces of σ′
22:         if p = k − 1 then
23:           S := S ∪ {(k, CHILD_ID(σ′), 0, iσ)}
24:         else
25:           let θ = (k, CHILD_ID(σ′), k − p − 1, (k+1) − 1 − iσ)
26:           S := S ∪ {θ}
27:           for all η ∈ BOUNDARY(θ) do
28:             if η is not visited then
29:               mark η as visited
30:               S := S ∪ {η}
31:             end if
32:           end if
33:         end if
34:       end if
35:     end for
36:   end for
37:   end if
38: return S

Given a vertex σ, we retrieve top k-simplices in St(σ), for 0 < k ≤ d, by the CLUSTER function, discussed in Section 6.3.3. The time complexity of this operation is O(σ∗ t), where σ∗ t is the number of top simplices incident at σ. For each top k-simplex σ′ in St(σ), we identify the position iσ of vertex σ in partial boundary relation R∗ k,0(σ′). Thus, the opposite (k − 1)-face θ of vertex σ in σ′
is the \((k-1)\)-face of \(\sigma'\) in position \(i_{\sigma}\), due to the schema exploited for constructing faces, described in Section 6.1.3. Then, we generate \textit{GhostSimplexPointer} references for the \((k-1)\)-face \(\theta\), and all the simplices bounding \(\theta\). The time complexity of these operations is \(O(1)\), for each top simplex \(\sigma'\). In this case, the time complexity of the \textit{LINK} query is \(O(\sigma'_i)\), where \(\sigma'_i\) is the number of top simplices incident at \(\sigma\).

If \(\sigma\) is not a vertex, then we retrieve, for \(p < k \leq d\), top \(k\)-simplices incident at any vertex \(v\) of \(\sigma\) by the \textit{CLUSTER} function (see Section 6.3.3), and, then, we select top \(k\)-simplices incident at all the vertices of \(\sigma\). The time complexity of this operation is \(O(v'_i)\), where \(v'_i\) is the number of top simplices incident at a vertex \(v\) of \(\sigma\). For each top \(k\)-simplex \(\sigma'\) in \(St(\sigma)\), we identify the opposite \((k-p-1)\)-face \(\theta\) of \(\sigma\). First, we identify the position \(i_{\sigma}\) of \(\sigma\) among \(p\)-faces of \(\sigma'\). If \(\sigma\) is a \((k-1)\)-face of \(\sigma'\), then we consider the vertex stored in position \(i_{\sigma}\) in partial boundary relation \(R^p_{k,0}(\sigma)\). Otherwise, the opposite \((k-p-1)\)-face \(\theta\) for \(\sigma\) in \(\sigma'\) is the \((k-p-1)\)-face of \(\sigma'\) in position \((k+1) - i_{\sigma}\) among the \((k-p-1)\)-faces of \(\sigma'\). This result is a direct consequence of the algorithm used for generating faces, which we have introduced in Section 6.1.3. Then, we generate \textit{GhostSimplexPointer} references for the \((k-p-1)\)-face \(\theta\), and all the simplices bounding \(\theta\). The time complexity of these operations is \(O(1)\), for each top simplex \(\sigma'\). In this case, the time complexity of the \textit{LINK} query is dominated by the retrieval of all the top simplices incident at any vertex \(v\) bounding \(\sigma\), thus, the time complexity is \(O(v'_i)\).

### 6.3.6 Recognizing Non-Manifold Singularities

In this section, we propose an algorithm for recognizing non-manifold singularities in any abstract simplicial \(d\)-complex, with \(d \leq 3\), described through the IA* data structure. In other words, we propose an algorithm for the IS\_MANIFOLD query in our implementation of the IA* data structure.

Given a simplicial \(d\)-complex \(\Sigma\), with \(d \leq 3\), we can characterize non-manifold vertices and edges as discussed in Section 6.2.6. In any case, the IA* data structure encodes, for each vertex \(v\), the representative top \(k\)-simplices for \(k\)-clusters in \(St(v)\), with \(0 < k \leq d\). Note that a vertex \(v\) is surely non-manifold if its star is formed by several clusters of different dimensions. The reverse is not true: for instance, in the arbitrary simplicial 3-complex in Figure 5.1, the star of the vertex 3 is formed by only one 2-cluster, but the vertex 3 is non-manifold, since it bounds the non-manifold edge (1,3). We can exploit partial co-boundary relation \(R^1_{1,2}(\tau)\), which characterizes a non-manifold edge \(\tau\) shared by at least one top triangle, as discussed in Section 5.1.1. It is clear that, in a simplicial 2-complex, a non-manifold edge \(\tau\) is completely characterized by partial co-boundary relation \(R^1_{1,2}(\tau)\). Conversely, in a simplicial 3-complex, a non-manifold edge \(\tau\) is not always characterized by partial co-boundary relation \(R^1_{1,2}(\tau)\). If there are not any top triangles in \(St(\tau)\), then we cannot exploit partial co-boundary relation relation \(R^1_{1,2}(\tau)\), because it is empty. For instance, the star of the non-manifold edge \(e\) in Figure 5.2(b) contains only two tetrahedra.

In this case, we analyze the number of clusters incident at the two vertices bounding edge \(\tau = (w,z)\). If only one cluster is incident at both the vertices \(w\) and \(z\), then this cluster is incident
at $\tau$, which is manifold. If there is not only one common cluster incident at $w$ and $z$, then it is mandatory to check if more than one cluster is incident at the edge $\tau$, starting from the top simplices incident at $w$ and $z$. Algorithm 6.13 (EDGE_ANALYSIS function) provides a possible solution for checking these situations.

**Algorithm 6.13** EDGE_ANALYSIS($\tau$) - IA$^*$ data structure

**Input:** a GhostSimplexPointer reference for an edge $\tau$ in a simplicial $d$-complex $\Sigma$

**Output:** true, if the edge $\tau$ is manifold; false, otherwise.

1: let $L := \emptyset$, $(w,z) = \text{GHOST(LOOKUP(\tau))}$, $n := 0$
2: for all $2 < k \leq d$ do
3:   for all $\sigma' \in \text{CLUSTER}(w,k)$ do
4:     if $z \in R_{k,0}^*(\sigma')$ then
5:       $L := L \cup \{\sigma'\}$
6:     end if
7:   end for
8: end for
9: for all $\sigma'$ in $L$ do
10:   if $\sigma'$ is not visited then
11:     mark $\sigma'$ as visited
12:     $n := n + 1$
13:   if $n > 1$ then
14:     return false
15: else
16:     let $q$ an empty queue
17:     enqueue $\sigma'$ in $q$
18:     while $q$ is not empty do
19:       dequeue $\lambda$ from $q$
20:       if $\lambda$ is not visited then
21:         mark $\lambda$ as visited
22:         for all $\eta$ in ADJACENT($\lambda$) do
23:           enqueue $\eta$ in $q$
24:         end for
25:       end if
26:     end while
27: end if
28: end if
29: end for
30: return true

In the first step of Algorithm 6.13, we retrieve top $k$-simplices in $St(w)$, with $2 < k \leq d$ by the CLUSTER function, described in Section 6.3.3. Then, we select top $k$-simplices (with $2 < k \leq d$), which are incident also at $z$. The time complexity of this step is $O(w^*_k)$, where $w^*_k$ is the number of top simplices incident at $w$. In the second step, we start to identify one $k$-cluster in $St(\tau)$, for any $2 < k \leq d$. Given a top simplex $\sigma'$, a $k$-cluster represented by $\sigma'$ is the transitive closure of the $(k-1)$-adjacency relation, starting from $\sigma'$. Note that partial adjacency relation $R_{k,k}^*$ is computed through the ADJACENT function, introduced in Section 6.3.4. If we identify more
than one cluster, then edge $\tau$ is non-manifold, otherwise it is manifold. At each step of this traversal, we visit top simplices adjacent to a top simplex in $St(\tau)$, thus the time complexity of these operations is $O(\tau^*_t)$, where $\tau^*_t$ is the number of top simplices incident at $\tau$. As a consequence, the time complexity of Algorithm 6.13 is $O(w^*_t)$, since $w^*_t$ is usually greater than $\tau^*_t$.

Algorithm 6.14 summarizes a possible algorithm for the IS_MANIFOLD query in our implementation of the IA* data structure, which exploits the EDGE_ANALYSIS function.

**Algorithm 6.14 IS_MANIFOLD($\sigma$) - IA* data structure**

**Input:** a GhostSimplexPointer reference for a $p$-simplex $\sigma$ in a simplicial $d$-complex $\Sigma$

**Output:** true, if the $p$-simplex $\sigma$ is manifold; false, otherwise.

1: let $p := \text{CHILD\_TYPE}(\sigma)$
2: if $p = 0$ then
3:   {Manifoldness test for a vertex}
4:   let $n = \sum \|R^*_0,q(\sigma)\|$, for $1 \leq q \leq d$
5:   if $n > 2$ then
6:     return false
7:   else if $n = 2$ then
8:     return “$St(\sigma)$ contains only two top edges”
9:   else
10:      for all $\lambda \in R^*_{0,1}(\sigma)$ do
11:       if not IS_MANIFOLD($\lambda$) then
12:          return false
13:       end if
14:      end for
15:      return true
16:   end if
17: else if $p = 1$ then
18:   {Manifoldness test for an edge}
19:   if $R^*_{1,2}(\sigma) \neq \emptyset$ then
20:      return false
21:   else if $d > 2$ then
22:      let $(w, z) = \text{GHOST}(\text{LOOKUP}(\tau))$
23:      if $\|R^*_{0,k}(w)\| = 1$ and $\|R^*_{0,k}(z)\| = 1$ for any $k$ then
24:          return true
25:      else
26:          return EDGE_ANALYSIS($\sigma$)
27:      end if
28:   else
29:      return true
30:   end if
31: else
32:   return true
33: end if

Note that, in a simplicial 2-complex $\Sigma$, a non-manifold edge $\sigma$ is non-manifold if and only if $R^*_{1,2}(\sigma)$ is not empty. In this case, the time complexity of the IS_MANIFOLD query is $O(1)$. Conversely,
in any simplicial 3-complex, an edge \( \sigma \) is surely non-manifold if \( R_{1,2}(\sigma) \) is not empty, but this is not a sufficient condition for being a non-manifold edge, as discussed above. If \( R_{1,2}(\sigma) \) is empty, then we execute the EDGE\_ANALYSIS function. Thus, in the worst case, the time complexity of the IS\_MANIFOLD query for an edge \( \sigma \) is \( O(\sigma^t) \), where \( \sigma^t \) is the number of top simplices in \( St(\sigma) \).

A vertex \( \sigma \) is surely non-manifold if its star is formed by more than two clusters. Also, if the star of \( \sigma \) is formed by only two top edges, then \( \sigma \) is manifold. Conversely, if the star of \( \sigma \) is formed by two clusters different than two top edges, then \( \sigma \) is non-manifold. These tests can be performed in \( O(1) \), because partial co-boundary relations are directly encoded in the IA* data structure for a vertex. If the star of the vertex \( \sigma \) is formed by only one cluster, then it is mandatory to check if \( \sigma \) bounds any non-manifold edge \( \lambda \) in \( \Sigma \). We can retrieve co-boundary relation \( R_{0,1}(\sigma) \) in \( O(\sigma^t) \), as demonstrated in Section 6.3.3. In the worst case, we recognize a non-manifold edge \( \lambda \) in \( O(\lambda^t) \), where \( \lambda^t \) is the number of top simplices incident at \( \lambda \). In any case, each top simplex in \( St(\lambda) \) is also incident at \( \sigma \), by definition. As a consequence, the time complexity of the IS\_MANIFOLD query for any vertex \( \sigma \) is linear in:

\[
\sigma^t + \sum_{\lambda \in R_{0,1}(\lambda)} \lambda^t \approx \sigma^t
\]

In the worst case, the time complexity of the IS\_MANIFOLD query, for any \( p \)-simplex \( \sigma \), is linear in the number \( \sigma^t \) of top simplices in \( St(\sigma) \).

### 6.3.7 Construction Algorithm

In this section, we describe a dimension-independent algorithm for building the IA* data structure from a soup of top simplices directly expressed in terms of their vertices, which provides a compact description of any simplicial \( d \)-complex \( \Sigma \).

As a consequence, we need to store vertices and top simplices of \( \Sigma \) in the IA* data structure, and establish topological relations among them. This goal can be achieved by three auxiliary data structures, namely \( Inc \), \( Adj \), and \( B \). Auxiliary data structure \( Inc \) is an array such that each location corresponds to a vertex \( v \) in \( \Sigma \), and contains top simplices incident at \( v \). Auxiliary data structure \( Adj \) is an array such that each location corresponds to a top \( p \)-simplex \( \sigma \), with \( 1 < p \leq d \), and contains top \( p \)-simplices adjacent to \( \sigma \). Finally, auxiliary data structure \( B \) is an array with \( d - 1 \) locations, where each location \( B[i] \), with \( 0 \leq i < d - 1 \), is recursively an array which contains a description of \( (i + 1) \)-faces bounding top \( (i + 2) \)-simplices in \( \Sigma \). Specifically, each \( (i + 1) \)-face is described by a raw \( (i + 1) \)-face. Each raw \( (i + 1) \)-face \( \psi \) in \( B[i] \) is generated according to the schema discussed in Section 6.1.3, and it is explicitly expressed by its vertices. A raw \( (i + 1) \)-face \( \psi \) is associated with the unique identifier of a \( (i + 2) \)-simplex \( \sigma \) bounded by \( \psi \).

The construction of the IA* data structure is performed by executing the following steps:

1. create \( d + 1 \) \textit{SimplicesContainer} arrays, and, for each vertex \( v \) in the input soup of top
simplices, generate a new Simplex record \( r_v \) in the SimplicesContainer array related to \( \Sigma^0 \);

2. for each top edge \( w = (v_1, v_2) \), sort all the vertices of \( w \) in increasing order, generate a new Simplex record in the SimplicesContainer array related to \( \Sigma^1 \), and store \( w \) in partial co-boundary relations \( R^*_{0,1}(v_1) \) and \( R^*_{0,1}(v_2) \);

3. execute the following steps, for each top \( p \)-simplex \( \sigma \), with \( 2 \leq p \leq d \):
   i) generate a new Simplex record \( r_\sigma \) in the SimplicesContainer array related to a subset of the collection \( \Sigma^p \);
   ii) store vertices of \( \sigma \), sorted in increasing order, in partial boundary relation \( R^*_{p,0}(\sigma) \), and store \( \sigma \) in Inc[\( v \)], for any vertex \( v \) bounding \( \sigma \);
   iii) generate all the \( \sigma_{p-1} \) \( (p-1) \)-faces of \( \sigma \), and store the corresponding raw \( (p-1) \)-faces in \( B[p-1] \).

4. Sort each array \( B[i] \), with \( 0 \leq i < d-1 \), according to the lexicographic order of vertices in the raw \( (i+1) \)-faces; as a consequence, all the unique \( (i+1) \)-simplices \( \tau \) shared by top \( (i+2) \)-simplices \( \sigma \) are stored in consecutive locations of \( B[i] \);

5. store in the auxiliary data structure \( Adj \) all the top \( (i+2) \)-simplices, which share any \( (i+1) \)-simplex \( \tau \) (identified in Step 4): as a consequence, given a top \( (i+2) \)-simplex \( \sigma \), location \( Adj[\sigma] \) contains all the top simplices adjacent to \( \sigma \);

6. for each vertex \( v \), identify all the \( k \)-clusters incident at \( v \), and, thus, the partial co-boundary relations \( R^*_{0,k}(v) \), for \( 0 < k \leq d \), by considering all the top simplices in the auxiliary data structures Inc and \( Adj \), as described in Algorithm 6.15 (IDENTIFY_CLUSTERS procedure).

7. For each \( (i+1) \)-simplex \( \tau \), with \( 0 \leq i < d-1 \), shared by several top \( (i+2) \)-simplices identified in Step 5, perform the following steps:
   i) if \( \tau \) is shared by more than two top \( (i+2) \)-simplices, then we encode partial adjacency relation \( R^*_{i+2,i+2} \) along \( \tau \) by partial co-boundary relation \( R^*_{i+1,i+2}(\tau) \). We generate a new Simplex record \( r_\tau \) in the SimplicesContainer array related to a subset of \( \Sigma^{i+1} \), and we store SimplexPointer references for all the top \( (i+2) \)-simplices in \( St(\tau) \) in the array \( cob \) in \( r_\tau \). Finally, for each top \( (i+2) \)-simplex \( \sigma \) in \( St(\tau) \), we encode SimplexPointer reference of \( r_\sigma \) in the location of the \( adj \) array in \( r_\sigma \) corresponding to \( \tau \).
   ii) If the star of \( \tau \) contains two top \( (i+2) \)-simplices \( \sigma \) and \( \sigma' \), encoded, respectively, in the Simplex records \( r_\sigma \) and \( r_{\sigma'} \), then we must check if any top \( h \)-simplex \( \psi \) is incident at \( \tau \), with \( h > i + 2 \). In other words, we check if there exists a top \( h \)-simplex \( \psi \) belonging to \( Inc[\psi] \), for any vertex \( v \) of \( \tau \). If such a simplex \( \psi \) exists, then the \( (i+1) \)-simplex \( \tau \) is non-manifold, and we proceed as the previous case. Otherwise, \( \sigma \) and \( \sigma' \) are adjacent along the manifold \( (i+1) \)-simplex \( \tau \). Hence, we store the SimplexPointer reference of \( \sigma' \) in the location of the array \( adj \) related to \( \tau \) in \( r_\sigma \), and vice versa.
• If the star of \( \tau \) contains only one top \((i+2)\)-simplex \( \sigma \), encoded in the Simplex record \( r_\sigma \), then we check if any top \( h \)-simplex \( \psi \) is incident at \( \tau \), with \( h > i + 2 \). If such a simplex \( \psi \) exists, then \( \tau \) is non-manifold. Hence, we generate a new Simplex record \( r_\tau \) in the SimplicesContainer array related to a subset of \( \Sigma^{i+1} \), then we store the SimplexPointer reference of \( \sigma \) in the array \( \text{cob} \) of \( r_\tau \). Finally, we store the SimplexPointer reference of \( r_\tau \) in the location of the adj array in \( r_\sigma \) corresponding to \( \tau \).

Algorithm 6.15 IDENTIFY_CLUSTERS(v,Inc,Adj) - IA* data structure

Input: a SimplexPointer reference for a vertex \( v \) in a simplicial \( d \)-complex \( \Sigma \)
the auxiliary data structure \( \text{Inc} \) storing all the top simplices incident at any vertex \( v \) in \( \Sigma \)
the auxiliary data structure \( \text{Adj} \) storing the adjacency relation for top simplices in \( \Sigma \)

Output: the partial co-boundary relation \( R^*_0(v) \), with \( 1 < k \leq d \)

1: for all \( \sigma' \in \text{Inc}[v] \) do
2:     if \( \sigma' \) is not visited then
3:         let \( k := \dim(\sigma') \), \( q \) be an empty queue
4:         \{ Retrieval of a new \( k \)-cluster starts from \( \sigma' \) \}
5:         \( R^*_{0,k}(v) = R^*_{0,k}(v) \cup \{ \sigma' \} \)
6:         enqueue \( \sigma' \) in \( q \)
7:     while \( q \) is not empty do
8:         dequeue \( \sigma'' \) from \( q \)
9:         if \( \sigma'' \) is not visited then
10:            mark \( \sigma'' \) as visited
11:               for all \( \lambda \in \text{Adj}[\sigma''] \) do
12:                   enqueue \( \lambda \) in \( q \)
13:               end for
14:         end if
15:     end while
16: end if
17: end for

Now, we can evaluate the time complexity of the algorithm used for building the IA* data structure from a soup of top simplices. We assume to generate new Simplex records, raw faces, and SimplicesContainer object in \( O(1) \).

In Step 1, we generate a new Simplex record for each vertex, and \( d + 1 \) SimplicesContainer arrays, thus the time complexity of this step is \( O(s_0 + d + 1) \). In Step 2, we generate a new Simplex record for each top edge, and store its vertices, thus the time complexity of this step is \( O(s_1^1) \). In Step 3, given a top \( p \)-simplex \( \sigma \), with \( 1 < p \leq d \), we sort all the vertices of \( \sigma \) in increasing order, and store them in \( R^*_{0,0}(\sigma) \). Also, we generate all the \( \binom{p+1}{p} \) raw \((p-1)\)-faces of \( \sigma \). Thus, the time complexity of this step is linear in:

\[
\sum_{p=2}^{d} s_p^1 \left( (p + 1) \log(p + 1) + 2(p + 1) + \binom{p + 1}{p} \right) \approx \sum_{p=2}^{d} s_p^1
\]

since \( (p + 1) \log(p + 1) \), \( p + 1 \) and \( \binom{p+1}{p} \) are constant values depending only on \( p \). At the end of this step, each location \( B[i] \), with \( 0 \leq i < d - 1 \), contains \( b_i = \binom{i+3}{i+2} s_{i+2}^2 \) faces of dimension.
In other words, \( b_i \approx s_{i+2}^t \), with \( 0 \leq i < d - 1 \). In Step 4, we sort each location \( B[i] \), with \( 0 \leq i < d - 1 \), and the time complexity of this operation is \( b_i \log(b_i) \approx s_{i+2}^t \log(s_{i+2}^t) \). In Step 5, we store the \( \sigma_{a}^{i+2} \) top \((i+2)\)-simplices adjacent for each top \((i+2)\)-simplex \( \sigma \) in the auxiliary data structure \( Adj \), and, thus, the time complexity is \( \mathcal{O}(\sigma_{a}^{i+2}) \). In Step 6, we retrieve all the \( k \)-clusters in the star of any vertex \( v \) in \( \Sigma \), with \( 1 \leq k \leq d \), starting from top simplices in \( St(v) \), stored in \( Inc[v] \). Given a top \( k \)-simplex \( \sigma' \), a \( k \)-cluster represented by \( \sigma' \) is the transitive closure of the \((k-1)\)-adjacency relation, starting from \( \sigma' \). Note that partial adjacency relation \( \mathcal{R}_{k,k}^{s} \) is stored in the \( Adj \) auxiliary data structure. Thus, the time complexity of this step is \( \mathcal{O}(v_{i}^t) \), where \( v_{i}^t \) is the number of top simplices incident at \( v \). Finally, in Step 7, we finalize adjacency relation \( \mathcal{R}_{i+1,i+2}^{s} \) for top \((i+2)\)-simplices, by encoding partial co-boundary relation \( \mathcal{R}_{i+1,i+2}^{s} \), with \( 0 \leq i < d - 1 \). The time complexity is dominated by storing all the \( \mathcal{N}_{i+2}^{s} \) top \((i+2)\)-simplices incident at any non-manifold \((i+1)\)-simplex \( \tau \), thus the time complexity of this step is linear in:

\[
\sum_{i=0}^{d-2} \sum_{\tau \in \Sigma^{i+1}} \mathcal{N}_{i+2}^{s}
\]

As a consequence, the time complexity of our algorithm is linear in:

\[
s_0 + s_1^t + \sum_{p=2}^{d} s_p^t \log(s_p^t) + \sum_{v \in \Sigma^o} v_{i}^t + \sum_{p=2}^{d} \left( \sum_{\sigma \in \Sigma^p} \sigma_{a}^{p} + \sum_{\tau \in \Sigma^{p-1}} \mathcal{N}_{p}^{\tau} \right)
\]
Chapter 7

Analysis and Comparisons of Topological Data Structures

The Mangrove Topological Data Structure (Mangrove TDS) framework, which we have introduced in Chapter 6, supports a wide number of topological data structures under a common application interface. In our framework, any topological data structure can be described through a graph-based representation, which we call a mangrove.

The content of this chapter is twofold. On one side, we present implementations of all the topological data structures analyzed in Chapters 4 and 5. In Section 7.1, we propose an implementation of the Triangle Segment (TS) data structure [DFMPS04]. In Section 7.2, we present an implementation of the Non-Manifold Indexed data structure with Adjacencies (NMIA) [DFH03]. These implementations of the TS and NMIA data structures are equivalent to their new definitions, introduced in Sections 5.2 and 5.3, respectively. In Section 7.3, we propose an implementation of the Simplified Incidence Graph (SIG) [DFGH04], discussed in Section 4.3. Finally, in Section 7.4, we analyze a restriction of the Incidence Graph (IG) [Ede87] to simplicial complexes, discussed in Section 4.2. Here, we complete these topological data structures, and provide their navigation and construction algorithms.

On the other side, we exploit the common platform provided by Mangrove TDS framework in order to perform quantitative comparisons among running times of topological queries and building algorithms for all the topological data structures mentioned above, plus the Incidence Simplicial (IS) [DFHPC10] data structure, and Generalized Indexed Data Structures with Adjacencies (IA*) [CDFW11], whose implementations are described in Chapter 6. We present our experimental results in Section 7.5.

Our tests highlight several properties for each data structure. Specifically, our tests show that the IS and IA* data structures offer an optimal compromise regarding their expressive power, storage cost, and efficiency of all the queries. Our tests also show that the GhostSimplexPointer references improve expressive power of local mangroves, and provide an effective representation of simplices.
not directly encoded. In any case, this implicit representation can be applied only to simplicial complexes, since boundary relations are constant.

7.1 Implementing the Triangle Segment data structure

In this section, we propose a possible implementation of the Triangle Segment (TS) data structure [DFMPS04]. Specifically, we discuss a new proposal of the TS data structure, which we have introduced in Section 5.2.

Recall that the TS representation is an explicit and adjacency-based data structure, specific for simplicial 2-complexes embedded in the Euclidean space $E^3$. It encodes all the vertices and top simplices, plus a subset of boundary and adjacency relations for each top simplex, and of co-boundary relations for each vertex. As a consequence, the TS data structure is represented through a local mangrove. The TS data structure exploits the radial sorting of triangles around any non-manifold edge in order to reduce the amount of encoded information. Hence, it is mandatory to provide a geometric embedding in the Euclidean space $E^3$ for the simplicial 2-complex $\Sigma$ to be represented. This embedding is completely defined by associating a Euclidean point with each vertex in $\Sigma$.

It is clear that the TS data structure can be considered as a slight modification of the IA$^\star$($2D$) data structure, introduced in Section 5.4.1. The main difference is given by the different encoding of non-manifold adjacency of three or more triangles along an edge.

Our starting point is the implementation of the IA$^\star$($2D$) data structure, namely the restriction to simplicial 2-complexes of the dimension-independent implementation of the IA$^\star$ data structure introduced in Section 6.3. Thus, we can describe the TS data structure in terms of modifications to our implementation of the IA$^\star$($2D$) representation in order to efficiently encode non-manifold adjacency in the TS data structure. Specifically, in Section 7.1.1, we propose a description of the TS data structure in terms of the internal data structures offered by the Mangrove TDS framework. Then, in Section 7.1.2, we briefly explain how algorithms of the IA$^\star$($2D$) data structure may be modified and reused in the TS data structure. Finally, in Section 7.1.3, we describe a slight modification of the construction algorithm of the IA$^\star$($2D$) data structure, described in Section 6.3.7, in order to build the TS data structure from a soup of top simplices directly expressed in terms of their vertices.

7.1.1 Implementation of the Data Structure

In this section, we describe the implementation of the TS data structure in terms of the internal data structures offered by the Mangrove TDS framework. Specifically, we discuss a new proposal of the TS data structure, which we have introduced in Section 5.2.

As demonstrated in Section 5.4.1, the main difference between the TS and IA$^\star$($2D$) data structures
is given by the different encoding of non-manifold adjacency along edges. Remaining content of the TS data structure coincides with the content of the IA∗(2D) representation, described in Section 6.3.1.

Given a simplicial 2-complex Σ embedded in the Euclidean space $\mathbb{E}^3$, the TS data structure encodes partial co-boundary relation $R_{1,2}^*(\tau)$ for each non-manifold edge $\tau$ shared by more than two triangles. Partial co-boundary relation $R_{1,2}^*(\tau)$ is encoded for each triangle $t$ in $St(\tau)$, and consists of two triangles, which immediately precede and follow $t$ in counter-clockwise order around $\tau$. The TS data structure encodes adjacency relation $R_{2,2}$ for each triangle. As discussed in Section 5.2, for a triangle $t$, adjacency relation $R_{2,2}(t)$ along an edge $\tau$ of $t$ is encoded as either one triangle adjacent to $t$, if $\tau$ is manifold, or as a reference to two triangles in $R_{1,2}^*(\tau)$, if $\tau$ is non-manifold.

Each non-manifold edge $\tau$ bounding a triangle $t$ is encoded through a Simplex record $r_\tau$ in the SimplicesContainer array related to the collection $\Sigma^1$ of 1-simplices in $\Sigma$. Each record $r_\tau$ is assigned to a unique SimplexPointer reference. In the cob array in $r_\tau$, we encode two SimplexPointer references to the two triangles, which immediately precede and follow $t$ in counter-clockwise order around $\tau$. Note that we encode a different record $r_\tau$ for each triangle $t$ incident at $\tau$.

Array adj in the Simplex record $r_t$ related to a triangle $t$ describes triangles adjacent along the three edges of $t$. If any edge $\tau$ of $t$ is non-manifold, then location adj[\tau] contains the SimplexPointer reference of edge $\tau$ related to partial co-boundary relation $R_{1,2}^*(\tau)$ with respect to $t$. Otherwise, location adj[\tau] contains the SimplexPointer reference of a triangle adjacent to $t$ along $\tau$, if it exists.

Finally, we encode Euclidean coordinates of each vertex through a local property associated with each Simplex record related to a vertex in $\Sigma^0$.

### 7.1.2 Retrieving Non-Manifold Adjacencies

In this section, we briefly explain how we can modify and reuse algorithms of the IA∗(2D) data structure for implementing topological queries in the TS data structure. As demonstrated in Section 5.4.1, the main difference between the TS and the IA∗(2D) data structures is given by the different encoding of non-manifold adjacency along edges.

Note that the TS and IA∗(2D) data structures encode the same boundary relations, thus we can reuse BOUNDARY query of the IA∗(2D) data structure, introduced in Section 6.3.2.

Conversely, the TS data structure encodes partial relation $R_{1,2}^*(\tau)$ for each triangle $t$ incident at a non-manifold edge $\tau$. Recall that it consists of two triangles $t_1$ and $t_2$, which immediately precede and follow $t$ in counter-clockwise order around $\tau$. Hence, we can retrieve all the triangles incident at any non-manifold edge $\tau$ by traversing all the partial relations $R_{1,2}^*(\tau)$ sorted in counter-clockwise or clockwise order around $\tau$. As a consequence, we have to modify the CLUSTER and ADJACENT functions, introduced, respectively, in Sections 6.3.3 and 6.3.4, for retrieving all the triangles adjacent to a given triangle.
In the ADJACENT function, presented in Algorithm 7.1, we retrieve all the triangles adjacent to a triangle \( \sigma \). Triangles adjacent to \( \sigma \) through manifold edges of \( \sigma \) are directly encoded through adjacency relation \( R_{2,2} \). Otherwise, we have to traverse the complete sequence of partial relations \( R_{1,2}^* \) for all the triangles incident at a non-manifold edge \( \tau \) of \( \sigma \). In our proposal, triangles adjacent to \( \sigma \) are sorted in counter-clockwise order around a non-manifold edge \( \tau \) of \( \sigma \). In any case, the time complexity of the ADJACENT function remains \( O(\sigma_{ta}) \), where \( \sigma_{ta} \) is the number of triangles adjacent to \( \sigma \).

Algorithm 7.1 ADJACENT(\( \sigma \)) - TS data structure

**Input:** a SimplexPointer reference to a triangle \( \sigma \) in a simplicial 2-complex \( \Sigma \)

**Output:** the set \( S \) of triangles adjacent to \( \sigma \)

1: let \( S := \emptyset \)
2: for all \( \tau \) in \( R_{2,1}(\sigma) \) do
3:  if \( R_{1,2}^*(\tau) = \emptyset \) then
4:    \{Edge \( \tau \) is a manifold edge of \( \sigma \}\}
5:    let \( \theta \) be \( R_{2,2}(\sigma) \) along \( \tau \)
6:    \( S := S \cup \{\theta\} \)
7:  else
8:    \{Edge \( \tau \) is a non-manifold edge of \( \sigma \), and we exploit \( R_{1,2}^*(\tau) \)\}
9:    let \( \{t_1, t_2\} = R_{1,2}(\tau) \), \( \theta = t_2 \)
10:   while \( \theta \neq \sigma \) do
11:     \( S := S \cup \{\theta\} \)
12:    let \( \{t_1, t_2\} = R_{1,2}^*(\tau) \), \( \theta = t_2 \)
13:   end while
14: end if
15: end for
16: return \( S \)

In the CLUSTER function, presented in Algorithm 7.2, we retrieve all the top \( p \)-simplices incident at any vertex \( v \), namely top edges and triangles in \( St(v) \). Note that top edges incident at \( v \) are given by partial co-boundary relation \( R_{0,1}^*(v) \). We can retrieve all the triangles in \( St(v) \) by expanding the content of each 2-cluster in partial co-boundary relation \( R_{0,2}^*(v) \) through the ADJACENT function. In any case, the time complexity of the CLUSTER function remains \( O(v_t^p) \), where \( v_t^p \) is the number of top \( p \)-simplices incident at \( v \).

Hence, we can reuse implementations of the STAR, ADJACENCY, and LINK queries in the IA*(2D) data structure, introduced in Sections 6.3.3, 6.3.4, and 6.3.5, respectively. Thus, time complexities of these queries do not change.

Finally, we can simplify the IS_MANIFOLD query in the IA*(2D) data structure, proposed in Section 6.3.6. The TS data structure represents only simplicial 2-complexes, thus, the star of a vertex is formed only by edges and triangles, while the star of an edge is formed only by triangles. A non-manifold edge \( \tau \) is completely characterized by partial co-boundary relation \( R_{1,2}^*(\tau) \), and it can be recognized in \( O(1) \). A non-manifold vertex \( v \) can be characterized in the same way as the IA*(2D) data structure, and it can be recognized in \( O(v_t^e) \), where \( v_t^e \) is the number of top
Algorithm 7.2 CLUSTER(v, p) - TS data structure

**Input:** a SimplexPointer reference to a vertex v in a simplicial 2-complex $\Sigma$
the dimension $p$ of the required top $p$-simplices incident at $v$

**Output:** the set $S$ of top $p$-simplices incident at $v$

```plaintext
1: let $S := \emptyset$
2: if $p = 2$ then
3:   {We expand each 2-cluster in $R^*_{0,2}(v)$}
4:   for all $\sigma' \in R^*_{0,2}(v)$ do
5:     let $q$ be an empty queue
6:     enqueue $\sigma'$ in $q$
7:     while $q$ is not empty do
8:       dequeue $\sigma$ from $q$
9:       if $\sigma$ is not visited then
10:           mark $\sigma$ as visited
11:           $S := S \cup \{\sigma\}$
12:           for all $\sigma'' \in$ ADJACENT($\sigma$) do
13:             if $\sigma''$ is not visited then
14:               enqueue $\sigma''$ in $q$
15:           end if
16:       end if
17:   end while
18: end if
19: else
20:   {We retrieve all the top edges in $R^*_{0,1}(v)$}
21:   for all $\sigma' \in R^*_{0,1}(v)$ do
22:     $S := S \cup \{\sigma'\}$
23:   end for
24: end if
25: return $S$
```

simplices in $St(v)$, since a non-manifold edge is recognized in $O(1)$.

### 7.1.3 Construction Algorithm

In this section, we propose a slight modification of the construction algorithm of the IA$^*(2D)$ data structure, described in Section 6.3.7. Here, the challenge is to build the TS data structure, which describes a simplicial 2-complex $\Sigma$ described by a soup of top simplices directly expressed in terms of their vertices. Recall that we also need Euclidean coordinates associated with all the vertices in the input soup of top simplices.

We can completely reuse the first six steps in the construction algorithm of the IA$^*(2D)$ data structure, by also reusing auxiliary data structures $Inc$, $Adj$, and $B$. Each location of the auxiliary data structure $Inc$ corresponds to a vertex $v$, and contains all the triangles in $St(v)$. Each location of the auxiliary data structure $Adj$ corresponds to a triangle $t$, and contains all the triangles
adjacent to $t$. Finally, auxiliary data structure $B$ is formed only by only one location $B[0]$, which contains raw 1-faces shared by triangles.

In any case, in Step 1, we must also generate a local property related to all the vertices for storing their Euclidean coordinates, and store a Euclidean point for each vertex. The time complexity of this step is $O(s_0)$, where $s_0$ is the number of vertices in the input soup of top simplices. Also, the IDENTIFY_CLUSTERS procedure retrieves partial co-boundary relation $R_{0,2}(v)$, for any vertex $v$. The time complexity of these six steps is linear in:

$$s_0 + s_1^t + s_2^t \log(s_2) + \sum_{v \in \Sigma} \|v_1^v\| + \sum_{t \in \Sigma^2} t_2^t$$

where $s_0$ and $s_1$ are the numbers of vertices and edges in $\Sigma$, $s_1^t$ is the number of top edges in $\Sigma$, $v_1^v$ is the number of triangles incident at any vertex $v$ in $\Sigma$, and $t_2^t$ is the number of triangles adjacent to any triangle $t$ in $\Sigma$.

In Step 7, we must consider the different encoding of non-manifold adjacency for three or more triangles in the TS data structure. Specifically, we need to encode partial co-boundary relation $R_{1,2}^*(\tau)$ for any non-manifold edge $\tau$ in $\Sigma$. Recall that any non-manifold edge $\tau$ in a simplicial 2-complex $\Sigma$ is shared by more than two triangles. Given a non-manifold edge $\tau$ shared by several triangles, we perform the following steps:

i) If the star of $\tau$ contains three or more triangles, then $\tau$ is surely a non-manifold edge. Hence, we can identify partial co-boundary relation $R_{1,2}^*(\tau)$ for each triangle in $St(\tau)$ in two steps. In the first step, we radially sort all the $N_2^\tau$ triangles incident at $\tau = (v_1, v_2)$ in counter-clockwise order around $\tau$ by using the Euclidean coordinates of each vertex. This operation is equivalent to radially sort all the vertices of these triangles different than $v_1$ and $v_2$ around $\tau$ [PS85]. The sorted sequence of triangles is stored in a circular double-linked list. The time complexity of this operation is $O(N_2^\tau \log N_2^\tau)$. In the second step, we perform the following operations, for each triangle $t$ in $St(\tau)$:

- generate a new Simplex record $r_\tau$, corresponding to edge $\tau$, in the SimplicesContainer array related to a subset of the collection $\Sigma^1$. Then, store the SimplexPointer references for the two triangles, which precede and follow $t$ in counter-clockwise order around $\tau$, in the array cob of $r_\tau$.
- Store the SimplexPointer reference of $t$ in the location of adj array in the Simplex record $r_\tau$ (related to the triangle $t$), which corresponds to edge $\tau$.

ii) If the star of $\tau$ contains only two triangles $t$ and $t'$, respectively encoded in the Simplex records $r_t$ and $r_{t'}$, then $t$ and $t'$ are adjacent along $\tau$. Hence, we store the SimplexPointer reference of $t'$ in the location of array adj related to $\tau$ in $r_t$, and vice versa.

iii) If the star of $\tau$ contains only one triangle $t$, then the $\tau$ is a manifold edge of $t$, and we do not encode any other information.
The time complexity of this step is dominated by generating all the Simplex records for each non-manifold edge. Recall that any triangle \( t \) has \( N_t^2 \) non-manifold edges, as discussed in Section 5.2.

As a consequence, the time complexity of this algorithm is linear in:

\[
s_0 + s_1^t + s_2 \log(s_2) + \sum_{v \in \Sigma^0} v^2 + \sum_{t \in \Sigma^2} (t_2^2 + N_t^2)
\]

### 7.2 Implementing the Non-Manifold Indexed data structure with Adjacencies

In this section, we propose a possible implementation of the *Non-Manifold Indexed data structure with Adjacencies (NMIA)* data structure [DFH03]. Specifically, we discuss the new proposal of the NMIA data structure, which we have introduced in Section 5.3.

Recall that the NMIA representation is an explicit and adjacency-based data structure, specific for arbitrary simplicial 3-complexes embedded in the Euclidean space \( \mathbb{E}^3 \). It encodes all the vertices and top simplices, plus a subset of boundary and adjacency relations for each top simplex, and of co-boundary relations for each vertex. As a consequence, the NMIA data structure is represented through a *local mangrove*. The NMIA exploits the radial sorting of triangles and tetrahedra around any non-manifold edge in order to reduce the amount of encoded information. Hence, it is mandatory to provide a geometric embedding in the Euclidean space \( \mathbb{E}^3 \) for the simplicial 3-complex \( \Sigma \) to be represented. This embedding is completely defined by associating a Euclidean point with each vertex in \( \Sigma \).

In Section 7.2.1, we propose a description of the NMIA data structure in terms of the internal data structures offered by the Mangrove TDS framework. The NMIA data structure encodes the same boundary relations as the IA* data structure, thus, boundary of any simplex can be retrieved by using the BOUNDARY query in Section 6.3.2, restricted to simplicial 3-complexes. In [DFH03, Hui08] the authors introduce algorithms for retrieving co-boundary and adjacency relations for any simplex, which are straightforward to implement in the Mangrove TDS framework. As discussed in Section 5.3, all the vertex-based and edge-based co-boundary and adjacency relations are local, while co-boundary relation \( R_{2,3} \) and adjacency relation \( R_{2,2} \) are optimal. In Sections 7.2.2 and 7.2.3, we describe algorithms for executing the LINK and IS_MANIFOLD queries, respectively. Finally, in Section 7.2.4, we propose an algorithm for building the NMIA data structure from a soup of top simplices directly expressed in terms of their vertices.

#### 7.2.1 Implementation of the Data Structure

In this section, we describe the implementation of the NMIA data structure in terms of the internal data structures offered by the Mangrove TDS framework. Specifically, we discuss a new proposal of the NMIA data structure, which we have introduced in Section 5.3.
In order to encode the NMIA data structure, we need four *SimplicesContainer* arrays, related to subsets of the collections $\Sigma^p$ of $p$-simplices in $\Sigma$, with $0 \leq p \leq 3$. Specifically, the *SimplicesContainer* array related to $\Sigma^1$ encodes all the top edges in $\Sigma^1_t$, plus several *Simplex* records related to non-manifold edges in $\Sigma$. Remaining *SimplicesContainer* arrays encode all the vertices in $\Sigma^0$ and top $p$-simplices in $\Sigma^p_t$, with $p \neq 1$.

Each vertex $v$ in $\Sigma$ is encoded through a *Simplex* record $r_v$ in the *SimplicesContainer* array related to $\Sigma^0$. Each record $r_v$ is assigned to a unique *SimplexPointer* reference. In the `cob` array of $r_v$, we store several *SimplexPointer* references to representative simplices of top edges and vertex-based clusters in $\text{St}(v)$.

Each non-manifold edge $\tau$, which bounds a top simplex $\sigma$ in any cluster $C_\sigma$, is encoded through a *Simplex* record $r_\tau$ in the *SimplicesContainer* array related to $\Sigma^1_t$. Each record $r_\tau$ is assigned to a unique *SimplexPointer* reference. In the `cob` array of $r_\tau$, we encode two *SimplexPointer* references to representative simplices of the clusters which immediately precede and follow $C_\sigma$ in counter-clockwise order around $\tau$. Note that we encode a *Simplex* record $r_\tau$ for each top simplex $\sigma$ incident at $\tau$.

Each top $p$-simplex $\sigma$, with $1 \leq p \leq d$, is encoded through a *Simplex* record $r_\sigma$ in the *SimplicesContainer* array related to $\Sigma^p_t$. Each record $r_\sigma$ is assigned to a unique *SimplexPointer* reference. In the `bnd` array of $r_\sigma$, we store $p + 1$ *SimplexPointer* references, one for each vertex in partial boundary relation $R^*_{p,0}(\sigma)$.

In the `adj` array of any *Simplex* record $r_\sigma$ related to a top triangle $\sigma$, we encode relation $R_{2,cl}(\tau)$ along three edges $\tau$ of $\sigma$. If an edge $\tau$ of $\sigma$ is manifold, then we encode the *SimplexPointer* reference of the top triangle $\gamma$ adjacent to $\sigma$ along $\tau$. Otherwise, we store the *SimplexPointer* reference of the record $r_\tau$ which describes partial co-boundary relation $R^*_{1,cl}(\tau)$ with respect to $\sigma$.

In the `adj` array of the *Simplex* record $r_\sigma$ related to a tetrahedron $\sigma$, we encode *SimplexPointer* references of tetrahedra adjacent to $\sigma$ along its four triangles, namely the adjacency relation $R_{3,3}(\sigma)$. If any edge $\tau$ of a tetrahedron $\sigma$ is non-manifold, then we need to encode partial relation $R^*_{3,cl}(\tau)$ along $\tau$ through partial co-boundary relation $R^*_{1,cl}(\tau)$ with respect to $\sigma$. In this case, in the `aux_bnd` hash tables of $r_\sigma$, we store the *SimplexPointer* reference of the record $r_\tau$, which describes partial co-boundary relation $R^*_{1,cl}(\tau)$ with respect to $\sigma$.

Finally, we encode Euclidean coordinates of each vertex through a *local property* associated with each *Simplex* record related to a vertex in $\Sigma^0$.

### 7.2.2 Retrieving Link of a Simplex

In this section, we discuss an algorithm for retrieving link of a simplex $\sigma$ in any arbitrary simplicial 3-complex $\Sigma$ described through the NMIA data structure. In other words, we describe an algorithm for the LINK query in our implementation of the NMIA data structure. Recall that the link $\text{Lk}(\sigma)$ is given by all the simplices in $\Sigma$, which form the combinatorial boundary of simplices in $\text{St}(\sigma)$.
and are not incident at \( \sigma \).

As discussed in Section 4.1, each cluster in the star of a \( p \)-simplex \( \sigma \), with \( 0 \leq p < 3 \), is bounded by simplices belonging to \( \text{Lk}(\sigma) \). Specifically, each top \( k \)-simplex \( \sigma' \) in \( \text{St}(\sigma) \), with \( p < k \leq 3 \), corresponds to any \((k - p - 1)\)-face \( \theta \) of \( \sigma' \), which is the opposite face of \( \sigma \) in \( \sigma' \). Any \((k - p - 1)\)-face \( \theta \) is formed by all the vertices of \( \sigma' \), which do not belong to \( \sigma \). As a consequence, \( \theta \) (and all the simplices bounding \( \theta \)) belong to \( \text{Lk}(\sigma) \).

Thus, we can exploit this idea for retrieving link of a \( p \)-simplex \( \sigma \) in \( \Sigma \) in two steps. In the first step, we retrieve top simplices incident at \( \sigma \). In the second step, for each top \( k \)-simplex \( \sigma' \) in \( \text{St}(\sigma) \), with \( k > p \), we retrieve opposite \((k - p - 1)\)-face \( \theta \) of \( \sigma' \) with respect to \( \sigma' \), and all the simplices bounding \( \theta \). In other words, we exploit a solution similar to the approach exploited in Algorithm 6.12, regarding the \( \text{LINK} \) query in the IA* data structure.

Top simplices incident at any vertex \( v \) are retrieved through a breadth-first traversal of top edges and vertex-based clusters in \( \text{St}(v) \). Specifically, top edges in \( \text{St}(v) \) are directly encoded in partial co-boundary relation \( \mathcal{R}_{0,1}^*(v) \). Top triangles and tetrahedra in \( \text{St}(v) \) are retrieved by expanding each vertex-based cluster in \( \mathcal{R}_{0,3}(v) \) and in \( \mathcal{R}_{0,3}^*(v) \) through relation \( \mathcal{R}_{1,cl}^* \) and adjacency relation \( \mathcal{R}_{3,3} \). The time complexity of this operation is \( \mathcal{O}(v'_t) \), where \( v'_t \) is the number of top simplices incident at \( v \). For each top \( k \)-simplex \( \sigma \) in \( \text{St}(v) \), with \( 0 < k \leq 3 \), we consider opposite \((k - 1)\)-face \( \theta_v \) of \( v \) in \( \sigma \), plus simplices bounding \( \theta_v \), in the same way as in Section 6.3.5. Thus, the time complexity of the \( \text{LINK} \) query for any vertex \( v \) is \( \mathcal{O}(v'_t) \).

Any non top edge \( e \) is described through a \textit{GhostSimplexPointer} reference, which provides a reference to any top \( k \)-simplex \( \sigma \) in \( \text{St}(e) \), with \( 1 < k \leq 3 \). If \( e \) is manifold, then we can retrieve all the top \( k \)-simplices incident at \( e \) by transitive closure of adjacency relation \( \mathcal{R}_{k,k}^* \) along \( e \), starting from \( \sigma \). Top triangles incident at any non-manifold edge \( e \) are directly encoded in partial co-boundary relation \( \mathcal{R}_{1,cl}^*(e) \), while tetrahedra in \( \text{St}(e) \) are retrieved by expanding each 3-cluster in \( \mathcal{R}_{1,cl}^*(e) \) through adjacency relation \( \mathcal{R}_{3,3} \). The time complexity of these operations is \( \mathcal{O}(e'_t) \), where \( e'_t \) is the number of top simplices incident at \( e \). For each top \( k \)-simplex \( \sigma \) incident at any edge \( e \), with \( 1 < k \leq 3 \), we consider opposite \((k - 2)\)-face \( \theta_e \) of \( e \) in \( \sigma \), plus simplices bounding \( \theta_e \), in the same way as in Section 6.3.5. Thus, the time complexity of the \( \text{LINK} \) query for any edge \( e \) is \( \mathcal{O}(e'_t) \).

Any non top triangle \( f \) is described through a \textit{GhostSimplexPointer} reference, which provides a reference to any tetrahedron \( \sigma \) incident at \( f \). Note that \( f \) is shared between at most two tetrahedra \( \sigma \) and \( \sigma' \). This latter (if it exists) is adjacent to \( \sigma \) along \( f \). This information is directly encoded in adjacency relation \( \mathcal{R}_{3,3}^*(\sigma) \). In this case, \( \text{Lk}(f) \) is formed by two opposite vertices of \( t \) for \( \sigma \) and \( \sigma' \), respectively. Thus, the time complexity of the \( \text{LINK} \) query for any triangle \( f \) is \( \mathcal{O}(1) \).

### 7.2.3 Recognizing Non-Manifold Singularities

In this section, we discuss an algorithm for recognizing non-manifold singularities in any arbitrary simplicial 3-complex, described through the NMIA data structure. In other words, we propose an
Given an arbitrary simplicial 3-complex $\Sigma$, the NMIA data structure encodes all the top edges, and a representative simplex for each vertex-based cluster incident at any vertex $v$ in $\Sigma$. This latter corresponds to any connected component in $Lk(v)$. As a consequence, we can recognize any non-manifold vertex by exploiting the same approach used the IS data structure, introduced in Section 6.2.6. Conversely, any non-manifold edge $\tau$ is completely characterized by partial co-boundary relation $R^*_{1,d}(\tau)$. Algorithm 7.3 summarizes implementation of the IS_MANIFOLD query in our implementation of the NMIA data structure.

\begin{algorithm}
\caption{IS_MANIFOLD($\sigma$) - NMIA data structure}
\textbf{Input:} a GhostSimplexPointer reference for a $p$-simplex $\sigma$ in a simplicial 3-complex $\Sigma$
\textbf{Output:} true, if the $p$-simplex $\sigma$ is manifold; false, otherwise.
\begin{algorithmic}[1]
\State $\text{if } \dim(\sigma) = 0 \text{ then}$
\State \hspace{1em} \{Manifoldness test for a vertex\}
\State $n = \| R_{0,1}(\sigma) \| + \| R_{0,2}(\sigma) \| + \| R_{0,3}(\sigma) \|$
\State $\text{if } n > 2 \text{ then}$
\State \hspace{1em} return false
\State $\text{else if } n=2 \text{ then}$
\State \hspace{1em} return "LINK($\sigma$) contains only two vertices"
\State $\text{else if } n=1 \text{ then}$
\State \hspace{1em} for all $\lambda \in R_{0,1}(\sigma)$ do
\State \hspace{2em} if not IS_MANIFOLD($\lambda$) then
\State \hspace{3em} return false
\State \hspace{2em} end if
\State \hspace{1em} end for
\State \hspace{1em} return true
\State $\text{else}$
\State \hspace{1em} return true
\State $\text{end if}$
\State $\text{else if } \dim(\sigma) = 1 \text{ then}$
\State \hspace{1em} \{Manifoldness test for an edge\}
\State $\text{return } R^*_{1,d}(\sigma) = \emptyset$
\State $\text{else}$
\State \hspace{1em} return true
\State $\text{end if}$
\end{algorithmic}
\end{algorithm}

A non-manifold edge $\sigma$ is always characterized by partial co-boundary relation $R^*_{1,d}(\sigma)$, and, thus, the time complexity of this operation is $O(1)$. Recognition of a non-manifold vertex $\sigma$ is dominated by the analysis of all the edges incident at $\sigma$. Thus, in the worst case, the time complexity of this operation is linear in $O(\sigma^*_t)$, where $\sigma^*_t$ is the number of top simplices incident at $\sigma$. 

153
7.2.4 Construction Algorithm

In this section, we discuss an algorithm for building the NMIA data structure. Here, the challenge is to build the NMIA data structure, which describes a simplicial 3-complex Σ described by a soup of top simplices directly expressed in terms of their vertices. Recall that we also need Euclidean coordinates associated with all the vertices in the input soup of top simplices.

Basically, we store all the vertices and top simplices of Σ in the NMIA data structure and establish topological relations among them.

This is achieved through three auxiliary data structures, namely Inc, E, and B. Auxiliary data structure Inc is an array such that each location corresponds to a vertex v in Σ, and contains all the top triangles and tetrahedra in St(v). Auxiliary data structure E is an array such that each location corresponds to an edge e, and contains all the top triangles and tetrahedra in St(e).

Finally, auxiliary data structure B is an array with two locations, namely B[0] and B[1], which are recursively two arrays. Location B[0] contains a description of edges bounding top triangles and tetrahedra in Σ. In other words, it simulates co-boundary relations R_{1,2} and R_{1,3}, restricted to top triangles and tetrahedra. Conversely, location B[1] contains a description of triangles bounding tetrahedra in Σ. In other words, it simulates co-boundary relation R_{2,3}. Specifically, all the edges and triangles in B[0] and B[1] are described through a raw face (see Section 6.2.7). Each raw face is generated according to the schema discussed in Section 6.1.3, and is explicitly expressed by its vertices. Each raw face τ is associated with the unique identifier of a top p-simplex σ, with 1 < p ≤ 3, which is bounded by τ.

The NMIA data structure can be built by executing the following steps:

1. create four SimplicesContainer arrays, generate a local property for storing Euclidean coordinates, and, for each vertex v in the input soup of top simplices, generate a new Simplex record r_v in the SimplicesContainer array related to Σ^0, and store its Euclidean coordinates;

2. for each top edge w = (v_1, v_2), sort all the vertices of w in increasing order, generate a new Simplex record in the SimplicesContainer array related to Σ^1, and store w in partial co-boundary relations R_{0,1}^*(v_1) and R_{0,1}^*(v_2);

3. execute the following steps, for each top p-simplex σ, with 1 < p ≤ 3:
   i) generate a new Simplex record r_σ in the SimplicesContainer array related to Σ^p;
   ii) store all the vertices of σ, sorted in increasing order, in partial boundary relation R_{p,0}^*(σ), and store σ in Inc[v], for any vertex v bounding σ;
   iii) for each triangle σ, store all the raw 1-faces related to three edges bounding σ in B[0];
   iv) for each tetrahedron σ, store all the raw 1-faces related to six edges bounding σ in B[0], and store all the raw 2-faces related to four triangles bounding σ in B[1];
4. sort $B[0]$ according to the lexicographic order of vertices in raw edges: as a consequence, all the top triangles and tetrahedra incident at the same raw edge $\tau$ are consecutive in $B[0]$, and, thus, they can be stored in $E[\tau]$;

5. sort $B[1]$ according to the lexicographic order of vertices in raw triangles: as a consequence, all the tetrahedra incident at the same raw triangle are consecutive in $B[1]$, and, thus, it is possible to complete adjacency relation $R_{3,3}$;

6. for each vertex $v$, identify all the vertex-based clusters incident at $v$, and, thus, partial co-boundary relations $R_{0,k}^*(v)$, for $1 < k \leq 3$, by considering all the top simplices in $Inc$ and $E$, as performed in Algorithm 7.4 (VERTEX_BASED_CLUSTERS procedure);

7. for each edge $\tau$, identified in Step 4, perform the following steps:
   
   i) identify all the edge-based clusters incident at $\tau$ by considering all the top simplices in $E$, as performed in Algorithm 7.5 (EDGE_BASED_CLUSTERS function);

   ii) if $\tau$ is shared only between two top triangles $\sigma$ and $\sigma'$, respectively encoded in the Simplex records $r_\sigma$ and $r_{\sigma'}$, then $\sigma$ and $\sigma'$ are adjacent along $\tau$. Hence, we store the SimplexPointer reference of $\sigma'$ in the location of array $adj$ related to $\tau$ in $r_\sigma$, and vice versa.

   iii) If $\tau$ is shared by more than two top triangles, or by several top triangles and 3-clusters, then it is a non-manifold edge. Hence, we need to encode partial co-boundary relation $R_{1,cl}^*(\tau)$ in two steps. In the first step, we radially sort all the top triangles, and the representative simplices of all the 3-clusters in $St(\tau)$ in counter-clockwise order around $\tau$. Note that, in the worst case, each 3-cluster in $St(\tau)$ may be formed by only one tetrahedron, thus, we operate on $\tau^*_2$ top triangles and $\tau^*_3$ tetrahedra, incident at $\tau = (v_1, v_2)$. Recall that $\tau^*_2$ and $\tau^*_3$ are the numbers of triangles and tetrahedra in $St(\tau)$, respectively. For each tetrahedron $\sigma$ incident at $\tau$, we can consider one of its two triangles, which is incident at $\tau$. In this way, we reduce this problem to radial sorting of several triangles around an edge, as discussed in Section 7.1.3. This operation is equivalent to radially sort all the vertices of these triangles different than $v_1$ and $v_2$ around $\tau$ [PS85]. Resulting sequence of triangles is stored in a circular double-linked list. The time complexity of this operation is $O(\tau^*_1 \log(\tau^*_3))$, where $\tau^*_3$ is the number of top simplices in $St(\tau)$. At the end of this operation, we replace any triangle describing a 3-cluster $C_\sigma$ with the tetrahedron $\sigma$ which is the representative simplex of $C_\sigma$.

In the second step, we perform the following operations for each top simplex $\psi$ in $St(\tau)$, belonging to any cluster $C_\psi$:

   - generate a new Simplex record $r_{\tau}$, corresponding to $\tau$, in the SimplicesContainer array related to $\Sigma^1$. Then, store SimplexPointer references for the representative simplices of two clusters which precede and follow $C_\psi$ in counter-clockwise order around $\tau$ in the array $cob$ of $r_{\tau}$.

   - If $\psi$ is a top triangle encoded in the Simplex record $r_\psi$, then store the SimplexPointer reference of $r_{\tau}$ in the location of $adj$ array in $r_\psi$ corresponding to $\tau$. 

155
Algorithm 7.4 VERTEX_BASED_CLUSTERS\( (v, Inc, E) \)

**Input:** a GhostSimplexPointer reference for a vertex \( v \) in a simplicial 3-complex \( \Sigma \)
- the auxiliary data structure \( Inc \)
- the auxiliary data structure \( E \)

**Output:** the partial co-boundary relation \( \mathcal{R}_{0,k}^*(v) \), with \( 1 < k \leq 3 \)

1: for all \( \sigma' \) in \( Inc[v] \) do
2: \hspace{1em} let \( k := \dim(\sigma') \)
3: \hspace{1em} if \( \sigma' \) is not visited then
4: \hspace{2em} \{ Retrieval of a new vertex-based cluster in \( St(v) \) starts from \( \sigma' \) \}
5: \hspace{2em} \( \mathcal{R}_{0,k}^*(v) = \mathcal{R}_{0,k}^*(v) \cup \{\sigma'\} \)
6: \hspace{2em} let \( q \) be an empty queue
7: \hspace{2em} enqueue \( \sigma' \) in \( q \)
8: \hspace{2em} while \( q \) is not empty do
9: \hspace{3em} dequeue \( \sigma'' \) from \( q \)
10: \hspace{3em} if \( \sigma'' \) is not visited then
11: \hspace{4em} mark \( \sigma'' \) as visited
12: \hspace{4em} if \( \dim(\sigma'') = 2 \) then
13: \hspace{5em} \{ We continue traversal on all the edges incident at \( v \) in the top triangle \( \sigma'' \) \}
14: \hspace{5em} let \( i_v \) be the position of \( v \) in \( \mathcal{R}_{2,0}^*(\sigma'') \)
15: \hspace{5em} for all \( 0 \leq k \leq 2 \) do
16: \hspace{6em} if \( k \neq i_v \) then
17: \hspace{7em} let \( \eta_k \) be the edge in position \( k \) of the top triangle \( \sigma'' \)
18: \hspace{7em} for all \( \tau \) in \( E[\eta_k] \) do
19: \hspace{8em} enqueue \( \tau \) in \( q \)
20: \hspace{7em} end for
21: \hspace{6em} end if
22: \hspace{5em} end for
23: \hspace{4em} else
24: \hspace{5em} \{ We continue traversal on all the edges incident at \( v \) in the tetrahedron \( \sigma'' \) \}
25: \hspace{5em} for all \( 0 \leq k \leq 5 \) do
26: \hspace{6em} let \( \eta_k \) be the edge in position \( k \) of the tetrahedron \( \sigma'' \)
27: \hspace{6em} if \( \sigma \in \mathcal{R}_{1,0}(\eta_k) \) then
28: \hspace{7em} for all \( \tau \) in \( E[\eta_k] \) do
29: \hspace{8em} enqueue \( \tau \) in \( q \)
30: \hspace{7em} end for
31: \hspace{6em} end if
32: \hspace{5em} end for
33: \hspace{4em} end if
34: \hspace{4em} end if
35: \hspace{4em} end while
36: \hspace{2em} end if
37: end for

– If \( \psi \) is a tetrahedron encoded in the Simplex record \( r_\psi \), then store the Simplex-Pointer reference of \( r_\tau \) in a new entry of the aux\_bnd hash tables in \( r_\psi \).

Now, we can evaluate the time complexity of this algorithm, where we assume to generate new...
Algorithm 7.5 EDGE_BASED_CLUSTERS(\(\tau, E\))

**Input:** a GhostSimplexPointer reference for an edge \(\tau\) in a simplicial 3-complex \(\Sigma\)
the auxiliary data structure \(E\)

**Output:** the set \(S\) of all the edge-based clusters incident at \(\tau\)

1: let \(S := \emptyset\)
2: for all \(\sigma \in E[\tau]\) do
3:    if \(\sigma\) is not visited then
4:       \(S := S \cup \{\sigma\}\)
5:    if \(\dim(\sigma) = 3\) then
6:       {Compute transitive closure of the 2-adjacency relation restricted to tetrahedra in \(St(\tau)\)}
7:       let \(q\) be an empty queue
8:       enqueue \(\sigma\) in \(q\)
9:       while \(q\) is not empty do
10:          dequeue \(\sigma'\) from \(q\)
11:          if \(\sigma'\) is not visited then
12:             mark \(\sigma'\) as visited
13:             for all \(\eta \in R_3,3(\sigma')\) do
14:                if \(\tau \in R_3,1(\eta)\) then
15:                   enqueue \(\eta\) in \(q\)
16:             end if
17:          end for
18:       end while
19:    else
20:       {We visit a top triangle \(\sigma\) in \(St(\tau)\) }
21:       mark \(\sigma\) as visited
22:    end if
23: end if
24: end for
25: return \(S\)

**Simplex** records, raw faces, local properties, and **SimplicesContainer** arrays in \(O(1)\).

In Step 1, we generate a new **Simplex** record for each vertex, \(d + 1\) **SimplicesContainer** arrays, and save Euclidean coordinates for each vertex. Thus, the time complexity of this step is \(O(s_0 + d + 1)\).

In Step 2, we generate a new **Simplex** record for each top edge, and store its vertices, thus, the time complexity of this step is \(O(s_1^d)\). In Step 3, given a top \(p\)-simplex \(\sigma\), with \(1 < p \leq 3\), we sort all the vertices of \(\sigma\) in increasing order, and store them in \(R_3,p(\sigma)\). Then, we generate all the raw edges of each top triangle in \(B[0]\), and all the raw edges and raw triangles of a tetrahedron in \(B[1]\). Hence, the time complexity of this step is \(O(s_2^d + s_3)\). At the end of this step, \(B[0]\) contains \(b_0 \approx s_2^d + s_3\) raw edges, while \(B[1]\) contains \(b_1 \approx s_3\) raw triangles. In Step 4 and Step 5, we sort \(B[0]\) and \(B[1]\), thus, the time complexity of this step is \(O(b_0 \log b_0 + b_1 \log b_1) \approx O((s_2^d + s_3) \log(s_2^d + s_3))\).

In Step 6, we execute Algorithm 7.4 (VERTEX_BASED_CLUSTERS procedure). In this procedure, we identify all the vertex-based clusters incident at any vertex \(v\) in \(\Sigma\), namely partial
co-boundary relations $R^1_{3,1}(v)$, for $1 < k \leq 3$, by considering all the top simplices in $Inc$ and $E$. Recall that location $Inc[v]$ contains all the top simplices in $St(v)$. Here, a vertex-based cluster is retrieved as the transitive closure of 1-adjacency relation for triangles and tetrahedra in $St(v)$. Given a top simplex $\sigma''$ in $St(v)$, we continue our traversal only on edges of $\sigma''$ which are incident at $v$. Specifically, if $i_v$ is the position of $v$ in partial boundary relation $R^1_{2,0}(\sigma'')$, then edge of $\sigma''$ in position $i_v$ is not incident at $v$ (see Section 6.1.3). Top simplices incident at any edge $\tau$ are encoded in $E[\tau]$. Adjacency relation $R_{3,3}$ is directly encoded in the NMIA data structure, and boundary relation $R_{3,1}$ is retrieved in $O(1)$. At the end of these operations, we have visited all the top simplices in $E[\tau]$. As a consequence, the time complexity of Algorithm 7.4 is $O(v^*_t)$, where $v^*_t$ is the number of top simplices in $St(v)$. Thus, the time complexity of Step 6 is $O(v^*_t)$.

In Step 7, we execute Algorithm 7.5 (EDGE\_BASED\_CLUSTERS function). In this function, we identify all the top triangles and 3-clusters incident at $\tau$ by considering all the top simplices in $E$. Location $E[\tau]$ contains all the top triangles and tetrahedra in $St(\tau)$. Here, each top triangle is immediately retrieved. Conversely, a 3-cluster in $St(\tau)$ is retrieved as the transitive closure of adjacency relation among tetrahedra along triangles incident at $\tau$. Adjacency relation $R_{3,3}$ is directly encoded in the NMIA data structure, while boundary relation $R_{3,1}$ is retrieved in $O(1)$. At the end of these operations, we have visited all the top simplices in $E[\tau]$. As a consequence, the time complexity of Algorithm 7.5 is $O(\tau^*_t)$, where $\tau^*_t$ is the number of top simplices in $St(\tau)$.

Furthermore, we radially sort all the top simplices incident at any edge $\tau$, and this operation requires $O(\tau^*_t \log \tau^*_t)$. Finally, time complexity of the remaining operations is dominated by constructing $Simplex$ records for all the non-manifold edges bounding top triangles and tetrahedra. Recall that a top triangle $f$ has $N^3_f$ non-manifold edges, while each tetrahedron has $N^3_t$ non-manifold edges, as discussed in Section 5.3. The time complexity of Step 7 is linear in:

$$\sum_{\tau \in \Sigma^1} \tau^*_t \log \tau^*_t + \sum_{f \in \Sigma^2} N^2_f + \sum_{t \in \Sigma^3} N^3_t$$

Thus, if $N^2 = s^2_2 + s^2_3$, where $s^2_2$ and $s^2_3$ the number of, respectively, top triangles and tetrahedra, then the time complexity of our algorithm is linear in:

$$s_0 + s^1_1 + N^2 + N^3 \log N^3 + \sum_{v \in \Sigma^0} v^*_t + \sum_{\tau \in \Sigma^1} \tau^*_t \log \tau^*_t + \sum_{f \in \Sigma^2} N^2_f + \sum_{t \in \Sigma^3} N^3_t$$

### 7.3 Implementing the Simplified Incidence Graph

In this section, we propose an implementation of the Simplified Incidence Graph (SIG) data structure [DFGH04], discussed in Section 4.3, in the context of our Mangrove TDS framework. Recall that the SIG representation is an explicit, incidence-based, and dimension-independent data structure, which encodes all the simplices in an abstract simplicial complex, plus a subset of boundary and co-boundary relations for each simplex. As a consequence, the SIG data structure is represented through a global mangrove.
In Section 7.3.1, we discuss a complete description of the SIG data structure in terms of internal data structures offered by the Mangrove TDS framework. The SIG data structure encodes the same boundary relations as the IS data structure, thus, boundary of any simplex is retrieved by using the same algorithm introduced in Section 6.2.2. In [DFGH04] the authors introduce algorithms for retrieving co-boundary and adjacency relations, which can be reused with minor modifications as implementations of STAR and ADJACENCY queries. Also, LINK query can be implemented by reusing the same algorithm used in the IS data structure, introduced in Section 6.2.5, where we exploit implementation of the STAR query offered by the SIG data structure. Note that the time complexity of the STAR, ADJACENCY, and LINK queries is $O(\sigma^t_\star)$, where $\sigma^t_\star$ is the number of top simplices incident at any simplex $\sigma$. Thus, these queries are optimal only for simplicial complexes embedded in $\mathbb{E}^3$, while they are local for simplicial $h$-complexes, with $h \geq 4$. In Section 7.3.2, we describe an algorithm for executing the IS,MANIFOLD query. Finally, in Section 7.3.3, we introduce an algorithm for building the SIG data structure from a soup of top simplices directly expressed in terms of their vertices.

### 7.3.1 Implementation of the Data Structure

In this section, we describe a complete description of the SIG data structure in terms of internal data structures offered by the Mangrove TDS framework.

The SIG data structure encodes all the simplices in any abstract simplicial $d$-complex $\Sigma$, thus we need $d + 1$ SimplicesContainer arrays, one for each collection $\Sigma^p$ of $p$-simplices in $\Sigma$, with $0 \leq p \leq d$.

Each $p$-simplex $\sigma$ in $\Sigma$ is encoded through a Simplex record $r_\sigma$ in the SimplicesContainer array related to $\Sigma^p$. Each record $r_\sigma$ is assigned to a unique SimplexPointer reference. We exploit several SimplexPointer references for storing all the simplices in boundary and partial co-boundary relations $R_{p,p-1}(\sigma)$ and $R^*_{p,q}(\sigma)$, with $p < q \leq d$. Specifically, in the bnd array of $r_\sigma$, we store $p + 1$ SimplexPointer references, one for each $(p - 1)$-simplex in $R_{p,p-1}(\sigma)$. Conversely, in the cob array of $r_\sigma$, we store several SimplexPointer references, one for each representative $q$-simplex of a $q$-cluster in $St(\sigma)$ (with $p < q \leq d$), encoded in partial co-boundary relation $R^*_{p,q}$. In this case, arrays adj and aux_bnd in $r_\sigma$ are not used, and they are empty.

### 7.3.2 Recognizing Non-Manifold Singularities

In this section, we describe an algorithm for recognizing non-manifold singularities in any abstract simplicial $d$-complex, with $d \leq 3$, described through the SIG data structure. In other words, we propose an algorithm for the IS,MANIFOLD query in our implementation of the SIG data structure.

Given a simplicial $d$-complex $\Sigma$, with $d \leq 3$, the SIG data structure encodes, for each $p$-simplex $\sigma$, with $0 \leq p < d$, representative $k$-simplices of all the $k$-clusters in $St(\sigma)$, with $0 < k \leq d$. Hence,
we propose a slight modification of the solution discussed in Section 6.3.6, regarding the IA*
data structure. Basically, the key idea of this approach consists of characterizing a non-manifold
simplex \( \sigma \) in terms of clusters in \( St(\sigma) \) instead of connected components in \( Lk(\sigma) \). Algorithm 7.6
summarizes our implementation of the IS_MANIFOLD query in the SIG data structure.

Algorithm 7.6 IS_MANIFOLD(\( \sigma \)) - SIG data structure

**Input:** a SimplexPointer reference for a \( p \)-simplex \( \sigma \) in a simplicial \( d \)-complex \( \Sigma \)

**Output:** true, if the \( p \)-simplex \( \sigma \) is manifold; false, otherwise.

1: let \( p := \text{dim}(\sigma) \), \( n = \sum \| R^*_p q(\sigma) \| \), for \( p < q \leq d \)
2: if \( p=0 \) then
3: {Manifoldness test for a vertex}
4: if \( n > 2 \) then
5: return false
6: else if \( n=2 \) then
7: return “\( St(\sigma) \) contains only two top edges”
8: else
9: for all \( \lambda \in R^*_{0,1}(\sigma) \) do
10: if not IS_MANIFOLD(\( \lambda \)) then
11: return false
12: end if
13: end for
14: return true
15: end if
16: else if \( p=1 \) then
17: {Manifoldness test for an edge}
18: if \( n > 2 \) then
19: return false
20: else if \( n<2 \) then
21: return true
22: else
23: return “\( \text{LINK}(\sigma) \) contains only two vertices”
24: end if
25: end else
26: return true
27: end if

Recognition of a non-manifold edge \( \sigma \) is dominated by the retrieval of \( Lk(\sigma) \) through the LINK
query. Recall that the LINK query in the SIG data structure is basically the same query in the
IS data structure. Thus, in the worst case, the time complexity of this operation is \( \mathcal{O}(\sigma^*_t) \), where
\( \sigma^*_t \) is the number of top simplices in \( St(\sigma) \).

Recognition of a non-manifold vertex \( \sigma \) is dominated by the analysis of all the edges incident at \( \sigma \).
Co-boundary relation \( R^*_{0,1}(\sigma) \) can be retrieved in \( \mathcal{O}(\sigma^*_t) \). Also, we can recognize a non-manifold
dge \( \lambda \) in \( \mathcal{O}(\lambda^*_t) \). In this case, the time complexity of IS_MANIFOLD query is linear in:

\[
\sigma^*_t + \sum_{\lambda \in R^*_{0,1}(\sigma)} \lambda^*_t \approx \sigma^*_t
\]
since, for each edge $\lambda$, all the top simplices in $St(\lambda)$ are also in $St(\sigma)$.

As a consequence, we can state that the time complexity of the IS,MANIFOLD query, for any $p$-simplex $\sigma$, is linear in the number $\sigma^t_*$ of top simplices in $St(\sigma)$.

### 7.3.3 Construction Algorithm

In this section, we describe a dimension-independent algorithm for building the SIG data structure from a soup of top simplices directly expressed in terms of their vertices. Recall that there exists an algorithm for building the SIG data structure from an existing IG data structure [DFGH04, Hui08]. Here, the challenge is to simulate this algorithm without building explicitly any Incidence Graph.

The SIG data structure can be built in two steps through a slight modification of the algorithm used for building the IS data structure, introduced in Section 6.2.7.

In the first step, we generate all the simplices in $\Sigma$ and their boundary relations through auxiliary data structure $B$, already used for constructing the IS data structure. Recall that auxiliary data structure $B$ is an array with $d - 1$ locations such that each location $B[i]$, with $0 \leq i < d - 1$, is recursively an array, which contains all the raw $(i + 1)$-faces bounding several raw $(i + 2)$-faces (see Section 6.2.7). Here, we also exploit auxiliary data structures $Inc$ and $Adj$. Auxiliary data structure $Inc$ is an array such that each location corresponds to a $p$-simplex $\sigma$ in $\Sigma$, with $0 \leq p < d$, and contains all the top $h$-simplices in $St(\sigma)$, with $p < h \leq d$. Similarly, auxiliary data structure $Adj$ is an array such that each location corresponds to a top $p$-simplex $\sigma$ in $\Sigma$, with $0 \leq p < d$, and contains all the top $p$-simplices adjacent to $\sigma$.

In the second step, we identify all the clusters in the star of any simplex by exploiting the same approach used for building the IA$^*$ data structure, introduced in Section 6.3.7.

We can construct the SIG data structure by executing the following steps:

1. create $d + 1$ SimplicesContainer arrays, and, for each vertex $v$ in the input soup of top simplices, generate a new Simplex record $r_v$ in the SimplicesContainer array related to $\Sigma^0$;

2. for each top edge $w = (v_1, v_2)$, sort all the vertices of $w$ in increasing order, generate a new Simplex record in the SimplicesContainer array related to $\Sigma^1$, and store $w$ in partial co-boundary relations $R^0_{h,1}(v_1)$ and $R^0_{h,1}(v_2)$;

3. for each top $p$-simplex $\tau$, with $2 \leq p \leq d$, sort all the vertices of $\tau$ in increasing order, generate all the $\tau^p_k = \binom{p+1}{k+1}$ $k$-faces of $\tau$, and store the corresponding raw faces in $B[k - 1]$, for all $1 \leq k < p$;

4. execute the following steps, for $i = 0, \ldots, d - 2$:
   
   i) sort $B[i]$ with respect to lexicographic order of the vertices in raw $(i + 1)$-faces: as a consequence, all the unique $(i + 1)$-simplices $\sigma$ in $\Sigma$ shared by several $(i + 2)$-simplices $\lambda$, are stored in consecutive locations of $B[i]$;
ii) for every \((i + 1)\)-simplex \(\sigma\) identified at the previous step, generate a new Simplex record \(r_\sigma\) in the SimplicesContainer array related to \(\Sigma^{i+1}\), and store the SimplexPointer reference of \(r_\sigma\) in place of the old identifier of \(\sigma\), assigned in Step 3;

iii) if \(\sigma\) is an edge \((v_1, v_2)\), add \(v_1\) and \(v_2\) to boundary relation \(R_{1,0}(\sigma)\);

iv) given all the non top \((i + 2)\)-simplices \(\lambda\) in \(\text{St}(\sigma)\) (retrieved in Step 4i)), add \(\sigma\) to boundary relation \(R_{i+2,i+1}(\lambda)\);

v) store in Adj all the top \((i + 2)\)-simplices \(\tau\) (identified in Step 4i)) which share a \((i + 1)\)-simplex \(\sigma\): in this way, given a top \((i + 2)\)-simplex \(\tau\), \(\text{Adj}[\tau]\) contains all the top \((i + 2)\)-simplices adjacent to \(\tau\).

5. Generate a new Simplex record for each top \(p\)-simplex \(\tau\), with \(2 \leq p \leq d\), and add all the \((p - 1)\)-faces \(\sigma\) of \(\tau\) to boundary relation \(R_{p,p-1}(\tau)\). Add \(\tau\) to locations \(\text{Inc}[\psi]\) related to all the faces \(\psi\) bounding \(\tau\).

6. For \(0 \leq p < d\) and \(p < q \leq d\), retrieve partial co-boundary relation \(R^*_{p,q}(\sigma)\), for each \(p\)-simplex \(\sigma\), namely representative simplices of all the \(q\)-clusters in the star of \(\sigma\).

First five steps are basically the same operations performed for building the IS data structure, introduced in Section 6.2.7. In Step 6, we retrieve all the \(q\)-clusters incident at any \(p\)-simplex, with \(0 \leq p < d\) and \(p < q \leq d\), through a generalization of Algorithm 6.15, introduced in Section 6.3.7. Here, the key idea is to exploit the content of \(\text{Inc}\) and \(\text{Adj}\) not only on vertices, but also on other simplices in \(\Sigma\). It is quite clear that time complexity of this operation is \(O(\|\text{St}(\sigma)\|)\), where \(\|\text{St}(\sigma)\|\) is the number of simplices incident at \(\sigma\).

As a consequence, the time complexity of our algorithm is linear in:

\[
s_0 + s'_1 + N' \log(N') + \sum_{\sigma \in \Sigma} \|\text{St}(\sigma)\|
\]

where \(N' = \sum_{p=2}^{d} s'_p\) is the sum of the number \(s'_p\) of top \(p\)-simplices in \(\Sigma\), with \(p > 2\).

7.4 Implementing the Incidence Graph

In this section, we propose an implementation of the Incidence Graph (IG) [Ede87], restricted to simplicial complexes [DF03], in the context of the Mangrove TDS framework. We have discussed this topological data structure in Section 4.2. Recall that the Incidence Graph is an explicit, incidence-based, and dimension-independent data structure, which encodes all the simplices in an abstract simplicial complex, plus a subset of boundary and co-boundary relations for each simplex. As a consequence, the IG data structure is represented through a global mangrove, namely the IG-graph, which we have introduced in Section 4.2.

In Section 7.4.1, we propose a complete description of the IG data structure in terms of internal data structures offered by the Mangrove TDS framework. The IG data structure encodes the same
boundary relations as the IS data structure, thus, boundary of any simplex is retrieved by using the same algorithm introduced in Section 6.2.2. In Section 4.2, we have introduced and discussed algorithms for retrieving co-boundary and adjacency relations for any simplex, which can be reused with minor modifications as implementations of the STAR and ADJACENCY queries. Also, the LINK query can be implemented by reusing the same algorithm used in the IS data structure, introduced in Section 6.2.5, where we exploit implementation of the STAR query offered by the IG data structure. Note that the STAR, ADJACENCY, and LINK queries are optimal in the IG data structure. In Section 7.4.2, we propose an algorithm for executing the IS MANIFOLD query.

Finally, in Section 7.4.3, we discuss an algorithm for building the IG data structure from a soup of top simplices directly expressed in terms of their vertices.

7.4.1 Implementation of the Data Structure

In this section, we propose a complete description of the IG data structure, restricted to simplicial complexes, in terms of the data structures offered by the Mangrove TDS framework.

The IG data structure encodes all the simplices in any abstract simplicial $d$-complex $\Sigma$, thus we need $d+1$ SimplicesContainer arrays, one for each collection $\Sigma^p$ of $p$-simplices in $\Sigma$, with $0 \leq p \leq d$. Each $p$-simplex $\sigma$ in $\Sigma$ is encoded through a Simplex record $r_\sigma$ in the SimplicesContainer array related to $\Sigma^p$. Each record $r_\sigma$ is assigned to a unique SimplexPointer reference. We exploit several SimplexPointer references for storing all the simplices in boundary and co-boundary relations $R_{p,p-1}(\sigma)$ and $R_{p,p+1}(\sigma)$. Specifically, in the $bnd$ array of $r_\sigma$, we store $p+1$ SimplexPointer references, one for each $(p-1)$-simplex in $R_{p,p-1}(\sigma)$. Conversely, in the $cob$ array of $r_\sigma$, we store several SimplexPointer references, one for each $(p+1)$-simplex in $R_{p,p+1}(\sigma)$. In this case, arrays $adj$ and $aux_bnd$ in $r_\sigma$ are not used, and they are empty.

7.4.2 Recognizing Non-Manifold Singularities

In this section, we propose an algorithm for recognizing non-manifold singularities in any abstract simplicial $d$-complex, with $d \leq 3$, described through the IG data structure. In other words, we propose an algorithm for the IS_MANIFOLD query in our implementation of the IG data structure.

Given any abstract simplicial $d$-complex $\Sigma$, with $d \leq 3$, we can characterize non-manifold vertices and edges as discussed in Section 6.2.6. In any case, the IG data structure does not provide the number of connected components in the link of a simplex $\sigma$ in $\Sigma$.

As a consequence, we have to retrieve this information through Algorithm 7.7 (IDENTIFY_COMPONENTS function). Here, we assume that the input list $l_\sigma$ contains SimplexPointer references of all the simplices in the link of $\sigma$. At the end of this algorithm, we visit each simplex $\sigma'$ in $Lk(\sigma)$ by associating $\sigma'$ with a connected component in $Lk(\sigma)$. It can be easily proven that a connected component in $Lk(\sigma)$ corresponds to a biconnected component in the IG star-graph $G^G_{\sigma}$ of $\sigma$, introduced in Section 4.2. This means that we visit all the nodes and arcs in $G^G_{\sigma}$.
As a consequence, time complexity of Algorithm 7.7 is $O(\|St(\sigma)\|)$, where $\|St(\sigma)\|$ is the number of all the simplices incident at $\sigma$.

Algorithm 7.7 IDENTIFY_COMPONENTS($\sigma$, $l_\sigma$) - IG data structure

Input: a SimplexPointer reference referring a $p$-simplex in a simplicial $d$-complex $\Sigma$
the list $l_\sigma$ containing SimplexPointer references of simplices in $Lk(\sigma)$

Output: the number $n$ of connected components in $Lk(\sigma)$

1: let $p := \text{dim}(\sigma)$, $n := 0$
2: for all $\sigma' \in l_\sigma$ do
3: if $\sigma'$ is not visited then
4: let $q$ an empty queue
5: {Retrieval of a new connected component starts from $\sigma'$}
6: enqueue $\sigma'$ in $q$
7: $n := n + 1$
8: while $q$ is not empty do
9: dequeue $\sigma''$ from $q$
10: if $\sigma''$ is not visited then
11: mark $\sigma''$ as visited
12: for all $\lambda \in \text{STAR}(\sigma'')$ do
13: if $\lambda \in l_\sigma$ then
14: enqueue $\lambda$ in $q$
15: end if
16: end for
17: for all $\lambda \in \text{BOUNDARY}(\sigma'')$ do
18: if $\lambda \in l_\sigma$ then
19: enqueue $\lambda$ in $q$
20: end if
21: end for
22: end if
23: end while
24: end if
25: end for
26: return $n$

At this point, we are able to exploit information provided by the IDENTIFY_COMPONENTS function for recognizing any non-manifold simplex in $\Sigma$ by following the same algorithm described in Section 6.2.6. Algorithm 7.8 (IS_MANIFOLD query) summarizes these operations.

Recognition of a non-manifold edge $\sigma$ is dominated by the retrieval of $Lk(\sigma)$ and its connected components through the IDENTIFY_COMPONENTS function. Note that the LINK query in the IG data structure is optimal. Thus, the time complexity of this operation is $O(\|St(\sigma)\|)$, where $\|St(\sigma)\|$ is the number of simplices incident at $\sigma$.

Recognition of a non-manifold vertex $\sigma$ is dominated by the analysis of edges incident at $\sigma$. Co-boundary relation $R_{0,1}(\sigma)$ is optimal, and it can be retrieved in $\|R_{0,1}(\sigma)\|$, i.e., it is linear in the number of edges incident at $\sigma$. Also, we can recognize a non-manifold edge $\lambda$ in $O(\|St(\lambda)\|)$. In
Algorithm 7.8 IS_MANIFOLD(σ) - IG data structure

**Input:** a SimplexPointer reference for a $p$-simplex $σ$ in a simplicial $d$-complex $Σ$

**Output:** true, if $σ$ is manifold; false, otherwise.

1: if $\dim(σ) = 0$  
2: \{Manifoldness test for a vertex\} 
3: let $l_σ = \text{LINK}(σ)$, $n := \text{IDENTIFY\_COMPONENTS}(σ, l_σ)$ 
4: if $n > 2$ then  
5: \text{return false}  
6: else if $n=2$ then  
7: \text{return “$l_σ$ contains only 2 vertices”}  
8: else if $n=1$ then  
9: for all $λ \in R_{0,1}(σ)$ do  
10: if not IS_MANIFOLD($λ$) then  
11: \text{return false}  
12: end if  
13: end for  
14: \text{return true}  
15: else  
16: \text{return true}  
17: end if  
18: else if $\dim(σ) = 1$ then  
19: \{Manifoldness test for an edge\} 
20: if $R_{1,2}(σ)$ contains only two top triangles then  
21: \text{return true}  
22: else if $R_{1,2}(σ)$ contains only one top triangle then  
23: \text{return true}  
24: else  
25: let $l_σ = \text{LINK}(σ)$  
26: if $l_σ$ contains only 2 vertices then  
27: \text{return true}  
28: else if $\text{IDENTIFY\_COMPONENTS}(σ, l_σ) \geq 2$ then  
29: \text{return false}  
30: else  
31: \text{return true}  
32: end if  
33: end if  
34: else  
35: \text{return true}  
36: end if

In this case, the time complexity of the IS_MANIFOLD query is linear in:

$$\| R_{0,1}(σ) \| + \sum_{λ \in R_{0,1}(σ)} \| St(λ) \| \approx \| St(σ) \|$$

since, for each edge $λ$, simplices in $St(λ)$ are also in $St(σ)$.

As a consequence, we can state that time complexity of the IS_MANIFOLD query, for any $p$-simplex $σ$, is linear in the number $\| St(σ) \|$ of simplices in $St(σ)$. 

165
7.4.3 Construction Algorithm

In this section, we propose a dimension-independent algorithm for building the IG data structure from a soup of top simplices directly expressed in terms of their vertices.

The IG data structure can be built through a slight modification of the first five steps in the algorithm used for building the IS data structure, introduced in Section 6.2.7. Here, we exploit only the auxiliary data structure \(B\) for generating all the simplices from the input sequence of top simplices. Recall that \(B\) is an array with \(d - 1\) locations, where each location \(B[i]\), with \(0 \leq i < d - 1\), is recursively an array, which contains all the raw \((i + 1)\)-faces bounding several raw \((i + 2)\)-faces. In other words, \(B\) contains all the simplices in a simplicial \(d\)-complex \(\Sigma\), and their boundary relations.

As a consequence, in Step 4 and in Step 5, we complete the Incidence Graph, since boundary and co-boundary relations are symmetric. In fact, we can store simplices directly in co-boundary relations \(R^*_{p,p+1}\) in the IG data structure, with \(0 \leq p < d\). Thus, time complexity of this algorithm is linear in:

\[
s_0 + s_1^t + N^t \log(N^t)
\]

where \(N^t = \sum_{p=2}^{d} s_p^t\) is the sum of the number \(s_p^t\) of top \(p\)-simplices in \(\Sigma\), with \(p > 2\).

7.5 Experimental Comparisons

In this section, we propose quantitative comparisons among running times of topological queries and building algorithms for each topological data structure discussed in Chapters 4 and 5, which we have implemented in the Mangrove TDS framework. We present our results for simplicial 2- and 3-complexes.

Specifically, we consider the IS(2D) and IA\(^*\)(2D) data structures, namely specializations of the IS and IA\(^*\) data structures for simplicial 2-complexes, introduced, respectively, in Sections 4.4.1 and 5.4.1. We also consider the IG(2D) and SIG(2D) data structures, namely specializations of the IG and SIG data structures for arbitrary simplicial 2-complexes, introduced in Section 4.4.1. Finally, we consider the TS data structure, discussed in Section 5.2, which is a data structure specific for simplicial 2-complexes embedded in the Euclidean space \(\mathbb{E}^3\).

Similarly, we consider the IS(3D) and IA\(^*\)(3D) data structures, namely specializations of the IS and IA\(^*\) data structures for simplicial 3-complexes, which we have introduced, respectively, in Sections 4.4.2 and 5.4.2. We also consider the IG(3D) and SIG(3D) data structures, namely specializations of the IG and SIG data structures to simplicial 3-complexes, introduced in Section 4.4.2. Finally, we consider the NMIA data structure, discussed in Section 5.3, which is a data structure specific for simplicial 3-complexes embedded in the Euclidean space \(\mathbb{E}^3\).

In Sections 7.5.1 and 7.5.2, we propose, respectively, our running times and comparisons regarding
execution of the BOUNDARY and STAR queries. We propose our results regarding execution of the ADJACENCY and LINK queries in Sections 7.5.3 and 7.5.4, respectively. In Section 7.5.5, we propose a quantitative analysis regarding execution of the IS_MANIFOLD query. Finally, in Section 7.5.6, we discuss our results regarding construction algorithms of all the data structures from a soup of top simplices directly expressed in terms of their vertices.

Digital shapes used in our tests are freely available [GGG09], and have been already analyzed in Sections 4.4 and 5.4. We have tested our implementation of the Mangrove TDS framework on a workstation with 1.8 Ghz Intel®Core 2 Duo processor and 3 Gb of RAM. Running times presented in this section are expressed in milliseconds, and are retrieved through a standard timer in the platform-independent QT Library® [QT08].

### 7.5.1 Experimental Analysis of Boundary Retrieval

In this section, we propose comparisons regarding execution of the BOUNDARY query for all the data structures implemented in the Mangrove TDS framework. We compare approaches exploited in the IS and in IA* data structures, discussed in Sections 6.2.2 and 6.3.2, respectively.

We summarize average running times of the BOUNDARY query for simplicial 2-complexes in Table 7.1. We denote running times required for retrieving boundary of \( p \)-simplices, with \( 0 < p \leq 2 \), as \( B^{p, 2D}_{IS} \) and \( B^{p, 2D}_{IA*} \) in the IS(2D) and IA*(2D) data structures, respectively.

<table>
<thead>
<tr>
<th>Shape</th>
<th>( B^{1, 2D}_{IS} )</th>
<th>( B^{1, 2D}_{IA*} )</th>
<th>( B^{2, 2D}_{IA*} )</th>
<th>( B^{2, 2D}_{IS} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cone</td>
<td>0.0057</td>
<td>0.0060</td>
<td>0.0063</td>
<td>0.0086</td>
</tr>
<tr>
<td>Crumb</td>
<td>0.0054</td>
<td>0.0059</td>
<td>0.0062</td>
<td>0.0086</td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>0.0053</td>
<td>0.0053</td>
<td>0.0061</td>
<td>0.0087</td>
</tr>
<tr>
<td>Football 1</td>
<td>0.0050</td>
<td>0.0055</td>
<td>0.0059</td>
<td>0.0085</td>
</tr>
<tr>
<td>Football 2</td>
<td>0.0058</td>
<td>0.0061</td>
<td>0.0068</td>
<td>0.0091</td>
</tr>
<tr>
<td>Torus</td>
<td>0.0053</td>
<td>0.0058</td>
<td>0.0065</td>
<td>0.0095</td>
</tr>
<tr>
<td>Armchair</td>
<td>0.0057</td>
<td>0.0063</td>
<td>0.0067</td>
<td>0.0090</td>
</tr>
<tr>
<td>800-Cubes</td>
<td>0.0063</td>
<td>0.0066</td>
<td>0.0069</td>
<td>0.0092</td>
</tr>
<tr>
<td>Cylinders</td>
<td>0.0063</td>
<td>0.0067</td>
<td>0.0069</td>
<td>0.0089</td>
</tr>
<tr>
<td>Pinched-pie</td>
<td>0.0059</td>
<td>0.0061</td>
<td>0.0063</td>
<td>0.0090</td>
</tr>
<tr>
<td>Twist</td>
<td>0.0062</td>
<td>0.0065</td>
<td>0.0068</td>
<td>0.0091</td>
</tr>
<tr>
<td>Robot</td>
<td>0.0057</td>
<td>0.0060</td>
<td>0.0064</td>
<td>0.0090</td>
</tr>
<tr>
<td>Balance</td>
<td>0.0065</td>
<td>0.0068</td>
<td>0.0070</td>
<td>0.0091</td>
</tr>
<tr>
<td>Carter</td>
<td>0.0058</td>
<td>0.0061</td>
<td>0.0064</td>
<td>0.0084</td>
</tr>
<tr>
<td>Chandelier</td>
<td>0.0055</td>
<td>0.0059</td>
<td>0.0062</td>
<td>0.0083</td>
</tr>
<tr>
<td>Frame</td>
<td>0.0061</td>
<td>0.0064</td>
<td>0.0068</td>
<td>0.0087</td>
</tr>
<tr>
<td>Tower</td>
<td>0.0062</td>
<td>0.0068</td>
<td>0.0071</td>
<td>0.0090</td>
</tr>
<tr>
<td>Tower-wir</td>
<td>0.0063</td>
<td>0.0069</td>
<td>0.0074</td>
<td>0.0089</td>
</tr>
</tbody>
</table>

Table 7.1: Average running times for retrieving boundary of edges and triangles in digital 2D shapes, represented through the IS(2D) (\( B^{1, 2D}_{IS} \) and \( B^{2, 2D}_{IS} \), respectively) and IA*(2D) (\( B^{1, 2D}_{IA*} \) and \( B^{2, 2D}_{IA*} \), respectively) data structures.
We summarize average running times of the BOUNDARY query for simplicial 3-complexes in Table 7.2. We denote running times required for retrieving boundary of \( p \)-simplices, with \( 0 < p \leq 3 \), as \( B_{IS}^{3D} \) and \( B_{IA}^{3D} \) in the IS(3D) and IA* (3D) data structures, respectively.

<table>
<thead>
<tr>
<th>Shape</th>
<th>( B_{IS}^{3D} )</th>
<th>( B_{IA}^{3D} )</th>
<th>( B_{IS}^{2D} )</th>
<th>( B_{IA}^{2D} )</th>
<th>( B_{IS}^{1D} )</th>
<th>( B_{IA}^{1D} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>0.0061</td>
<td>0.0063</td>
<td>0.0068</td>
<td>0.0072</td>
<td>0.0075</td>
<td>0.0102</td>
</tr>
<tr>
<td>Cylinder</td>
<td>0.0065</td>
<td>0.0066</td>
<td>0.0073</td>
<td>0.0076</td>
<td>0.0079</td>
<td>0.0106</td>
</tr>
<tr>
<td>Gargoyle</td>
<td>0.0055</td>
<td>0.0057</td>
<td>0.0064</td>
<td>0.0068</td>
<td>0.0072</td>
<td>0.0097</td>
</tr>
<tr>
<td>Rings</td>
<td>0.0056</td>
<td>0.0059</td>
<td>0.0063</td>
<td>0.0066</td>
<td>0.0069</td>
<td>0.0093</td>
</tr>
<tr>
<td>Torus 3D</td>
<td>0.0058</td>
<td>0.0060</td>
<td>0.0061</td>
<td>0.0065</td>
<td>0.0069</td>
<td>0.0094</td>
</tr>
<tr>
<td>Arc</td>
<td>0.0060</td>
<td>0.0065</td>
<td>0.0071</td>
<td>0.0074</td>
<td>0.0078</td>
<td>0.0105</td>
</tr>
<tr>
<td>Balloon</td>
<td>0.0059</td>
<td>0.0063</td>
<td>0.0070</td>
<td>0.0073</td>
<td>0.0076</td>
<td>0.0102</td>
</tr>
<tr>
<td>Bucket</td>
<td>0.0060</td>
<td>0.0065</td>
<td>0.0071</td>
<td>0.0074</td>
<td>0.0079</td>
<td>0.0106</td>
</tr>
<tr>
<td>Chime</td>
<td>0.0060</td>
<td>0.0064</td>
<td>0.0068</td>
<td>0.0072</td>
<td>0.0077</td>
<td>0.0104</td>
</tr>
<tr>
<td>Flasks</td>
<td>0.0059</td>
<td>0.0061</td>
<td>0.0065</td>
<td>0.0069</td>
<td>0.0072</td>
<td>0.0097</td>
</tr>
<tr>
<td>Halves</td>
<td>0.0063</td>
<td>0.0066</td>
<td>0.0067</td>
<td>0.0072</td>
<td>0.0075</td>
<td>0.0101</td>
</tr>
<tr>
<td>Sierpinski</td>
<td>0.0063</td>
<td>0.0066</td>
<td>0.0073</td>
<td>0.0076</td>
<td>0.0079</td>
<td>0.0091</td>
</tr>
<tr>
<td>Teapot</td>
<td>0.0059</td>
<td>0.0065</td>
<td>0.0069</td>
<td>0.0071</td>
<td>0.0074</td>
<td>0.0099</td>
</tr>
<tr>
<td>Wheel</td>
<td>0.0062</td>
<td>0.0066</td>
<td>0.0071</td>
<td>0.0074</td>
<td>0.0077</td>
<td>0.0104</td>
</tr>
</tbody>
</table>

Table 7.2: Average running times for retrieving boundary of edges, triangles, and tetrahedra in digital 3D shapes, represented through the IS(3D) (\( B_{IS}^{1D} \), \( B_{IS}^{2D} \), and \( B_{IS}^{3D} \)) and IA* (3D) (\( B_{IA}^{1D} \), \( B_{IA}^{2D} \), and \( B_{IA}^{3D} \)) data structures.

These tests show that the BOUNDARY query in the IA* data structure is, on average, as efficient as in the IS data structure. Note that our results do not depend on the dimension of simplicial complexes of interest. On average, \( B_{IS}^{1,2D} \approx B_{IA}^{1,2D} \) and \( B_{IS}^{3,2D} \approx B_{IA}^{3,2D} \).

It is also interesting to evaluate performances of these algorithms. In a simplicial 2-complex, \( B_{IA}^{1,2D} \) is about 5.5% larger than \( B_{IS}^{1,2D} \). Similarly, in a simplicial 3-complex, \( B_{IA}^{3,2D} \) is approximately 5.4% larger than \( B_{IS}^{3,2D} \), and \( B_{IA}^{3,3D} \) is approximately only 5% larger than \( B_{IS}^{3,3D} \). Note that these simplices are not directly encoded in the IA* data structure, thus we generate GhostSimplexPointer references of interest by visiting hierarchies of faces (see Section 6.3.2). These results show that overhead needed to retrieve GhostSimplexPointer references related to boundary simplices is negligible. Finally, in a simplicial 2-complex, \( B_{IA}^{3,2D} \) is about 34% smaller than \( B_{IS}^{3,2D} \). Conversely, in a simplicial 3-complex, \( B_{IA}^{3,3D} \) is about 35% smaller than \( B_{IS}^{3,3D} \). In this case, we generate GhostSimplexPointer references of interest without performing a traversal of the IA* boundary graph. Also, this result shows the validity of GhostSimplexPointer references, since they improve the expressive power of the IA* data structure.

### 7.5.2 Experimental Analysis of Star Retrieval

In this section, we propose comparisons regarding execution of the STAR query for all the data structures implemented in the Mangrove TDS framework. We have designed algorithms for retrieving the star of a simplex in the IS and IA* data structures, respectively, in Sections 6.2.3
and 6.3.3. In Section 7.1.2, we have discussed an implementation of the STAR query for the TS data structure. Implementations of the STAR query in the NMIA, SIG, and IG data structures have been discussed in [DFH03, DFGH04, Hui08] and in Section 4.2. Since we analyze simplicial complexes embedded in the Euclidean space $\mathbb{E}^4$, the STAR queries are optimal.

First, we analyze average running times of the STAR query for simplicial 2-complexes, represented by the IS(2D), IG(2D), SIG(2D), IA(2D) and TS data structures. For these data structures, we denote running times required for retrieving the star of p-simplices, with $0 \leq p < 2$, as $S_{IS}^{p,2D}$, $S_{IG}^{p,2D}$, $S_{SIG}^{p,2D}$, $S_{IA}^{p,2D}$, and $S_{TS}^{p,2D}$, respectively. Average running times for the STAR query are summarized in Table 7.3.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$S_{IS}^{1,2D}$</th>
<th>$S_{IS}^{0,2D}$</th>
<th>$S_{SIG}^{1,2D}$</th>
<th>$S_{IS}^{0,1D}$</th>
<th>$S_{IS}^{1,1D}$</th>
<th>$S_{IS}^{0,1D}$</th>
<th>$S_{TS}^{1,2D}$</th>
<th>$S_{TS}^{0,1D}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cone</td>
<td>0.0610</td>
<td>0.0610</td>
<td>0.1092</td>
<td>0.0546</td>
<td>0.0546</td>
<td>0.0350</td>
<td>0.0412</td>
<td>0.0412</td>
</tr>
<tr>
<td>Crumb</td>
<td>0.0580</td>
<td>0.0610</td>
<td>0.0920</td>
<td>0.0449</td>
<td>0.0449</td>
<td>0.0269</td>
<td>0.0344</td>
<td>0.0344</td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>0.0470</td>
<td>0.0500</td>
<td>0.1056</td>
<td>0.0275</td>
<td>0.0275</td>
<td>0.0256</td>
<td>0.0299</td>
<td>0.0299</td>
</tr>
<tr>
<td>Football 1</td>
<td>0.0877</td>
<td>0.0880</td>
<td>0.1664</td>
<td>0.0747</td>
<td>0.0747</td>
<td>0.0374</td>
<td>0.0436</td>
<td>0.0436</td>
</tr>
<tr>
<td>Football 2</td>
<td>0.0780</td>
<td>0.0840</td>
<td>0.1581</td>
<td>0.0720</td>
<td>0.0720</td>
<td>0.0291</td>
<td>0.0406</td>
<td>0.0406</td>
</tr>
<tr>
<td>Torus</td>
<td>0.1592</td>
<td>0.1595</td>
<td>0.3003</td>
<td>0.1033</td>
<td>0.1033</td>
<td>0.2194</td>
<td>0.2261</td>
<td>0.2261</td>
</tr>
<tr>
<td>Armchair</td>
<td>0.1078</td>
<td>0.1086</td>
<td>0.2122</td>
<td>0.0765</td>
<td>0.0772</td>
<td>0.0599</td>
<td>0.0608</td>
<td>0.0678</td>
</tr>
<tr>
<td>800-Cubes</td>
<td>0.1140</td>
<td>0.1245</td>
<td>0.2426</td>
<td>0.1081</td>
<td>0.2194</td>
<td>0.0685</td>
<td>0.0735</td>
<td>0.1654</td>
</tr>
<tr>
<td>Cylinders</td>
<td>0.0770</td>
<td>0.0820</td>
<td>0.1319</td>
<td>0.0440</td>
<td>0.0549</td>
<td>0.0233</td>
<td>0.0294</td>
<td>0.0305</td>
</tr>
<tr>
<td>Pinched-pie</td>
<td>0.0770</td>
<td>0.0820</td>
<td>0.1609</td>
<td>0.0761</td>
<td>0.0848</td>
<td>0.0336</td>
<td>0.0403</td>
<td>0.0497</td>
</tr>
<tr>
<td>Twist</td>
<td>0.0660</td>
<td>0.0690</td>
<td>0.1248</td>
<td>0.0624</td>
<td>0.0632</td>
<td>0.0339</td>
<td>0.0409</td>
<td>0.0446</td>
</tr>
<tr>
<td>Robot</td>
<td>0.0860</td>
<td>0.0880</td>
<td>0.1686</td>
<td>0.0706</td>
<td>0.0733</td>
<td>0.0654</td>
<td>0.0671</td>
<td>0.0715</td>
</tr>
<tr>
<td>Balance</td>
<td>0.0922</td>
<td>0.0969</td>
<td>0.1742</td>
<td>0.0623</td>
<td>0.0624</td>
<td>0.0743</td>
<td>0.0792</td>
<td>0.0792</td>
</tr>
<tr>
<td>Carter</td>
<td>0.0940</td>
<td>0.0958</td>
<td>0.1840</td>
<td>0.1008</td>
<td>0.1018</td>
<td>0.0642</td>
<td>0.0703</td>
<td>0.0750</td>
</tr>
<tr>
<td>Chandelier</td>
<td>0.1477</td>
<td>0.1483</td>
<td>0.2801</td>
<td>0.0926</td>
<td>0.0955</td>
<td>0.0755</td>
<td>0.0815</td>
<td>0.0843</td>
</tr>
<tr>
<td>Frame</td>
<td>0.0567</td>
<td>0.0608</td>
<td>0.1074</td>
<td>0.0557</td>
<td>0.0558</td>
<td>0.0380</td>
<td>0.0389</td>
<td>0.0389</td>
</tr>
<tr>
<td>Tower</td>
<td>0.1503</td>
<td>0.1510</td>
<td>0.2847</td>
<td>0.0986</td>
<td>0.0997</td>
<td>0.0789</td>
<td>0.0839</td>
<td>0.0889</td>
</tr>
<tr>
<td>Tower-wir</td>
<td>0.1372</td>
<td>0.1375</td>
<td>0.2561</td>
<td>0.0952</td>
<td>0.0983</td>
<td>0.1039</td>
<td>0.1126</td>
<td>0.1208</td>
</tr>
</tbody>
</table>

Table 7.3: Average running times for retrieving the star of vertices and edges in digital 2D shapes, represented through the IG(2D) ($S_{IG}^{0,2D}$ and $S_{IG}^{1,2D}$, respectively), IS(2D) ($S_{IS}^{0,2D}$ and $S_{IS}^{1,2D}$, respectively), SIG(2D) ($S_{SIG}^{0,2D}$ and $S_{SIG}^{1,2D}$, respectively), IA(2D) ($S_{IA}^{0,2D}$ and $S_{IA}^{1,2D}$, respectively), and TS ($S_{TS}^{0,2D}$ and $S_{TS}^{1,2D}$, respectively) data structures. Note that $S_{IS}^{1,2D} = S_{IS}^{1,2D} = S_{IS}^{1,2D}$.

We start our analysis from the retrieval of $St(e)$ for an edge $e$ in a simplicial 2-complex $\Sigma$. The IS(2D), IG(2D), and SIG(2D) directly encode co-boundary relation $\mathcal{R}_{1,2}$, thus $S_{IS}^{1,2D} = S_{IG}^{1,2D} = S_{SIG}^{1,2D}$. Conversely, the IA(2D) and TS data structures do not encode co-boundary relation $\mathcal{R}_{1,2}$, except for a non-manifold edge. In this case, the IA(2D) data structure encodes partial relation $\mathcal{R}_{1,2}^*(e)$ for a non-manifold edge $e$ through a single record, while the TS data structure encodes a record for each triangle in $St(e)$. Our tests show that, on average, $S_{IS}^{1,2D}$ is about 6% larger than $S_{IA}^{1,2D}$, since we must traverse the list of records related to a non-manifold edge. However, this overhead depends on the number of triangles incident at a non-manifold edge, and it can become quite large. For instance, $S_{IS}^{1,2D} \approx 2.2 \times S_{IS}^{1,2D}$ in the “800-Cubes” shape, where a very large number of records are required for encoding partial co-boundary relation $\mathcal{R}_{1,2}^*$, as already
noticed in Section 5.4.1. Note that, if there are no non-manifold edges, then the IA∗(2D) and TS data structures are the same, like, for instance, in the “Balance” and “Frame” shapes. Our tests also show that, on average, \( S_{IA^*}^{1,2D} \) is about 13% larger than \( S_{TS}^{1,2D} \). Note that edges are not encoded in the IA∗(2D) data structure, thus this result is really interesting, since we must retrieve all the triangles incident at one vertex of an edge, and select triangles incident at the input edge. This improvement is a consequence of our implicit representation of simplices, specifically of our hierarchies of faces, which allows for a fast retrieval of simplices incident at a vertex (see Section 6.1.3). Retrieving the BOUNDARY query is an extremely fast operation for top simplices, as discussed in Section 7.5.1.

Now, we analyze the retrieval of \( St(v) \) for a vertex \( v \) in a simplicial 2-complex \( \Sigma \). Our tests show that the IA∗(2D) data structure is faster than the other representations. Recall that we retrieve all the top simplices in \( St(v) \) by expanding each 2-cluster in \( St(v) \) through adjacency relation \( R_{2,2} \), which is directly encoded in the IA∗(2D) data structure. Top edges in \( St(v) \) are directly encoded in partial co-boundary relation \( R_{0,1}^* \). Non-top simplices in \( St(v) \) can be efficiently retrieved by our hierarchies of faces (see Section 6.1.3) and Algorithm 6.8 (CLUSTER function). The TS data structure is slower than the IA∗(2D) data structure, due to its encoding of non-manifold adjacency along an edge. Our tests confirm that, on average, \( S_{TS}^{0,2D} \) is about 6% larger than \( S_{IA^*}^{0,2D} \), as just discussed above. In the IS(2D), IG(2D), and SIG(2D) data structures, we visit co-boundary relations, restricted to simplices in \( St(v) \). It is clear that the IG(2D) data structure is more efficient than the IS(2D) and SIG(2D) representations, since it encodes co-boundary relations \( R_{0,1} \) and \( R_{1,2} \). In the IS(2D) data structure, we must retrieve all the top simplices and their faces incident at \( v \) by visiting the IS star-graph \( G_{IS}^v \). Our tests show that, on average, \( S_{IG}^{0,2D} \) is about 4% smaller than \( S_{IS}^{0,2D} \), and \( S_{SIG}^{0,2D} \approx 1.9 \times S_{IS}^{0,2D} \). In any case, these representations are less efficient than the IA∗(2D) and TS data structures. In fact, our tests show that, on average, \( S_{IG}^{2,2D} \) is about 24% larger than \( S_{IS}^{2,2D} \). Note that the bottleneck in these data structures is the BOUNDARY query, which is not as efficient as in the adjacency-based data structures, as demonstrated in Section 7.5.1.

As a consequence, we can summarize our results, regarding the retrieval of the star of a simplex in a simplicial 2-complex:

- \( S_{IA^*}^{0,2D} < S_{TS}^{0,2D} < S_{IG}^{0,2D} < S_{IS}^{0,2D} < S_{SIG}^{0,2D} \), for vertices;
- \( S_{IG}^{1,2D} = S_{SIG}^{1,2D} = S_{IS}^{1,2D} < S_{IA^*}^{1,2D} < S_{TS}^{1,2D} \), for edges.

Now, we analyze average running times of the STAR query for simplicial 3-complexes, represented by the IS(3D), IG(3D), SIG(3D), IA∗(3D) and NMIA data structures. For these data structures, we denote running times required for retrieving the star of \( p \)-simplices, with \( 0 \leq p < 3 \), as \( S_{IS}^{p,3D} \), \( S_{IG}^{p,3D} \), \( S_{SIG}^{p,3D} \), \( S_{IA^*}^{p,3D} \), and \( S_{NMIA}^{p} \), respectively. Table 7.4 summarizes average running times for the IG(3D), IS(3D), and SIG(3D) data structures, while Table 7.5 summarizes average running times for the IA∗(3D) and NMIA data structures.

We start our analysis from the retrieval of \( St(v) \) for a vertex \( v \) in a simplicial 3-complex \( \Sigma \). Our tests show that the IA∗(3D) data structure is faster than the other representations. In this case,
Table 7.4: Average running times for retrieving the star of vertices, edges, and triangles in digital 3D shapes represented through the IS(3D) ($S_{IS}^1$, $S_{IS}^2$, and $S_{IS}^3$, respectively), IG(3D) ($S_{IG}^0$, $S_{IG}^1$, and $S_{IG}^2$, respectively), and SIG(3D) ($S_{SIG}^0$, $S_{SIG}^1$, and $S_{SIG}^2$, respectively) data structures. Note that $S_{IS}^1 = S_{IG}^1 = S_{SIG}^1$.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$S_{IS}^0$</th>
<th>$S_{IS}^1$</th>
<th>$S_{IS}^2$</th>
<th>$S_{IS}^3$</th>
<th>$S_{IG}^0$</th>
<th>$S_{IG}^1$</th>
<th>$S_{IG}^2$</th>
<th>$S_{SIG}^0$</th>
<th>$S_{SIG}^1$</th>
<th>$S_{SIG}^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>0.2185</td>
<td>0.3182</td>
<td>0.5227</td>
<td>0.0909</td>
<td>0.0958</td>
<td>0.1920</td>
<td>0.0151</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cylinder</td>
<td>0.2717</td>
<td>0.3729</td>
<td>0.6134</td>
<td>0.1034</td>
<td>0.1080</td>
<td>0.2157</td>
<td>0.0155</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gargoyle</td>
<td>0.2798</td>
<td>0.3980</td>
<td>0.6628</td>
<td>0.1490</td>
<td>0.1555</td>
<td>0.3034</td>
<td>0.0144</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rings</td>
<td>0.2518</td>
<td>0.3454</td>
<td>0.5710</td>
<td>0.1350</td>
<td>0.1411</td>
<td>0.2683</td>
<td>0.0182</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Torus 3D</td>
<td>0.3392</td>
<td>0.4965</td>
<td>0.7937</td>
<td>0.1583</td>
<td>0.1638</td>
<td>0.3144</td>
<td>0.0161</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arc</td>
<td>0.2154</td>
<td>0.2308</td>
<td>0.3385</td>
<td>0.0536</td>
<td>0.0536</td>
<td>0.1494</td>
<td>0.0141</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balloon</td>
<td>0.1055</td>
<td>0.1317</td>
<td>0.2200</td>
<td>0.0532</td>
<td>0.0549</td>
<td>0.0989</td>
<td>0.0130</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bucket</td>
<td>0.1321</td>
<td>0.1509</td>
<td>0.2642</td>
<td>0.0419</td>
<td>0.0419</td>
<td>0.0958</td>
<td>0.0188</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chime</td>
<td>0.1382</td>
<td>0.1584</td>
<td>0.2439</td>
<td>0.0456</td>
<td>0.0552</td>
<td>0.1405</td>
<td>0.0158</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flasks</td>
<td>0.1965</td>
<td>0.2801</td>
<td>0.4651</td>
<td>0.0888</td>
<td>0.0950</td>
<td>0.1869</td>
<td>0.0220</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Halves</td>
<td>0.2937</td>
<td>0.3225</td>
<td>0.7024</td>
<td>0.0630</td>
<td>0.0646</td>
<td>0.1311</td>
<td>0.0194</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sierpinski</td>
<td>0.4417</td>
<td>0.4607</td>
<td>0.8300</td>
<td>0.0522</td>
<td>0.0527</td>
<td>0.1084</td>
<td>0.0139</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teapot</td>
<td>0.1736</td>
<td>0.2098</td>
<td>0.3712</td>
<td>0.1161</td>
<td>0.1179</td>
<td>0.2154</td>
<td>0.0127</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wheel</td>
<td>0.2463</td>
<td>0.2935</td>
<td>0.4577</td>
<td>0.0612</td>
<td>0.0659</td>
<td>0.1419</td>
<td>0.0238</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.5: Average running times for retrieving the star of vertices, edges, and triangles in digital 3D shapes represented through the IA*(3D) ($S_{IA}^0$, $S_{IA}^1$, and $S_{IA}^2$, respectively) and NMIA ($S_{NM}^0$, $S_{NM}^1$, and $S_{NM}^2$, respectively) data structures. Note that $S_{NM}^2 = S_{IA}^2$.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$S_{IA}^0$</th>
<th>$S_{IA}^1$</th>
<th>$S_{IA}^2$</th>
<th>$S_{NM}^0$</th>
<th>$S_{NM}^1$</th>
<th>$S_{NM}^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>0.3026</td>
<td>0.3026</td>
<td>0.0978</td>
<td>0.0978</td>
<td>0.0150</td>
<td></td>
</tr>
<tr>
<td>Cylinder</td>
<td>0.3394</td>
<td>0.3394</td>
<td>0.1005</td>
<td>0.1005</td>
<td>0.0156</td>
<td></td>
</tr>
<tr>
<td>Gargoyle</td>
<td>0.3446</td>
<td>0.3446</td>
<td>0.1577</td>
<td>0.1577</td>
<td>0.0143</td>
<td></td>
</tr>
<tr>
<td>Rings</td>
<td>0.4163</td>
<td>0.4163</td>
<td>0.1436</td>
<td>0.1436</td>
<td>0.0181</td>
<td></td>
</tr>
<tr>
<td>Torus 3D</td>
<td>0.4856</td>
<td>0.4856</td>
<td>0.1668</td>
<td>0.1668</td>
<td>0.0162</td>
<td></td>
</tr>
<tr>
<td>Arc</td>
<td>0.1992</td>
<td>0.2615</td>
<td>0.0551</td>
<td>0.0651</td>
<td>0.0141</td>
<td></td>
</tr>
<tr>
<td>Balloon</td>
<td>0.1001</td>
<td>0.1740</td>
<td>0.0575</td>
<td>0.0718</td>
<td>0.0131</td>
<td></td>
</tr>
<tr>
<td>Bucket</td>
<td>0.1321</td>
<td>0.1509</td>
<td>0.0479</td>
<td>0.0678</td>
<td>0.0187</td>
<td></td>
</tr>
<tr>
<td>Chime</td>
<td>0.1301</td>
<td>0.1585</td>
<td>0.0576</td>
<td>0.0584</td>
<td>0.0159</td>
<td></td>
</tr>
<tr>
<td>Flasks</td>
<td>0.1437</td>
<td>0.3147</td>
<td>0.1024</td>
<td>0.1065</td>
<td>0.0221</td>
<td></td>
</tr>
<tr>
<td>Halves</td>
<td>0.2738</td>
<td>0.2659</td>
<td>0.0676</td>
<td>0.0699</td>
<td>0.0194</td>
<td></td>
</tr>
<tr>
<td>Sierpinski</td>
<td>0.1099</td>
<td>0.0990</td>
<td>0.0553</td>
<td>0.0573</td>
<td>0.0140</td>
<td></td>
</tr>
<tr>
<td>Teapot</td>
<td>0.1680</td>
<td>0.2309</td>
<td>0.1205</td>
<td>0.1368</td>
<td>0.0129</td>
<td></td>
</tr>
<tr>
<td>Wheel</td>
<td>0.2463</td>
<td>0.3557</td>
<td>0.0669</td>
<td>0.0934</td>
<td>0.0238</td>
<td></td>
</tr>
</tbody>
</table>

the NMIA representation is slower than the IA*(3D) data structure, and our tests show that, on average, $S_{NM}^0$ is about 25% larger than $S_{IA}^3$. The NMIA data structure becomes faster than the IA*(3D) data structure, when the input simplicial 3-complex $\Sigma$ is regular, since none co-boundary relation $R_{1,2}$ is encoded in the IA*(3D) data structure. Recall that the NMIA data structure encodes partial relations $R_{2,3}$ and $R_{3,4}$, which allow for navigation along 1-adjacent top triangles and 3-clusters, like in the “Halves” and “Sierpinski” shapes. In this case, $S_{IA}^3$ is about 8% larger than $S_{NM}^0$. In the IS(3D), IG(3D), and SIG(3D) data structures, we exploit
the same dimension-independent algorithms used in the two-dimensional case, which are based on the traversal of their co-boundary relations, restricted to simplices in $St(v)$. The IG(3D) data structure is more efficient than the IS(3D) and SIG(3D) representations, since it directly encodes co-boundary relations $R_{0,1}$, $R_{1,2}$, and $R_{2,3}$. Our tests show that, on average, $S_{IG}^{0,3D}$ is about 27% smaller than $S_{IS}^{0,3D}$, and $S_{IS}^{0,3D}$ is about 68% smaller than $S_{SIG}^{0,3D}$. In any case, these data structures are less efficient than the IA$^*$ representation, in fact our tests show that, on average, $S_{IG}^{0,3D}$ is about 20% larger than $S_{IA^*}^{0,3D}$. Moreover, our tests show that, on average, $S_{IS}^{0,3D}$ is about 4% smaller than $S_{NM}^{0}$. 

Now, we can analyze the retrieval of $St(e)$ for an edge $e$ in a simplicial 3-complex $\Sigma$. Our tests show that the IG(3D) data structure is faster than the other representations, since it directly encodes co-boundary relations, restricted to simplices in $St(e)$. Our tests show that, on average, $S_{IG}^{1,3D}$ is about 6% smaller than $S_{IS}^{1,3D}$, and $S_{SIG}^{1}(3D) \approx 2 \times S_{IS}^{1,3D}$. In the IA$^*(3D)$ and NMIA data structures, edges are not directly encoded. In any case, partial co-boundary relation $R_{i,cl}$ is encoded in the NMIA data structure, and it allows for the navigation on top triangles and 3-clusters which are 1-adjacent. Thus, it can be exploited for computing the star of an edge. Conversely, in the IA$^*$ data structure, we do not encode any information for these configurations, and we need to navigate on top simplices incident at a vertex of the edge of interest. Our tests show that, on average, $S_{NM}^{1}$ is about 35% smaller than $S_{IA^*}^{1,3D}$. Moreover, $S_{NM}^{1}$ is, on average, only 4% larger than $S_{IS}^{1,3D}$. This result is really interesting, since edges are represented by $\text{GhostSimplexPointer}$ references, and, also in this case, implicit representation of simplices, combined with our precomputed hierarchies of faces (see Section 6.1.3) improves the efficiency of queries. Our tests show that $S_{SIG}^{1,3D} \approx 2 \times S_{IA^*}^{1,3D}$.

Finally, we analyze the retrieval of $St(f)$ for a triangle $f$ in a simplicial 3-complex $\Sigma$. The IS(3D), IG(3D), and SIG(3D) data structures directly encode co-boundary relation $R_{2,3}$, thus $S_{IG}^{2,3D} = S_{SIG}^{2,3D} = S_{IS}^{2,3D}$. Conversely, the IA$^*(3D)$ and NMIA data structures directly encode adjacency relation $R_{3,3}$, and the STAR query can be easily retrieved, since we use $\text{GhostSimplexPointer}$ references. Our tests show that $S_{IS}^{2,3D} \approx S_{IA^*}^{2,3D} \approx S_{NM}^{2,3D}$.

As a consequence, we can summarize our results, regarding the retrieval of the star of a simplex in a simplicial 3-complex:

- $S_{IA^*}^{0,3D} < S_{IG}^{0,3D} < S_{IS}^{0,3D} < S_{NM}^{0} < S_{SIG}^{0,3D}$, for vertices;
- $S_{IG}^{1,3D} < S_{IS}^{1,3D} < S_{NM}^{1} < S_{IA^*}^{1,3D} < S_{SIG}^{1,3D}$, for edges;
- $S_{IG}^{2,3D} = S_{SIG}^{2,3D} = S_{IS}^{2,3D} \approx S_{IA^*}^{2,3D} = S_{NM}^{2,3D}$, for triangles.

7.5.3 Experimental Analysis of Adjacency Retrieval

In this section, we present comparisons regarding execution of the ADJACENCY query for all the data structures in the Mangrove TDS framework. Specifically, we have designed algorithms for retrieving simplices adjacency to a given simplex in the IS and IA$^*$ data structures in Sections 6.2.4
and 6.3.4, respectively. In Section 7.1.2, we have discussed a possible implementation of the ADJACENCY query for the TS data structure. Implementations of the ADJACENCY query in the NMIA, SIG, and IG data structures have been discussed in [DFH03, DFGH04, Huı08] and in Section 4.2.

First, we analyze average running times of the ADJACENCY query for simplicial 2-complexes, discretized by the IS(2D), IG(2D), SIG(2D), IA*(2D) and TS data structures. We denote average running times required for retrieving adjacency relation $R_{p,p}$, for $p$-simplices, with $0 \leq p \leq 2$, as $A^{p2D}_{IS}$, $A^{p2D}_{IG}$, $A^{pD}_{SIG}$, $A^{p2D}_{IA^*}$, and $A^{p}_{TS}$, respectively. Table 7.6 summarizes average running times for the IG(2D), IS(2D), and SIG(2D) data structures, while Table 7.7 summarizes average running times for the IA*(2D) and TS data structures.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$A^{02D}_{IS}$</th>
<th>$A^{02D}_{IG}$</th>
<th>$A^{02D}_{SIG}$</th>
<th>$A^{12D}_{IS}$</th>
<th>$A^{12D}_{IG}$</th>
<th>$A^{12D}_{SIG}$</th>
<th>$A^{22D}_{IS}$</th>
<th>$A^{22D}_{IG}$</th>
<th>$A^{22D}_{SIG}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cone</td>
<td>0.0062</td>
<td>0.0359</td>
<td>0.1373</td>
<td>0.0104</td>
<td>0.1220</td>
<td>0.3323</td>
<td>0.0063</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crumb</td>
<td>0.0096</td>
<td>0.0385</td>
<td>0.1506</td>
<td>0.0075</td>
<td>0.1220</td>
<td>0.3108</td>
<td>0.0066</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>0.0125</td>
<td>0.0500</td>
<td>0.1375</td>
<td>0.0085</td>
<td>0.1000</td>
<td>0.2863</td>
<td>0.0063</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Football 1</td>
<td>0.0089</td>
<td>0.0576</td>
<td>0.1818</td>
<td>0.0076</td>
<td>0.1760</td>
<td>0.3672</td>
<td>0.0081</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Football 2</td>
<td>0.0108</td>
<td>0.0462</td>
<td>0.1688</td>
<td>0.0111</td>
<td>0.1680</td>
<td>0.3463</td>
<td>0.0066</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Torus</td>
<td>0.0071</td>
<td>0.1356</td>
<td>0.3217</td>
<td>0.0087</td>
<td>0.3190</td>
<td>0.6727</td>
<td>0.0065</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Armchair</td>
<td>0.0059</td>
<td>0.0846</td>
<td>0.2209</td>
<td>0.0074</td>
<td>0.2172</td>
<td>0.4599</td>
<td>0.0091</td>
<td></td>
<td></td>
</tr>
<tr>
<td>800-Cubes</td>
<td>0.0077</td>
<td>0.0709</td>
<td>0.2603</td>
<td>0.0102</td>
<td>0.2490</td>
<td>0.5885</td>
<td>0.0063</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cylinders</td>
<td>0.0110</td>
<td>0.0330</td>
<td>0.1099</td>
<td>0.0167</td>
<td>0.1640</td>
<td>0.3767</td>
<td>0.0048</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pinched-pie</td>
<td>0.0057</td>
<td>0.0618</td>
<td>0.2184</td>
<td>0.0161</td>
<td>0.1640</td>
<td>0.4533</td>
<td>0.0085</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Twist</td>
<td>0.0094</td>
<td>0.0427</td>
<td>0.1402</td>
<td>0.0116</td>
<td>0.1380</td>
<td>0.2556</td>
<td>0.0073</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robot</td>
<td>0.0059</td>
<td>0.0629</td>
<td>0.1735</td>
<td>0.0076</td>
<td>0.1760</td>
<td>0.3547</td>
<td>0.0074</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balance</td>
<td>0.0098</td>
<td>0.0692</td>
<td>0.1852</td>
<td>0.0085</td>
<td>0.1938</td>
<td>0.3805</td>
<td>0.0059</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Carter</td>
<td>0.0084</td>
<td>0.0948</td>
<td>0.2449</td>
<td>0.0084</td>
<td>0.1916</td>
<td>0.5125</td>
<td>0.0067</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chandelier</td>
<td>0.0079</td>
<td>0.1240</td>
<td>0.3011</td>
<td>0.0083</td>
<td>0.2966</td>
<td>0.6729</td>
<td>0.0068</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame</td>
<td>0.0111</td>
<td>0.0426</td>
<td>0.1216</td>
<td>0.0153</td>
<td>0.3020</td>
<td>0.1912</td>
<td>0.0060</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tower</td>
<td>0.0068</td>
<td>0.1734</td>
<td>0.3526</td>
<td>0.0087</td>
<td>0.1924</td>
<td>0.6878</td>
<td>0.0060</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tower-wir</td>
<td>0.0064</td>
<td>0.1211</td>
<td>0.2819</td>
<td>0.0085</td>
<td>0.2750</td>
<td>0.5891</td>
<td>0.0071</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.6: Average running times for retrieving adjacency relation of vertices, edges, and triangles in digital 2D shapes represented by the IG(2D) ($A^{02D}_{IG}$, $A^{12D}_{IG}$, and $A^{22D}_{IG}$, respectively), IS(2D) ($A^{02D}_{IS}$, $A^{12D}_{IS}$, and $A^{22D}_{IS}$, respectively), and SIG(2D) ($A^{02D}_{SIG}$, $A^{12D}_{SIG}$, and $A^{22D}_{SIG}$, respectively) data structures. Note that $A^{22D}_{IS} = A^{22D}_{IG} = A^{22D}_{SIG}$.

Now, we analyze the retrieval of adjacency relation $R_{0,v}(v)$ for a vertex $v$ in a simplicial 2-complex. Note that the IG(2D) representation is the most efficient data structure, since it directly encodes co-boundary relation $R_{01}$ and boundary relation $R_{10}$, which allow for a fast and efficient retrieval of adjacency relation $R_{0,v}(v)$. Conversely, in other data structures, we need to retrieve this information. In the IS(2D) and SIG(2D) data structures, adjacency relation $R_{0,v}(v)$ is retrieved by combining relations $R_{0,1}(v)$ (not directly encoded) and $R_{1,0}$, which is directly encoded. According to the running times of the STAR query, presented in Section 7.5.2, it is clear that $A^{02D}_{IG} < A^{02D}_{IS} < A^{02D}_{SIG}$. Our tests shows that, on average, $A^{02D}_{IS} \approx 10 \times A^{02D}_{IG}$, and $A^{02D}_{SIG} \approx 3 \times A^{02D}_{IS}$. In the IA*(2D) and TS data structures, adjacency relation $R_{0,v}(v)$ is retrieved by selecting vertices
A implicit representation of simplices improves the efficiency of queries. In fact, the IA∗ result is quite interesting, since edges are not directly encoded in the IA∗

Note that, although the IA∗ shows that, on average, adjacency relation R above. Specifically, adjacency relation R shows that, on average, (A12D), (A12A∗), and (A22D), respectively, and TS (A1TS, A2TS, and A3TS, respectively) data structures.

Table 7.7: Average running times for retrieving adjacency relation of vertices, edges, and triangles in digital 2D shapes represented by the IA∗(2D) (A12D, A12A∗, and A22D, respectively), and TS (A1TS, A2TS, and A3TS, respectively) data structures.

different than v in all the top simplices in St(v), extracted through the CLUSTER functions (see Algorithm 6.8 and Algorithm 7.2). As discussed in Section 7.5.2, S102D IA∗ S1TS. Our tests confirm that, on average, A1TS is about 6% larger than A1IA∗. Moreover, A102D ≈ 6 × A1IG, and A12TS ≈ 1.5 × A2TS.

Now, we analyze the retrieval of adjacency relation R1,1(e) for an edge e in a simplicial 2-complex. Also in this case, the IG(2D) representation is the most efficient data structure, as just discussed above. Specifically, adjacency relation R1,1(e) is retrieved by combining boundary relation R1,0(e) and co-boundary relation R0,1, which are directly encoded in the IG(2D) data structure. Our tests shows that, on average, A12D ≈ 21 × A1IG, and A12SIG ≈ 2.3 × A1IG. The bottleneck in the IS(2D) and SIG(2D) data structures is given by co-boundary relation R0,1(v), for a vertex v. In fact, we must traverse all the top simplices in the star of v (and thus the complete star of v), although we need only edges in St(v). In the IA∗(2D) and TS data structures, adjacency relation R1,1(e) is equivalent to extract co-boundary relations R0,1 for vertices bounding e. Hence, as discussed in Section 7.5.2, A1IA∗ is 6% smaller than A1TS, A12TS ≈ 1.26 × A1TS, and A11IA∗ ≈ 15.7 × A1IG.

Note that, although the IA∗(2D) data structure is not as efficient as the IG representation, this result is quite interesting, since edges are not directly encoded in the IA∗ data structure, and our implicit representation of simplices improves the efficiency of queries. In fact, the IA∗ offers a better support to adjacency relation R1,1 than the IS(2D) and SIG(2D) data structures.

Finally, we analyze the retrieval adjacency relation R2,2(t) for a triangle t in a simplicial 2-complex. Note that A22TS = A22IG = A22SIG, since these data structures directly encode co-boundary

<table>
<thead>
<tr>
<th>Shape</th>
<th>A102D</th>
<th>A11TS</th>
<th>A112D</th>
<th>A21TS</th>
<th>A212D</th>
<th>A31TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cone</td>
<td>0.0203</td>
<td>0.0203</td>
<td>0.1092</td>
<td>0.1092</td>
<td>0.0094</td>
<td>0.0094</td>
</tr>
<tr>
<td>Crumb</td>
<td>0.0224</td>
<td>0.0224</td>
<td>0.0898</td>
<td>0.0898</td>
<td>0.0065</td>
<td>0.0065</td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>0.0375</td>
<td>0.0375</td>
<td>0.0550</td>
<td>0.0550</td>
<td>0.0064</td>
<td>0.0064</td>
</tr>
<tr>
<td>Football 1</td>
<td>0.0365</td>
<td>0.0365</td>
<td>0.1494</td>
<td>0.1494</td>
<td>0.0081</td>
<td>0.0081</td>
</tr>
<tr>
<td>Football 2</td>
<td>0.0366</td>
<td>0.0366</td>
<td>0.1440</td>
<td>0.1440</td>
<td>0.0065</td>
<td>0.0065</td>
</tr>
<tr>
<td>Torus</td>
<td>0.0718</td>
<td>0.0718</td>
<td>0.2066</td>
<td>0.2066</td>
<td>0.0066</td>
<td>0.0066</td>
</tr>
<tr>
<td>Armchair</td>
<td>0.0463</td>
<td>0.0468</td>
<td>0.1530</td>
<td>0.1544</td>
<td>0.0086</td>
<td>0.0082</td>
</tr>
<tr>
<td>800-Cubes</td>
<td>0.0608</td>
<td>0.1622</td>
<td>0.2162</td>
<td>0.4388</td>
<td>0.0063</td>
<td>0.1227</td>
</tr>
<tr>
<td>Cylinders</td>
<td>0.0330</td>
<td>0.0330</td>
<td>0.0880</td>
<td>0.1098</td>
<td>0.0049</td>
<td>0.0049</td>
</tr>
<tr>
<td>Pinched-pie</td>
<td>0.0618</td>
<td>0.0632</td>
<td>0.1522</td>
<td>0.1696</td>
<td>0.0065</td>
<td>0.0085</td>
</tr>
<tr>
<td>Twist</td>
<td>0.0248</td>
<td>0.0260</td>
<td>0.1248</td>
<td>0.1264</td>
<td>0.0059</td>
<td>0.0072</td>
</tr>
<tr>
<td>Robot</td>
<td>0.0404</td>
<td>0.0448</td>
<td>0.1412</td>
<td>0.1466</td>
<td>0.0074</td>
<td>0.0074</td>
</tr>
<tr>
<td>Balance</td>
<td>0.0577</td>
<td>0.0577</td>
<td>0.1246</td>
<td>0.1248</td>
<td>0.0060</td>
<td>0.0060</td>
</tr>
<tr>
<td>Carter</td>
<td>0.0480</td>
<td>0.0526</td>
<td>0.2016</td>
<td>0.2036</td>
<td>0.0062</td>
<td>0.0068</td>
</tr>
<tr>
<td>Chandelier</td>
<td>0.0655</td>
<td>0.0657</td>
<td>0.1852</td>
<td>0.1910</td>
<td>0.0066</td>
<td>0.0067</td>
</tr>
<tr>
<td>Frame</td>
<td>0.0355</td>
<td>0.0355</td>
<td>0.1114</td>
<td>0.1116</td>
<td>0.0060</td>
<td>0.0060</td>
</tr>
<tr>
<td>Tower</td>
<td>0.0697</td>
<td>0.0724</td>
<td>0.1972</td>
<td>0.1994</td>
<td>0.0054</td>
<td>0.0060</td>
</tr>
<tr>
<td>Tower-wir</td>
<td>0.0602</td>
<td>0.0652</td>
<td>0.1904</td>
<td>0.1966</td>
<td>0.0064</td>
<td>0.0071</td>
</tr>
</tbody>
</table>

174
relation $\mathcal{R}_{1,2}$, and exploit the same BOUNDARY query. Conversely, the $\text{IA}^*(2D)$ and TS data structures encodes adjacency relation $\mathcal{R}_{2,2}(t)$ by partial co-boundary relation $\mathcal{R}_{1,2}$ for each non-manifold edge of $t$. Recall that the TS data structure encodes the same relations as the IA$^*(2D)$ representation, but partial co-boundary relation $\mathcal{R}_{1,2}$, related to a non-manifold edge $e$ of $t$, is replicated for each triangle in $St(e)$. Our tests show that, on average, $A_{IG}^{2,2D} \approx A_{IA}^{2,2D}$, and $A_{IA}^{2,2D}$ is about 6% smaller than $A_{TS}^{2}$. These results are a consequence of our running times, presented in Section 7.5.2.

As a consequence, we can summarize our results, regarding the retrieval of adjacency relation $\mathcal{R}_{p,p}$ for a $p$-simplex, for $0 \leq p \leq 2$, in a simplicial 2-complex:

- $A_{IG}^{0,2D} < A_{IA}^{0,2D} < A_{TS}^{0} < A_{SIG}^{0,2D}$, for vertices;
- $A_{IG}^{1,2D} < A_{IA}^{1,2D} < A_{TS}^{1} < A_{SIG}^{1,2D}$, for edges;
- $A_{IG}^{2,2D} = A_{SIG}^{2,2D} = A_{TS}^{2} \approx A_{IA}^{2,2D} < A_{TS}^{2}$, for triangles.

Now, we analyze average running times of the ADJACENCY query for simplicial 3-complexes, discretized by the IS(3D), IG(3D), SIG(3D), $\text{IA}^*(3D)$ and NMIA data structures. For these data structures, we denote average running times required for retrieving adjacency relation $\mathcal{R}_{p,p}$, for $p$-simplices, with $0 \leq p \leq 3$, as $A_{IS}^{p,3D}$, $A_{IG}^{p,3D}$, $A_{SIG}^{p,3D}$, $A_{IA}^{p,3D}$, and $A_{NM}^{p}$, respectively. Table 7.8 summarizes average running times for the IG(3D), IS(3D), and SIG(3D) data structures, while Table 7.9 summarizes average running times for the $\text{IA}^*(3D)$ and NMIA data structures.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$A_{IG}^{0,3D}$</th>
<th>$A_{IS}^{0,3D}$</th>
<th>$A_{SIG}^{0,3D}$</th>
<th>$A_{IA}^{0,3D}$</th>
<th>$A_{IS}^{1,3D}$</th>
<th>$A_{SIG}^{1,3D}$</th>
<th>$A_{IA}^{1,3D}$</th>
<th>$A_{IS}^{2,3D}$</th>
<th>$A_{SIG}^{2,3D}$</th>
<th>$A_{IA}^{2,3D}$</th>
<th>$A_{IS}^{3,3D}$</th>
<th>$A_{SIG}^{3,3D}$</th>
<th>$A_{IA}^{3,3D}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>0.0099</td>
<td>0.2234</td>
<td>0.5128</td>
<td>0.0112</td>
<td>0.5304</td>
<td>1.0825</td>
<td>0.0077</td>
<td>0.1803</td>
<td>0.6160</td>
<td>0.0062</td>
<td>0.0062</td>
<td>0.0062</td>
<td>0.0062</td>
</tr>
<tr>
<td>Cylinder</td>
<td>0.0091</td>
<td>0.3782</td>
<td>0.8090</td>
<td>0.0134</td>
<td>0.8800</td>
<td>1.6600</td>
<td>0.0089</td>
<td>0.1706</td>
<td>0.7051</td>
<td>0.0045</td>
<td>0.0045</td>
<td>0.0045</td>
<td>0.0045</td>
</tr>
<tr>
<td>Gargoyle</td>
<td>0.0084</td>
<td>0.2911</td>
<td>0.6613</td>
<td>0.0110</td>
<td>0.6818</td>
<td>1.4727</td>
<td>0.0113</td>
<td>0.2789</td>
<td>1.2602</td>
<td>0.0069</td>
<td>0.0069</td>
<td>0.0069</td>
<td>0.0069</td>
</tr>
<tr>
<td>Rings</td>
<td>0.0095</td>
<td>0.2557</td>
<td>0.5722</td>
<td>0.0105</td>
<td>0.5731</td>
<td>1.2283</td>
<td>0.0079</td>
<td>0.1912</td>
<td>0.8715</td>
<td>0.0064</td>
<td>0.0064</td>
<td>0.0064</td>
<td>0.0064</td>
</tr>
<tr>
<td>Torus 3D</td>
<td>0.0070</td>
<td>0.4220</td>
<td>0.7719</td>
<td>0.0144</td>
<td>1.1031</td>
<td>1.9999</td>
<td>0.0086</td>
<td>0.3229</td>
<td>1.3500</td>
<td>0.0067</td>
<td>0.0067</td>
<td>0.0067</td>
<td>0.0067</td>
</tr>
<tr>
<td>Arc</td>
<td>0.0154</td>
<td>0.1385</td>
<td>0.5077</td>
<td>0.0115</td>
<td>0.6322</td>
<td>1.1916</td>
<td>0.0075</td>
<td>0.2090</td>
<td>0.5367</td>
<td>0.0063</td>
<td>0.0063</td>
<td>0.0063</td>
<td>0.0063</td>
</tr>
<tr>
<td>Balloon</td>
<td>0.0072</td>
<td>0.0839</td>
<td>0.2344</td>
<td>0.0102</td>
<td>0.2604</td>
<td>0.5681</td>
<td>0.0054</td>
<td>0.2019</td>
<td>0.3731</td>
<td>0.0064</td>
<td>0.0064</td>
<td>0.0064</td>
<td>0.0064</td>
</tr>
<tr>
<td>Bucket</td>
<td>0.0189</td>
<td>0.0943</td>
<td>0.2615</td>
<td>0.0599</td>
<td>0.4671</td>
<td>0.8623</td>
<td>0.0083</td>
<td>0.2188</td>
<td>0.3938</td>
<td>0.0055</td>
<td>0.0055</td>
<td>0.0055</td>
<td>0.0055</td>
</tr>
<tr>
<td>Chime</td>
<td>0.0081</td>
<td>0.1057</td>
<td>0.2480</td>
<td>0.0120</td>
<td>0.4814</td>
<td>0.7611</td>
<td>0.0065</td>
<td>0.2068</td>
<td>0.4778</td>
<td>0.0060</td>
<td>0.0060</td>
<td>0.0060</td>
<td>0.0060</td>
</tr>
<tr>
<td>Flasks</td>
<td>0.0061</td>
<td>0.0437</td>
<td>0.4674</td>
<td>0.0101</td>
<td>0.4974</td>
<td>1.0408</td>
<td>0.0074</td>
<td>0.3258</td>
<td>0.5947</td>
<td>0.0060</td>
<td>0.0060</td>
<td>0.0060</td>
<td>0.0060</td>
</tr>
<tr>
<td>Halves</td>
<td>0.0119</td>
<td>0.2817</td>
<td>0.6984</td>
<td>0.0154</td>
<td>0.8094</td>
<td>1.6134</td>
<td>0.0084</td>
<td>0.4383</td>
<td>0.6406</td>
<td>0.0051</td>
<td>0.0051</td>
<td>0.0051</td>
<td>0.0051</td>
</tr>
<tr>
<td>Sierpinski</td>
<td>0.0073</td>
<td>0.5054</td>
<td>1.0247</td>
<td>0.0077</td>
<td>0.5024</td>
<td>1.8549</td>
<td>0.0056</td>
<td>0.5898</td>
<td>1.0129</td>
<td>0.0051</td>
<td>0.0051</td>
<td>0.0051</td>
<td>0.0051</td>
</tr>
<tr>
<td>Teapot</td>
<td>0.0109</td>
<td>0.2244</td>
<td>0.5259</td>
<td>0.0100</td>
<td>0.4030</td>
<td>0.9715</td>
<td>0.0073</td>
<td>0.2914</td>
<td>0.8751</td>
<td>0.0062</td>
<td>0.0062</td>
<td>0.0062</td>
<td>0.0062</td>
</tr>
<tr>
<td>Wheel</td>
<td>0.0124</td>
<td>0.1940</td>
<td>0.4577</td>
<td>0.0124</td>
<td>0.5968</td>
<td>1.1300</td>
<td>0.0106</td>
<td>0.2441</td>
<td>0.6459</td>
<td>0.0061</td>
<td>0.0061</td>
<td>0.0061</td>
<td>0.0061</td>
</tr>
</tbody>
</table>

Table 7.8: Average running times for retrieving adjacency relation of vertices, edges, triangles, and tetrahedra in digital 3D shapes represented by the IG(3D) ($A_{IG}^{0,3D}$, $A_{IA}^{0,3D}$, $A_{2,3D}$, and $A_{IG}^{3,3D}$, respectively), IS(3D) ($A_{IS}^{0,3D}$, $A_{IA}^{0,3D}$, $A_{IS}^{2,3D}$, and $A_{IA}^{3,3D}$, respectively), and SIG(3D) ($A_{SIG}^{0,3D}$, $A_{IA}^{0,3D}$, $A_{SIG}^{2,3D}$, and $A_{IA}^{3,3D}$, respectively) data structures. Note that $A_{IA}^{3,3D} = A_{IG}^{3,3D} = A_{SIG}^{3,3D}$.

Now, we analyze the retrieval of adjacency relation $\mathcal{R}_{0,0}(v)$, for a vertex $v$, in a simplicial 3-complex. The IG(3D) representation is the most efficient data structure, since it directly encodes
co-boundary relation $R_{0,1}$ and boundary relation $R_{1,0}$, which allow for a fast and efficient retrieval of adjacency relation $R_{0,0}(v)$. Conversely, in other data structures, we need to retrieve this information. According to the running times of the STAR query, presented in Section 7.5.2, it is clear that $A_{1G}^{0,3D} < A_{1S}^{0,3D} < A_{1SIG}^{0,3D}$. Our tests shows that, on average, $A_{1S}^{0,3D} \approx 26 \times A_{1G}^{0,3D}$ and $A_{1SIG}^{0,3D} \approx 3 \times A_{1S}^{0,3D}$. In the IA$^*$ (3D) data structure, we exploit the same algorithm used in the two-dimensional case, while, in the NMIA data structure, adjacency relation $R_{0,0}(v)$ for a vertex $v$, is retrieved by removing $v$ from vertices bounding edges in $St(v)$, as discussed in Section 5.3. Our tests show that, on average, $A_{3NM}^{0} \approx 2.4 \times A_{1IA}^{1}$. Moreover, $A_{1IA}^{1} \approx 5.8 \times A_{1G}^{0,3D}$, and $A_{1S}^{0,3D} \approx 1.97 \times A_{3NM}^{0}$. Although the IA$^*$ (3D) data structure is not efficient as the IG(3D) representation, this result is interesting, since it proves that our implicit representations of simplices, combined with the IA$^*$ data structure, improves efficiency of queries for local mangroves.

Now, we analyze the retrieval of adjacency relation $R_{1,1}(e)$ for an edge $e = (v_0, v_1)$ in a simplicial 3-complex. Also in this case, the IG(3D) representation is the most efficient data structure, as discussed in the two-dimensional case. Specifically, adjacency relation $R_{1,1}(e)$ is retrieved by combining boundary relation $R_{1,0}$ and co-boundary relation $R_{0,1}$, which are directly encoded in the IG(3D) data structure. Our tests shows that, on average, $A_{1IG}^{1,3D} \approx 0.5 \times A_{1IS}^{1,3D}$, and $A_{1SIG}^{1,3D} \approx 2.1 \times A_{1IS}^{1,3D}$. Also in this case, the bottleneck in the IS(3D) and SIG(3D) data structures is given by co-boundary relation $R_{0,1}$. In the IA$^*$ (3D) and NMIA data structures, adjacency relation $R_{1,1}(e)$ is equivalent to extract co-boundary relations $R_{0,1}$ for vertices bounding edge $e$. Our tests show that, on average, $A_{1NM}^{1} \approx 1.4 \times A_{1IA}^{1}, A_{1S}^{1,3D} \approx 1.5 \times A_{1NM}^{1},$ and $A_{1IA}^{1,3D} \approx 22 \times A_{1IG}^{0,3D}$.

Now, we analyze the retrieval of adjacency relation $R_{2,2}(f)$ for a triangle $f$ in a simplicial 3-complex. Also in this case, the IG(3D) representation is the most efficient data structure, since boundary relation $R_{2,1}$ and co-boundary relation $R_{1,2}$ are directly encoded. Also in this case, the

<table>
<thead>
<tr>
<th>Shape</th>
<th>$A_{IA}^{3}$</th>
<th>$A_{IA}^{3}$</th>
<th>$A_{IA}^{3}$</th>
<th>$A_{IA}^{3}$</th>
<th>$A_{IA}^{3}$</th>
<th>$A_{IA}^{3}$</th>
<th>$A_{IA}^{3}$</th>
<th>$A_{IA}^{3}$</th>
<th>$A_{IA}^{3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>0.0371</td>
<td>0.0371</td>
<td>0.2475</td>
<td>0.2475</td>
<td>0.0785</td>
<td>0.0785</td>
<td>0.0060</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cylinder</td>
<td>0.0540</td>
<td>0.0540</td>
<td>0.4152</td>
<td>0.4152</td>
<td>0.0266</td>
<td>0.0266</td>
<td>0.0046</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gargoyle</td>
<td>0.0527</td>
<td>0.0527</td>
<td>0.3085</td>
<td>0.3085</td>
<td>0.0157</td>
<td>0.0157</td>
<td>0.0067</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rings</td>
<td>0.0726</td>
<td>0.0726</td>
<td>0.3699</td>
<td>0.3699</td>
<td>0.0887</td>
<td>0.0887</td>
<td>0.0064</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Torus 3D</td>
<td>0.0760</td>
<td>0.0760</td>
<td>0.4558</td>
<td>0.4558</td>
<td>0.0161</td>
<td>0.0161</td>
<td>0.0067</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arc</td>
<td>0.0462</td>
<td>0.0923</td>
<td>0.2337</td>
<td>0.2587</td>
<td>0.1497</td>
<td>0.1949</td>
<td>0.0064</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balloon</td>
<td>0.0361</td>
<td>0.0379</td>
<td>0.1733</td>
<td>0.1748</td>
<td>0.0921</td>
<td>0.1114</td>
<td>0.0063</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bucket</td>
<td>0.0566</td>
<td>0.0566</td>
<td>0.2036</td>
<td>0.4311</td>
<td>0.1188</td>
<td>0.1250</td>
<td>0.0055</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chime</td>
<td>0.0325</td>
<td>0.0650</td>
<td>0.1705</td>
<td>0.4418</td>
<td>0.1319</td>
<td>0.1466</td>
<td>0.0059</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flasks</td>
<td>0.0437</td>
<td>0.1358</td>
<td>0.3143</td>
<td>0.5004</td>
<td>0.1367</td>
<td>0.2292</td>
<td>0.0061</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Halves</td>
<td>0.097</td>
<td>0.2024</td>
<td>0.3397</td>
<td>0.6810</td>
<td>0.2475</td>
<td>0.3520</td>
<td>0.0051</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sierpinski</td>
<td>0.0488</td>
<td>0.0919</td>
<td>0.1570</td>
<td>0.2291</td>
<td>0.0430</td>
<td>0.3134</td>
<td>0.0051</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teapot</td>
<td>0.0768</td>
<td>0.1182</td>
<td>0.4320</td>
<td>0.4423</td>
<td>0.1884</td>
<td>0.2914</td>
<td>0.0062</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wheel</td>
<td>0.0547</td>
<td>0.1592</td>
<td>0.3473</td>
<td>0.3550</td>
<td>0.1672</td>
<td>0.1675</td>
<td>0.0061</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.9: Average running times for retrieving adjacency relation of vertices, edges, triangles, and tetrahedra in digital 3D shapes represented by the IA$^*$ (3D) ($A_{IA}^{0,3D}, A_{IA}^{1,3D}, A_{IA}^{3,3D},$ respectively), and NMIA ($A_{IA}^{0,3D}, A_{IA}^{1,3D}, A_{IA}^{3,3D},$ respectively) data structures.
bottleneck of IS(3D) and SIG(3D) data structures consists of co-boundary relation $R_{1,2}$, since we must traverse all the top simplices in the star of an edge, in order to retrieve only edges. Note that we exploit the same BOUNDARY query in the IG(3D), IS(3D), and SIG(3D) data structures. Our tests shows that, on average, $A_{IG}^{2,3D} \approx 36.9 \times A_{IG}^{1,3D}$, and $A_{SIG}^{2,3D} \approx 2.86 \times A_{SIG}^{1,3D}$. In the IA* (3D) data structure, we directly encode top triangles adjacent to a top triangle $f$ by partial adjacency relation $R_{2,2}^p(f)$, while the remaining triangles adjacent to $f$ are retrieved by combining boundary relation $R_{2,1}$, and co-boundary relation $R_{1,2}$. If $f$ is a non-top triangle, then we retrieve adjacency relation $R_{2,2}^p(f)$ by relations $R_{4,1}$ and $R_{1,2}$. The NMIA data structure is more efficient than the IA* (3D) representation, since we encode relation $R_{2,cl}^p(f)$, and adjacency relation $R_{3,3}$. As a consequence, we can retrieve top triangles adjacent to $f$, and expand 3-clusters which share an edge with $f$. Our tests show that, on average, $A_{IA*}^{2,3D} \approx 1.5 \times A_{NM}^{3,3D}$, and $A_{NM}^{3,3D} \approx 14 \times A_{IG}^{2,3D}$. This result is really interesting, since triangles are represented by GhostSimplexPointer references, and, also in this case, our implicit representation of simplices, combined with our hierarchies of faces (see Section 6.1.3) improves the efficiency of queries. Our tests show that $A_{IS}^{2,3D} \approx 4.4 \times A_{IA*}^{2,3D}$.

Finally, we analyze the retrieval of of adjacency relation $R_{3,3}(t)$ for a tetrahedron $t$ in a simplicial 3-complex. Note that $A_{IS}^{4,3D} = A_{IG}^{3,3D} = A_{SIG}^{3,3D}$, since these data structures directly encode co-boundary relation $R_{2,3}$, and exploit the same BOUNDARY query. Moreover, $A_{IA*}^{4,3D} = A_{NM}^{3,3D}$, since both the IA* (3D) and NMIA data structures directly encode adjacency relation $R_{3,3}$. Our tests show that $A_{IS}^{3,3D} \approx A_{IA*}^{3,3D}$.

As a consequence, we can summarize our results, regarding the retrieval of adjacency relation $R_{p,p}$ for a $p$-simplex, for $0 \leq p \leq 3$, in a simplicial 3-complex:

- $A_{IG}^{0,3D} < A_{IA*}^{0,3D} < A_{NM}^{0,3D} < A_{IS}^{0,3D} < A_{SIG}^{0,3D}$, for vertices;
- $A_{IG}^{1,3D} < A_{IA*}^{1,3D} < A_{NM}^{1,3D} < A_{IS}^{1,3D} < A_{SIG}^{1,3D}$, for edges;
- $A_{IG}^{2,3D} < A_{NM}^{2,3D} < A_{IA*}^{2,3D} < A_{IS}^{2,3D} < A_{SIG}^{2,3D}$, for triangles;
- $A_{IS}^{3,3D} = A_{IG}^{3,3D} = A_{SIG}^{3,3D} \approx A_{IA*}^{3,3D} = A_{NM}^{3,3D}$, for tetrahedra.

### 7.5.4 Experimental Analysis of Link Retrieval

In this section, we provide comparisons regarding execution of the LINK query for all the data structures implemented in the Mangrove TDS framework. Specifically, we have designed algorithms for retrieving the link of a simplex in the IS and IA* data structures in Section 6.2.5 and 6.3.5, respectively. We have discussed algorithms for retrieving the link of a simplex in the TS and NMIA data structures in Sections 7.1.2 and 7.2.2, respectively. Finally, in the SIG and IG data structures, we can exploit brief variants of the LINK query implemented in the IS data structure.

Recall that the link $Lk(\sigma)$ of a $p$-simplex $\sigma$ is given by all the simplices, which bound simplices in $St(\sigma)$, but are not incident at $\sigma$. The link $Lk(\sigma)$ is formed by opposite faces (and their boundary)
of σ with respect to each top simplex in St(σ). Thus, the LINK query can be expressed in terms of the BOUNDARY and STAR queries, discussed in Sections 7.5.1 and 7.5.2.

First, we analyze average running times of the LINK query for simplicial 2-complexes, discretized by the IG(2D), IS(2D), SIG(2D), IA*(2D) and TS data structures. For these data structures, we denote average running times required for retrieving the link of p-vertices, with 0 ≤ p < 2, as \( L_{IS}^{p,2D}, L_{IG}^{p,2D}, L_{SIG}^{p,2D}, L_{IA^*}^{p,2D}, L_{IA^*}^{p,2D}, \) and \( L_{TS}^{p} \), respectively. Average running times for the LINK query on simplicial 2-complexes are summarized in Table 7.10.

<table>
<thead>
<tr>
<th>Shape</th>
<th>( L_{IG}^{0,2D} )</th>
<th>( L_{IS}^{0,2D} )</th>
<th>( L_{IS}^{0,2D} )</th>
<th>( L_{IA^*}^{0,2D} )</th>
<th>( L_{TS}^{0} )</th>
<th>( L_{IG}^{1,2D} )</th>
<th>( L_{IS}^{1,2D} )</th>
<th>( L_{TS}^{1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cone</td>
<td>0.1310</td>
<td>0.1362</td>
<td>0.2587</td>
<td>0.0905</td>
<td>0.0905</td>
<td>0.0840</td>
<td>0.0616</td>
<td>0.0616</td>
</tr>
<tr>
<td>Crumb</td>
<td>0.1378</td>
<td>0.1433</td>
<td>0.2723</td>
<td>0.0929</td>
<td>0.0929</td>
<td>0.0667</td>
<td>0.0419</td>
<td>0.0419</td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>0.1000</td>
<td>0.1040</td>
<td>0.1976</td>
<td>0.0875</td>
<td>0.0875</td>
<td>0.0598</td>
<td>0.0427</td>
<td>0.0427</td>
</tr>
<tr>
<td>Football 1</td>
<td>0.1729</td>
<td>0.1798</td>
<td>0.3416</td>
<td>0.1031</td>
<td>0.1031</td>
<td>0.1024</td>
<td>0.0667</td>
<td>0.0667</td>
</tr>
<tr>
<td>Football 2</td>
<td>0.1613</td>
<td>0.1693</td>
<td>0.3217</td>
<td>0.0978</td>
<td>0.0978</td>
<td>0.0938</td>
<td>0.0603</td>
<td>0.0603</td>
</tr>
<tr>
<td>Torus</td>
<td>0.3178</td>
<td>0.3305</td>
<td>0.6279</td>
<td>0.1177</td>
<td>0.1177</td>
<td>0.1973</td>
<td>0.0923</td>
<td>0.0923</td>
</tr>
<tr>
<td>Armchair</td>
<td>0.2192</td>
<td>0.2279</td>
<td>0.4331</td>
<td>0.0905</td>
<td>0.0968</td>
<td>0.1808</td>
<td>0.0663</td>
<td>0.0701</td>
</tr>
<tr>
<td>800-Cubes</td>
<td>0.2376</td>
<td>0.2471</td>
<td>0.4695</td>
<td>0.2040</td>
<td>0.3453</td>
<td>0.1399</td>
<td>0.1206</td>
<td>0.2626</td>
</tr>
<tr>
<td>Cylinders</td>
<td>0.1538</td>
<td>0.1599</td>
<td>0.3038</td>
<td>0.0769</td>
<td>0.0815</td>
<td>0.0600</td>
<td>0.0467</td>
<td>0.0500</td>
</tr>
<tr>
<td>Pinched-pie</td>
<td>0.1609</td>
<td>0.1674</td>
<td>0.3181</td>
<td>0.1509</td>
<td>0.1882</td>
<td>0.0709</td>
<td>0.0608</td>
<td>0.0850</td>
</tr>
<tr>
<td>Twist</td>
<td>0.1299</td>
<td>0.1366</td>
<td>0.2595</td>
<td>0.0567</td>
<td>0.0601</td>
<td>0.0799</td>
<td>0.0488</td>
<td>0.0517</td>
</tr>
<tr>
<td>Robot</td>
<td>0.1689</td>
<td>0.1758</td>
<td>0.3340</td>
<td>0.0835</td>
<td>0.0885</td>
<td>0.1250</td>
<td>0.0604</td>
<td>0.0643</td>
</tr>
<tr>
<td>Balance</td>
<td>0.1811</td>
<td>0.1883</td>
<td>0.3578</td>
<td>0.1131</td>
<td>0.1131</td>
<td>0.1404</td>
<td>0.0651</td>
<td>0.0651</td>
</tr>
<tr>
<td>Carter</td>
<td>0.2375</td>
<td>0.2470</td>
<td>0.4693</td>
<td>0.0958</td>
<td>0.1012</td>
<td>0.2756</td>
<td>0.0714</td>
<td>0.0757</td>
</tr>
<tr>
<td>Chandelier</td>
<td>0.2946</td>
<td>0.3064</td>
<td>0.5822</td>
<td>0.1124</td>
<td>0.1196</td>
<td>0.2657</td>
<td>0.0858</td>
<td>0.0909</td>
</tr>
<tr>
<td>Frame</td>
<td>0.0821</td>
<td>0.0854</td>
<td>0.1623</td>
<td>0.0648</td>
<td>0.0648</td>
<td>0.0778</td>
<td>0.0333</td>
<td>0.0333</td>
</tr>
<tr>
<td>Tower</td>
<td>0.3819</td>
<td>0.3972</td>
<td>0.7545</td>
<td>0.1186</td>
<td>0.1257</td>
<td>0.2680</td>
<td>0.0873</td>
<td>0.0925</td>
</tr>
<tr>
<td>Tower-wir</td>
<td>0.2785</td>
<td>0.2896</td>
<td>0.5499</td>
<td>0.1112</td>
<td>0.1178</td>
<td>0.2418</td>
<td>0.0837</td>
<td>0.0887</td>
</tr>
</tbody>
</table>

Table 7.10: Average running times for retrieving the link of vertices and edges in digital 2D shapes, represented through the IG(2D) (\( L_{IG}^{0,2D} \) and \( L_{IG}^{1,2D} \), respectively), IS(2D) (\( L_{IS}^{0,2D} \) and \( L_{IS}^{1,2D} \), respectively), SIG(2D) (\( L_{SIG}^{0,2D} \) and \( L_{SIG}^{1,2D} \), respectively), IA*(2D) (\( L_{IA^*}^{0,2D} \) and \( L_{IA^*}^{1,2D} \), respectively), and TS (\( L_{TS}^{0} \) and \( L_{TS}^{1} \), respectively) data structures. Note that \( L_{IS}^{1} = L_{IG}^{1} = L_{SIG}^{1} \).

We start our analysis from the retrieval of Lk(e) for an edge e in a simplicial 2-complex Σ. The IS(2D), IG(2D), and SIG(2D) data structures do not encode co-boundary relation \( R_{1,2} \), and they exploit the same BOUNDARY query. Thus, \( L_{IS}^{1,2D} = L_{IG}^{1,2D} = L_{SIG}^{1,2D} \). Conversely, the IA*(2D) and TS data structures do not encode co-boundary relation \( R_{1,2} \), except for a non-manifold edge. Recall that the IA*(2D) data structure encodes partial relation \( R_{1,2}(e) \) for a non-manifold edge e through a single record, while the TS data structure encodes a record for each triangle in St(e). Our tests show that, on average, \( L_{TS}^{1} \) is about 6% larger than \( L_{IA^*}^{1,2D} \), since we must traverse the list of records related to a non-manifold edge, and extract, for each triangle t in St(e), opposite vertices of e with respect to t. Also in this case, this overhead depends on the number of triangles incident at a non-manifold edge, and it can become quite large: for instance, \( L_{TS}^{1} \approx 1.88 \times L_{IG}^{1,2D} \) and \( L_{TS}^{1} \approx 1.2 \times L_{IS}^{1,2D} \) in the “800-Cubes” and “Pinched-pie” shapes, respectively. If there are no non-manifold edges, then the IA*(2D) and TS data structures are the same, like, for instance,
in the “Balance” and “Frame” shapes. Our tests also show that, on average, $L_{1S}^{1,2D} \approx 2 \times L_{1S}^T$.

This improvement is a consequence of our implicit representation of simplices, which allows for a fast execution of the BOUNDARY query for local mangroves, as discussed in Section 7.5.1. In fact, as shown in Section 7.5.1, $S_{1S}^T > S_{1S}^{1,2D}$. However, in the IS(2D) data structure, we retrieve combinatorial boundary of each triangle $t$ in $St(e)$ in order to extract a vertex of $t$ in the link of $e$, as performed in Algorithm 6.4. Note that the BOUNDARY query in the IA$^*$ data structure is about 34% faster than in the IS representation, if applied on triangles, as discussed in Section 7.5.1.

Now, we analyze the retrieval of $Lk(v)$ for a vertex $v$ in a simplicial 2-complex $\Sigma$. Our tests show that the IA$^*(2D)$ data structure is faster than the other representations. Recall that we retrieve all the top simplices in $St(v)$ by Algorithm 6.8 (CLUSTER function), and then we retrieve the opposite face (and its boundary) of $v$ for each top simplex in $St(v)$. The TS data structure is slower than the IA$^*(2D)$ data structure, due to its encoding of non-manifold adjacency along an edge. Our tests confirm that, on average, $L_{TS}^0$ is about 6% larger than $L_{IA^*}^0$, as just discussed above. In the IS(2D), IG(2D), and SIG(2D) data structures, we retrieve simplices in the link of $v$ through variants of Algorithm 6.4. The key point is efficiency of the STAR query. It is clear that the IG(2D) data structure is more efficient than the IS(2D) and SIG(2D) representations, since it encodes co-boundary relations $R_{0,1}$ and $R_{1,2}$. Our tests show that, on average, $L_{IG}^0$ is about 4% smaller than $L_{IS}^0$, and $L_{SIG}^0 \approx 1.9 \times L_{IS}^0$. In any case, these representations are less efficient than the IA$^*(2D)$ and TS data structures. In fact, our tests show that, on average, $L_{IG}^{0,2D} \approx 1.8 \times L_{TS}^0$. Note that the bottleneck in these data structures is the BOUNDARY query, which is not as efficient as in the adjacency-based data structures, as demonstrated in Section 7.5.1.

As a consequence, we can summarize our results, regarding the retrieval of the link $Lk(\sigma)$ for a $p$-simplex $\sigma$, with $0 \leq p < 2$, in a simplicial 2-complex:

- $L_{IA^*}^0 < L_{IS}^0 < L_{IG}^0 < L_{IS}^{0,2D} < L_{SIG}^0$, for vertices;
- $L_{IA^*}^{0,2D} < L_{TS}^1 < L_{IG}^{1,2D} = L_{IS}^{1,2D} = L_{SIG}^{1,2D}$, for edges.

Now, we analyze average running times of the LINK query for simplicial 3-complexes, discretized by the IG(3D), IS(3D), SIG(3D), IA$^*(3D)$ and NMIA data structures. For these data structures, we denote average running times required for retrieving the link of $p$-vertices, with $0 \leq p < 3$, as $L_{IS}^{p,3D}, L_{IG}^{p,3D}, L_{SIG}^{p,3D}, L_{IA^*}^{p,3D}, L_{NM}^{p}$, respectively. Table 7.11 summarizes average running times for the IG(3D), IS(3D), and SIG(3D) data structures, while Table 7.12 summarizes average running times for the IA$^*(3D)$ and NMIA data structures.

We start our analysis from the retrieval of $Lk(v)$ for a vertex $v$ in a simplicial 3-complex $\Sigma$. Our tests show that the IA$^*(3D)$ and NMIA data structures are faster than all the other representations. Specifically, we retrieve top simplices incident at $v$ through CLUSTER functions (see Algorithm 6.8 and Algorithm 7.2). Then, we extract, for each top $p$-simplex $\sigma$ in $St(v)$, the opposite $(p - 1)$-face of $v$ with respect to $\sigma$, as discussed in Section 6.3.5. These faces can be easily retrieved by their GhostSimplexPointer references. As discussed in Section 7.5.2, in the NMIA representation, the STAR query is slower than in the IA$^*(3D)$ data structure. Our tests show that, on average, $L_{NM}^0$ is
about 68% larger than \( \mathcal{L}_{IS}^{0,3D} \) and \( \mathcal{L}_{SIG}^{0,3D} \). Clearly, the NMIA data structure becomes faster than the IA*(3D) data structure, if the input simplicial 3-complex \( \Sigma \) is regular, as discussed in Section 7.5.2. For 3D shapes represented through the IA*(3D) \( \mathcal{L}_{IA^*}^{0,3D} \), IG(3D) \( \mathcal{L}_{IG}^{0,3D} \), and SIG(3D) \( \mathcal{L}_{SIG}^{0,3D} \), respectively), and NMIA \( \mathcal{L}_{NM}^{0,3D} \), \( \mathcal{L}_{NM}^{2,3D} = \mathcal{L}_{IA^*}^{2,3D} \), respectively) data structures. Note that \( \mathcal{L}_{IA}^{2,3D} = \mathcal{L}_{IG}^{2,3D} = \mathcal{L}_{SIG}^{2,3D} \) and \( \mathcal{L}_{NM}^{2,3D} = \mathcal{L}_{IA^*}^{2,3D} \).

<table>
<thead>
<tr>
<th>Shape</th>
<th>( \mathcal{L}_{IS}^{0,3D} )</th>
<th>( \mathcal{L}_{IA}^{0,3D} )</th>
<th>( \mathcal{L}_{NM}^{0,3D} )</th>
<th>( \mathcal{L}_{IS}^{1,3D} )</th>
<th>( \mathcal{L}_{IA}^{1,3D} )</th>
<th>( \mathcal{L}_{NM}^{1,3D} )</th>
<th>( \mathcal{L}_{IS}^{2,3D} )</th>
<th>( \mathcal{L}_{IA}^{2,3D} )</th>
<th>( \mathcal{L}_{NM}^{2,3D} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>0.5416</td>
<td>0.6878</td>
<td>1.1555</td>
<td>0.2130</td>
<td>0.2258</td>
<td>0.4516</td>
<td>0.0157</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cylinder</td>
<td>0.8645</td>
<td>1.0979</td>
<td>1.8554</td>
<td>0.2459</td>
<td>0.2606</td>
<td>0.5473</td>
<td>0.0192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gargoyle</td>
<td>0.6726</td>
<td>0.8542</td>
<td>1.4350</td>
<td>0.3281</td>
<td>0.3480</td>
<td>0.6612</td>
<td>0.0226</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rings</td>
<td>0.5841</td>
<td>0.7418</td>
<td>1.2462</td>
<td>0.2895</td>
<td>0.3069</td>
<td>0.5831</td>
<td>0.0219</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Torus 3D</td>
<td>1.0844</td>
<td>1.3772</td>
<td>2.3147</td>
<td>0.3377</td>
<td>0.3580</td>
<td>0.7160</td>
<td>0.0171</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arc</td>
<td>0.4923</td>
<td>0.6252</td>
<td>1.0441</td>
<td>0.1877</td>
<td>0.1989</td>
<td>0.4376</td>
<td>0.0197</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balloon</td>
<td>0.2263</td>
<td>0.2829</td>
<td>0.4753</td>
<td>0.1669</td>
<td>0.1769</td>
<td>0.3520</td>
<td>0.0112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bucket</td>
<td>0.2642</td>
<td>0.3359</td>
<td>0.5576</td>
<td>0.1198</td>
<td>0.1377</td>
<td>0.2754</td>
<td>0.0187</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chime</td>
<td>0.3415</td>
<td>0.4303</td>
<td>0.7229</td>
<td>0.1273</td>
<td>0.1339</td>
<td>0.3078</td>
<td>0.0223</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flasks</td>
<td>0.4774</td>
<td>0.5991</td>
<td>1.0065</td>
<td>0.2095</td>
<td>0.2221</td>
<td>0.4220</td>
<td>0.0208</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Halves</td>
<td>0.6984</td>
<td>0.8800</td>
<td>1.4784</td>
<td>0.2291</td>
<td>0.2429</td>
<td>0.4858</td>
<td>0.0233</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sierpinski</td>
<td>0.9255</td>
<td>1.1726</td>
<td>1.9699</td>
<td>0.9234</td>
<td>0.9788</td>
<td>1.9576</td>
<td>0.0148</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teapot</td>
<td>0.5319</td>
<td>0.6702</td>
<td>1.1259</td>
<td>0.2371</td>
<td>0.2395</td>
<td>0.5029</td>
<td>0.0144</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wheel</td>
<td>0.4776</td>
<td>0.6018</td>
<td>1.0110</td>
<td>0.1586</td>
<td>0.1629</td>
<td>0.3421</td>
<td>0.0129</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shape</th>
<th>( \mathcal{L}_{IS}^{0,3D} )</th>
<th>( \mathcal{L}_{IA}^{0,3D} )</th>
<th>( \mathcal{L}_{NM}^{0,3D} )</th>
<th>( \mathcal{L}_{IS}^{1,3D} )</th>
<th>( \mathcal{L}_{IA}^{1,3D} )</th>
<th>( \mathcal{L}_{NM}^{1,3D} )</th>
<th>( \mathcal{L}_{IS}^{2,3D} )</th>
<th>( \mathcal{L}_{IA}^{2,3D} )</th>
<th>( \mathcal{L}_{NM}^{2,3D} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>0.2943</td>
<td>0.2943</td>
<td>0.0601</td>
<td>0.0601</td>
<td>0.0158</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cylinder</td>
<td>0.3721</td>
<td>0.3721</td>
<td>0.0891</td>
<td>0.0891</td>
<td>0.0190</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gargoyle</td>
<td>0.3786</td>
<td>0.3786</td>
<td>0.0862</td>
<td>0.0862</td>
<td>0.0228</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rings</td>
<td>0.4215</td>
<td>0.4215</td>
<td>0.0834</td>
<td>0.0834</td>
<td>0.0218</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Torus 3D</td>
<td>0.5236</td>
<td>0.5236</td>
<td>0.0684</td>
<td>0.0684</td>
<td>0.0171</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arc</td>
<td>0.2308</td>
<td>0.2885</td>
<td>0.0351</td>
<td>0.0351</td>
<td>0.0195</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balloon</td>
<td>0.1235</td>
<td>0.1544</td>
<td>0.0345</td>
<td>0.0466</td>
<td>0.0111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bucket</td>
<td>0.1132</td>
<td>0.1415</td>
<td>0.0299</td>
<td>0.0404</td>
<td>0.0188</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chime</td>
<td>0.1585</td>
<td>0.1997</td>
<td>0.0564</td>
<td>0.0672</td>
<td>0.0222</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flasks</td>
<td>0.3837</td>
<td>0.4835</td>
<td>0.0790</td>
<td>0.1066</td>
<td>0.0208</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Halves</td>
<td>0.4206</td>
<td>0.3869</td>
<td>0.0945</td>
<td>0.1414</td>
<td>0.0231</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sierpinski</td>
<td>0.0926</td>
<td>0.0852</td>
<td>0.0163</td>
<td>0.1183</td>
<td>0.0149</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teapot</td>
<td>0.1734</td>
<td>0.0369</td>
<td>0.0906</td>
<td>0.1223</td>
<td>0.0142</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wheel</td>
<td>0.2239</td>
<td>0.2776</td>
<td>0.0559</td>
<td>0.0755</td>
<td>0.0128</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.11: Average running times for retrieving the link of vertices, edges, and triangles in digital 3D shapes represented through the IS(3D) \( \mathcal{L}_{IS}^{0,3D} \), \( \mathcal{L}_{IS}^{1,3D} \), and \( \mathcal{L}_{IS}^{2,3D} \) data structures. Note that \( \mathcal{L}_{IS}^{2,3D} = \mathcal{L}_{IG}^{2,3D} = \mathcal{L}_{SIG}^{2,3D} \).
any case, these data structures are less efficient than the IA\(^{*}\)(3D) representation, in fact our tests show that, on average, \( L_{IG}^{0.3D} \approx 2.5 \times L_{NM}^0 \). Note that the bottleneck in these data structures is the BOUNDARY query, which is not as efficient as in the adjacency-based data structures, as demonstrated in Section 7.5.1.

Now, we analyze the retrieval of \( Lk(e) \) for an edge \( e \) in a simplicial 3-complex \( \Sigma \). Our tests show that the NMIA data structure is faster than the other representations, since it directly encodes relation \( R_{2,cl} \) and \( R_{3,cl} \) through partial relation \( R_{1,cl}^+ \). This latter relation allows for the navigation on top triangles and 3-clusters which are 1-adjacent. The IA\(^{*}\)(3D) data structure does not encode this relation, and we navigate on top simplices incident at a vertex of the edge of interest. Thus, it is clear that \( L_{NM}^1 < L_{IA^*}^{1.3D} \). Our tests show that, on average, \( L_{NM}^1 \) is about 35% smaller than \( L_{IA^*}^{1.3D} \). In the IS(3D), IG(3D), and SIG(3D) data structures, we retrieve simplices in the link of \( e \) through variants of Algorithm 6.4, like in the two-dimensional case. The key point is efficiency of the STAR query. It is clear that the IG(3D) data structure is more efficient than the IS(3D) and SIG(3D) representations. Our tests confirm that \( L_{IG}^{1.3D} \) is about 6% smaller than \( L_{IS}^{1.3D} \) and \( L_{SIG}^{1.3D} \approx 2 \times L_{IS}^{1.3D} \). Moreover, on average, \( S_{IA^*}^{1.3D} \approx 3.4 \times L_{IG}^{1.3D} \). This result tends to prove the validity of GhostSimplexPointer references, which improve the efficiency of queries.

Finally, we analyze the retrieval of \( Lk(f) \) for a triangle \( f \) in a simplicial 3-complex \( \Sigma \). The IS(3D), IG(3D), and SIG(3D) data structures directly encode co-boundary relation \( R_{2,3} \), and exploit the same BOUNDARY query. Thus, \( L_{IS}^{1.3D} = L_{SIG}^{2.3D} = L_{IS}^{2.3D} \). Conversely, the IA\(^{*}\)(3D) and NMIA data structures directly encode adjacency relation \( R_{3,3} \), and the STAR query can be easily retrieved, since we use GhostSimplexPointer references. Our tests show that \( L_{IS}^{2.3D} \approx L_{IA^*}^{2.3D} = L_{NM}^2 \).

As a consequence, we can summarize our results, regarding the retrieval of the link \( Lk(\sigma) \) for a \( p \)-simplex \( \sigma \), with \( 0 \leq p < 3 \), in a simplicial 3-complex:

- \( L_{IA^*}^{0.3D} < L_{NM}^0 < L_{IG}^{0.3D} < L_{IS}^{0.3D} < L_{SIG}^{0.3D} \), for vertices;
- \( L_{NM}^1 < L_{IA^*}^{1.3D} < L_{IG}^{1.3D} < L_{IS}^{1.3D} < L_{SIG}^{1.3D} \), for edges;
- \( L_{IG}^{2.3D} = L_{SIG}^{2.3D} = L_{IS}^{2.3D} \approx L_{IA^*}^{2.3D} = L_{NM}^2 \), for triangles.

### 7.5.5 Experimental Analysis of Non-Manifold Singularities Recognition

In this section, we provide comparisons regarding execution of the IS\_MANIFOLD query for all the data structures implemented in the Mangrove TDS framework. Specifically, we have designed algorithms for recognizing non-manifold singularities in the IS and IA\(^{*}\) data structures in Sections 6.2.6 and 6.3.6, respectively. We have also discussed implementations of the IS\_MANIFOLD query in the TS and NMIA data structures in Sections 7.1.2 and 7.2.3, respectively. Finally, in the SIG and IG data structures, we can exploit implementations of IS\_MANIFOLD query, which we proposed in Sections 7.3.2 and 7.4.2, respectively.

Recall that this operation is not decidable for simplicial \( d \)-complexes, with \( d \geq 6 \) [Nab96]. In the
remainder of this section, we concentrate our attention on simplicial complexes of dimension up to three.

First, we analyze average running times of the IS\_MANIFOLD query for simplicial 2-complexes, discretized by the IG(2D), IS(2D), SIG(2D), IA\(^∗\)(2D) and TS data structures. For these data structures, we denote average running times required for recognizing non-manifold \(p\)-simplices, with \(0 \leq p < 2\), as \(M_{IS}^{p, 2D}\), \(M_{IG}^{p, 2D}\), \(M_{SIG}^{p, 2D}\), \(M_{IA}^{p, 2D}\), and \(M_{TS}^{p, 2D}\). Average running times for the IS\_MANIFOLD query on simplicial 2-complexes are summarized in Table 7.13.

![Table 7.13: Average running times for recognizing non-manifold vertices and edges in digital 2D shapes, represented through the IG(2D) (\(M_{IG}^{2D}\) and \(M_{IG}^{1, 2D}\), respectively), IS(2D) (\(M_{IS}^{0, 2D}\) and \(M_{IS}^{1, 2D}\), respectively), SIG(2D) (\(M_{SIG}^{0, 2D}\) and \(M_{SIG}^{1, 2D}\), respectively), IA\(^∗\)(2D) (\(M_{IA}^{0, 2D}\) and \(M_{IA}^{1, 2D}\), respectively), and TS (\(M_{TS}^{1, 2D}\) and \(M_{TS}^{1, 2D}\), respectively) data structures. Note that \(M_{IA}^{1, 2D} = M_{TS}^{1, 2D}\).](image)

As discussed in Section 2.3, an edge \(e\) in a simplicial 2-complex is non-manifold if and only if its star is formed by more than two triangles, and, thus, we can check the number \(\|R_{1,2}(e)\|\) of triangles in \(St(e)\). In the IG(2D), IS(2D), and SIG(2D) data structures, we directly encode co-boundary relation \(R_{1,2}\), hence \(M_{IG}^{1, 2D} = M_{SIG}^{1, 2D}\). Conversely, in the IA\(^∗\)(2D) and TS data structures, a non-manifold edge \(e\) is always characterized by partial co-boundary relation \(R_{1,2}^{\bot}(e)\). As a consequence, \(M_{IA}^{1, 2D} = M_{TS}^{1, 2D}\). Thus, we can state that \(M_{IS}^{1, 2D} \approx M_{IA}^{1, 2D}\). Note that these running times do not depend on the specific 2D shape we are using.

Now, we analyze the recognition of a non-manifold vertex \(v\) in a simplicial 2-complex \(\Sigma\). Recall that a vertex is non-manifold if its link is formed by more than one connected component, or it is not formed by at most two vertices. Also, a non-manifold vertex bounds a non-manifold edge. In our algorithms, proposed in Chapters 6 and 7, we consider several cases, according to...
the content of topological data structures. Specifically, we consider information regarding the link of a vertex \( v \). In addition, critical aspects in the IS\_MANIFOLD query are the retrieval of co-boundary relation \( R_{0,1} \), and the recognition of non-manifold edges (just discussed above).

The IS(2D) data structure directly represents connected components of \( Lk(v) \), while the SIG(2D) representation encodes top edges and 2-clusters in \( St(v) \). Moreover, as discussed in Section 7.5.2, \( S_{IS}^{0,2D} < S_{SIG}^{0,2D} \), where \( S_{IS}^{0,2D} \) and \( S_{SIG}^{0,2D} \) are average running times of the STAR query in the IS(2D) and SIG(2D) data structures, respectively. Conversely, the IG(2D) data structure does not encode any information about the link of a vertex \( v \), thus we must retrieve additional information on \( Lk(v) \), as discussed in Section 7.4.2. As a consequence, \( M_{IS}^{0,2D} < M_{SIG}^{0,2D} < M_{IG}^{0,2D} \). Our tests show that \( M_{SIG}^{0,2D} \approx 1.8 \times M_{IS}^{0,2D} \), and \( M_{IG}^{0,2D} \approx 21 \times M_{SIG}^{0,2D} \). These results tend to prove that the IG(2D) data structure is not suitable for recognizing non-manifold vertices. Conversely, the IA\(^*(2D)\) and TS data structures support efficiently the IS\_MANIFOLD query, since each basic operation is efficient on these data structures. For instance, the IA\(^*(2D)\) data structure is the most efficient data structure for executing the STAR query on vertices, as shown in Section 7.5.2. Recall that the IA\(^*(2D)\) and TS data structures are the same if and only if there are no non-manifold edges. Our tests confirm that the \( M_{IA}^{0,2D} \) is about 6% larger than \( M_{IS}^{0,2D} \). Also in this case, this overhead can become quite large: for instance, \( M_{TS}^{0,2D} \approx 1.46 \times M_{IS}^{0,2D} \) and \( M_{TS}^{0,2D} \approx 1.2 \times M_{IS}^{0,2D} \) in the “800-Cubes” and “Pinched-pie” shapes, respectively. Our tests also show that, on average, \( M_{IS}^{0,2D} \approx 1.6 \times M_{TS}^{0,2D} \).

As a consequence, we can summarize our results, regarding the recognition of a non-manifold \( p \)-simplex \( \sigma \), with \( 0 \leq p < 2 \), in a simplicial 2-complex:

- \( M_{IA}^{0,2D} < M_{TS}^{0,2D} < M_{IS}^{0,2D} < M_{SIG}^{0,2D} < M_{IG}^{0,2D} \), for vertices;
- \( M_{IA}^{0,2D} = M_{IS}^{1,2D} = M_{SIG}^{1,2D} \approx M_{IA}^{1,2D} = M_{TS}^{1,2D} \), for edges.

Now, we analyze average running times of the IS\_MANIFOLD query for simplicial 3-complexes, discretized by the IG(3D), IS(3D), SIG(3D), IA\(^*(3D)\) and NMIA data structures. For these data structures, we denote average running times required for recognizing non-manifold \( p \)-simplices, with \( 0 \leq p < 2 \), as \( M_{IS}^{0,3D}, M_{IG}^{0,3D}, M_{SIG}^{0,3D}, M_{IA}^{0,3D} \), and \( M_{NMIA}^{0,3D} \). Table 7.14 summarizes average running times for the IG(3D), IS(3D), and SIG(3D) data structures, while Table 7.15 summarizes average running times for the IA\(^*(3D)\) and NMIA data structures.

Now, we analyze the recognition of a non-manifold edge \( e \) in a simplicial 3-complex \( \Sigma \). Note that, in the NMIA data structure, a non-manifold edge \( e \) is always characterized by partial relation \( R_{1,2}(e) \). As a consequence, the NMIA data structure is the most efficient data structure, as confirmed by our tests. Conversely, the IA\(^*(3D)\) data structure encodes partial relation \( R_{1,2}(e) \) only if there is at least one top triangle in \( St(e) \). Moreover, as discussed in Section 6.2.6, an edge is manifold if there is only one cluster incident at each vertex of \( e \), and, thus, incident at \( e \). As a consequence, the IA\(^*(3D)\) data structure is as efficient as the NMIA representation for manifolds, and if \( St(e) \) contains at least one top triangle. Otherwise, the IA\(^*(3D)\) representation is slower than the NMIA data structure, for instance with the “Halves” and “Sierpinski” shapes, where
additional information on the IG(3D) data structure does not encode any information about shapes, represented through the IG(3D) (Table 7.14: Average running times for recognizing non-manifold vertices and edges in digital 3D), and the number of clusters in \( L_k \), respectively) data structures.

<table>
<thead>
<tr>
<th>Shape</th>
<th>( M_{IS}^{3D} )</th>
<th>( M_{SIG}^{3D} )</th>
<th>( M_{IG}^{3D} )</th>
<th>( M_{IS}^{1,3D} )</th>
<th>( M_{SIG}^{1,3D} )</th>
<th>( M_{IG}^{1,3D} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>0.3092</td>
<td>0.5021</td>
<td>23.8700</td>
<td>0.0047</td>
<td>0.0048</td>
<td>1.9236</td>
</tr>
<tr>
<td>Cylinder</td>
<td>0.3645</td>
<td>0.5852</td>
<td>39.3440</td>
<td>0.0046</td>
<td>0.0047</td>
<td>2.5471</td>
</tr>
<tr>
<td>Gargoyle</td>
<td>0.3782</td>
<td>0.6441</td>
<td>37.3900</td>
<td>0.0046</td>
<td>0.0047</td>
<td>2.8897</td>
</tr>
<tr>
<td>Rings</td>
<td>0.3323</td>
<td>0.5563</td>
<td>30.4000</td>
<td>0.0046</td>
<td>0.0046</td>
<td>2.1215</td>
</tr>
<tr>
<td>Torus 3D</td>
<td>0.4694</td>
<td>0.7570</td>
<td>61.4900</td>
<td>0.0048</td>
<td>0.0048</td>
<td>3.2538</td>
</tr>
<tr>
<td>Arc</td>
<td>0.2308</td>
<td>0.3538</td>
<td>12.6200</td>
<td>0.0046</td>
<td>0.0046</td>
<td>1.5632</td>
</tr>
<tr>
<td>Balloon</td>
<td>0.4878</td>
<td>0.6916</td>
<td>6.7962</td>
<td>0.0769</td>
<td>0.0971</td>
<td>0.9438</td>
</tr>
<tr>
<td>Bucket</td>
<td>0.1698</td>
<td>0.1698</td>
<td>4.5472</td>
<td>0.0240</td>
<td>0.0240</td>
<td>0.7964</td>
</tr>
<tr>
<td>Chime</td>
<td>0.1789</td>
<td>0.2114</td>
<td>7.6829</td>
<td>0.0047</td>
<td>0.0048</td>
<td>0.9364</td>
</tr>
<tr>
<td>Flasks</td>
<td>0.3845</td>
<td>0.5833</td>
<td>19.8665</td>
<td>0.0166</td>
<td>0.0220</td>
<td>1.5445</td>
</tr>
<tr>
<td>Halves</td>
<td>0.6587</td>
<td>0.4365</td>
<td>27.4127</td>
<td>0.0047</td>
<td>0.0048</td>
<td>2.0969</td>
</tr>
<tr>
<td>Sierpinski</td>
<td>0.9374</td>
<td>0.0069</td>
<td>8.2706</td>
<td>0.0047</td>
<td>0.0048</td>
<td>2.9995</td>
</tr>
<tr>
<td>Teapot</td>
<td>0.7121</td>
<td>1.0333</td>
<td>15.2100</td>
<td>0.0764</td>
<td>0.1033</td>
<td>1.4143</td>
</tr>
<tr>
<td>Wheel</td>
<td>0.2587</td>
<td>0.3184</td>
<td>16.4030</td>
<td>0.0047</td>
<td>0.0048</td>
<td>1.4200</td>
</tr>
</tbody>
</table>

Table 7.14: Average running times for recognizing non-manifold vertices and edges in digital 3D shapes, represented through the IG(3D) (\( M_{IG}^{3D} \) and \( M_{IG}^{1,3D} \), respectively), IS(3D) (\( M_{IS}^{3D} \) and \( M_{IS}^{1,3D} \), respectively), and SIG(3D) (\( M_{SIG}^{3D} \) and \( M_{SIG}^{1,3D} \), respectively) data structures.

<table>
<thead>
<tr>
<th>Shape</th>
<th>( M_{IA}^{1,3D} )</th>
<th>( M_{IS}^{1,3D} )</th>
<th>( M_{NM}^{1,3D} )</th>
<th>( M_{IC}^{1,3D} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>0.1154</td>
<td>0.2891</td>
<td>0.0046</td>
<td>0.0047</td>
</tr>
<tr>
<td>Cylinder</td>
<td>0.1811</td>
<td>0.3262</td>
<td>0.0047</td>
<td>0.0048</td>
</tr>
<tr>
<td>Gargoyle</td>
<td>0.1445</td>
<td>0.2751</td>
<td>0.0048</td>
<td>0.0047</td>
</tr>
<tr>
<td>Rings</td>
<td>0.1606</td>
<td>0.2891</td>
<td>0.0046</td>
<td>0.0047</td>
</tr>
<tr>
<td>Torus 3D</td>
<td>0.1844</td>
<td>0.4288</td>
<td>0.0045</td>
<td>0.0047</td>
</tr>
<tr>
<td>Arc</td>
<td>0.0308</td>
<td>0.2000</td>
<td>0.0046</td>
<td>0.0046</td>
</tr>
<tr>
<td>Balloon</td>
<td>0.0487</td>
<td>0.1163</td>
<td>0.0046</td>
<td>0.0059</td>
</tr>
<tr>
<td>Bucket</td>
<td>0.0189</td>
<td>0.1509</td>
<td>0.0048</td>
<td>0.0060</td>
</tr>
<tr>
<td>Chime</td>
<td>0.0325</td>
<td>0.1220</td>
<td>0.0047</td>
<td>0.0120</td>
</tr>
<tr>
<td>Flasks</td>
<td>0.0998</td>
<td>0.2180</td>
<td>0.0046</td>
<td>0.0065</td>
</tr>
<tr>
<td>Halves</td>
<td>0.1202</td>
<td>0.3690</td>
<td>0.0045</td>
<td>0.0223</td>
</tr>
<tr>
<td>Sierpinski</td>
<td>0.0053</td>
<td>0.0069</td>
<td>0.0046</td>
<td>0.0228</td>
</tr>
<tr>
<td>Teapot</td>
<td>0.0953</td>
<td>0.1815</td>
<td>0.0049</td>
<td>0.0056</td>
</tr>
<tr>
<td>Wheel</td>
<td>0.0547</td>
<td>0.2214</td>
<td>0.0048</td>
<td>0.0057</td>
</tr>
</tbody>
</table>

Table 7.15: Average running times for recognizing non-manifold vertices and edges in digital 3D shapes, represented through the IA\(^+(3D)\) (\( M_{IA}^{1,3D} \) and \( M_{IA}^{1,3D} \), respectively) and NMIA (\( M_{NM}^{1,3D} \) and \( M_{NM}^{1,3D} \), respectively) data structures.

\( M_{IA}^{1,3D} \approx 4.9 \times M_{NM}^{1,3D} \). Our tests show that, on average, \( M_{IA}^{1,3D} \approx 1.8 \times M_{NM}^{1,3D} \). In the IS(3D) and SIG(3D) data structures, we first consider, respectively, the number of connected components in \( L_k(e) \), and the number of clusters in \( St(e) \). In addition, we check if \( L_k(e) \) is formed by only two vertices. Thus, according to our running times, presented in Section 7.5.4, \( M_{IS}^{1,3D} < M_{SIG}^{1,3D} \). Our results show, on average, that \( M_{IS}^{1,3D} \approx 1.07 \times M_{IS}^{1,3D} \) and \( M_{IS}^{1,3D} \approx 2.9 \times M_{IA}^{1,3D} \). Conversely, the IG(3D) data structure does not encode any information about \( L_k(e) \), thus we must retrieve additional information on \( L_k(e) \), as discussed in Section 7.4.2. Hence, \( M_{SIG}^{1,3D} < M_{IG}^{1,3D} \). Our tests
show that, on average, $\mathcal{M}_{I^G}^{1,3D} \approx 337 \times \mathcal{M}_{SIG}^{1,3D}$, thus the IG(3D) data structure is not efficient in recognizing non-manifold edges.

Now, we analyze the recognition of a non-manifold vertex $v$ in a simplicial 3-complex $\Sigma$. Also in this case, we consider information regarding the link of a vertex $v$. In addition, other critical aspects in the IS_MANIFOLD query are the retrieval of co-boundary relation $R_{0,1}$, and the recognition of non-manifold edges (just discussed above). As a consequence, the IA$^*$ (3D) data structure supports efficiently the IS_MANIFOLD query. According to our running times in Section 7.5.2, and those regarding the recognition of non-manifold edges, we conjecture that $\mathcal{M}_{IA^*}^{0,3D} < \mathcal{M}_{NM}^{0,3D} < \mathcal{M}_{IS}^{0,3D}$. Our tests show that $\mathcal{M}_{NM}^{0,3D} \approx 3 \times \mathcal{M}_{IA^*}^{0,3D}$ and $\mathcal{M}_{IS}^{0,3D} \approx 2.57 \times \mathcal{M}_{NM}^{0,3D}$. In the IS(3D), SIG(3D), and IG(3D) data structures, we exploit the same dimension-independent algorithm used in the two-dimensional case. As a consequence, $\mathcal{M}_{IS}^{0,3D} < \mathcal{M}_{SIG}^{0,3D} < \mathcal{M}_{IG}^{0,3D}$. Note that, in this case, the IG(3D) data structure is absolutely inefficient, due to the worst performances, while recognizing non-manifold edges. Our tests show that, on average, $\mathcal{M}_{SIG}^{0,3D} \approx 1.3 \times \mathcal{M}_{IS}^{0,3D}$ and $\mathcal{M}_{IG}^{0,3D} \approx 127 \times \mathcal{M}_{SIG}^{0,3D}$. However, in specific configurations, $\mathcal{M}_{SIG}^{0,3D}$ may be smaller than $\mathcal{M}_{IS}^{0,3D}$, for instance, when the star of a vertex $v$ does not contain any top triangle. In these situations, $St(v)$ is formed only by 3-clusters. In the “Sierpinski” shape, the star of each vertex contains only two 3-clusters, thus we immediately recognize a non-manifold vertex. In this case, $\mathcal{M}_{IS}^{0,3D} \approx 136 \times \mathcal{M}_{SIG}^{0,3D}$, since $Lk(v)$ contains only one connected component.

As a consequence, we can summarize our results, regarding the recognition of a non-manifold $p$-simplex $\sigma$, with $0 \leq p < 2$, in a simplicial 3-complex:

- $\mathcal{M}_{IA^*}^{0,3D} < \mathcal{M}_{NM}^{0,3D} < \mathcal{M}_{IS}^{0,3D} < \mathcal{M}_{SIG}^{0,3D} < \mathcal{M}_{IG}^{0,3D}$, for vertices;
- $\mathcal{M}_{NM}^{1,3D} < \mathcal{M}_{IA^*}^{1,3D} < \mathcal{M}_{IS}^{1,3D} < \mathcal{M}_{SIG}^{1,3D} < \mathcal{M}_{IG}^{1,3D}$, for edges.

### 7.5.6 Experimental Analysis of Construction Algorithms

In this section, we provide comparisons regarding the construction algorithm for all the data structures implemented in the Mangrove TDS framework. Specifically, we have designed algorithms for constructing the IS and IA$^*$ data structures in Sections 6.2.7 and 6.3.7, respectively. We have also proposed algorithms for building the TS and NMIA data structures in Sections 7.1.3 and 7.2.4, respectively. Finally, we have designed algorithms for building the SIG and IG data structures in Sections 7.3.3 and 7.4.3, respectively.

First, we analyze running times of these algorithms on simplicial 2-complexes described by the IG(2D), IS(2D), SIG(2D), IA$^*$ (2D) and TS data structures. We denote running times required for constructing these data structures as $R_{IG}^{2D}$, $R_{IS}^{2D}$, $R_{SIG}^{2D}$, $R_{IA^*}^{2D}$, and $R_{TS}$, respectively. Average running times for the construction algorithms are summarized in Table 7.16.

Construction of the IS(2D) and SIG(2D) data structures is equivalent to retrieve boundary relations encoded in the Incidence Graph, and identify partial co-boundary relations. In the IG(2D) data structure, boundary and co-boundary relations are symmetric, while we execute a traversal
Table 7.16: Average running times for constructing the IG(2D) ($R_{IG}^{2D}$), IA$^*$ (2D) ($R_{IA}^{2D}$), TS($R_{TS}$), SIG(2D) ($R_{SIG}^{2D}$), and IS(2D) ($R_{IS}^{2D}$) data structures, which describe digital 2D shapes.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$R_{IG}^{2D}$</th>
<th>$R_{IA}^{2D}$</th>
<th>$R_{TS}$</th>
<th>$R_{SIG}^{2D}$</th>
<th>$R_{IS}^{2D}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cone</td>
<td>166</td>
<td>202</td>
<td>203</td>
<td>305</td>
<td>500</td>
</tr>
<tr>
<td>Crumb</td>
<td>89</td>
<td>97</td>
<td>98</td>
<td>177</td>
<td>270</td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>24</td>
<td>27</td>
<td>29</td>
<td>38</td>
<td>62</td>
</tr>
<tr>
<td>Football 1</td>
<td>350</td>
<td>420</td>
<td>422</td>
<td>884</td>
<td>1554</td>
</tr>
<tr>
<td>Football 2</td>
<td>268</td>
<td>299</td>
<td>301</td>
<td>611</td>
<td>1047</td>
</tr>
<tr>
<td>Torus</td>
<td>3127</td>
<td>4569</td>
<td>4660</td>
<td>13961</td>
<td>32840</td>
</tr>
<tr>
<td>Armchair</td>
<td>1713</td>
<td>1834</td>
<td>1860</td>
<td>6517</td>
<td>11180</td>
</tr>
<tr>
<td>800-Cubes</td>
<td>1665</td>
<td>1679</td>
<td>1787</td>
<td>4224</td>
<td>9993</td>
</tr>
<tr>
<td>Cylinders</td>
<td>31</td>
<td>34</td>
<td>40</td>
<td>47</td>
<td>64</td>
</tr>
<tr>
<td>Pinched-pie</td>
<td>260</td>
<td>307</td>
<td>336</td>
<td>584</td>
<td>998</td>
</tr>
<tr>
<td>Twist</td>
<td>273</td>
<td>326</td>
<td>331</td>
<td>648</td>
<td>1085</td>
</tr>
<tr>
<td>Robot</td>
<td>859</td>
<td>1039</td>
<td>1065</td>
<td>2458</td>
<td>5027</td>
</tr>
<tr>
<td>Balance</td>
<td>1347</td>
<td>1472</td>
<td>1625</td>
<td>3088</td>
<td>6557</td>
</tr>
<tr>
<td>Carter</td>
<td>1038</td>
<td>1654</td>
<td>1694</td>
<td>3174</td>
<td>6783</td>
</tr>
<tr>
<td>Chandelier</td>
<td>2674</td>
<td>3771</td>
<td>3792</td>
<td>11243</td>
<td>26963</td>
</tr>
<tr>
<td>Frame</td>
<td>149</td>
<td>153</td>
<td>170</td>
<td>286</td>
<td>585</td>
</tr>
<tr>
<td>Tower</td>
<td>2528</td>
<td>3882</td>
<td>3883</td>
<td>11602</td>
<td>27343</td>
</tr>
<tr>
<td>Tower-wir</td>
<td>2048</td>
<td>3136</td>
<td>3158</td>
<td>8881</td>
<td>21524</td>
</tr>
</tbody>
</table>

of the star of each simplex in order to identify partial co-boundary relations in the IS(2D) and SIG(2D) data structures. Thus, $R_{SIG}^{2D} < R_{TS}$, and $R_{IG}^{2D} < R_{IS}^{2D}$. In fact, our tests show that, on average, $R_{SIG}^{2D} \approx 2.8 \times R_{IG}^{2D}$, and $R_{IS}^{2D} \approx 5.5 \times R_{IG}^{2D}$. Moreover, $R_{TS}^{2D} \approx 1.9 \times R_{SIG}^{2D}$.

As discussed in Section 6.3.7, while constructing the IA$^*$ (2D) data structure, we generate and sort raw faces related to edges of all the triangles, and then we identify adjacency relation $R_{2,2}$ and partial co-boundary relations $R_{0,2}^*$. In the IG(2D) data structure, we generate and sort raw faces related to all the faces of a top simplex, except vertices, and then we store co-boundary relations, as discussed in Section 7.4.3. As a consequence, in a simplicial 2-complex, we repeat the same operation performed in the IA$^*$ (2D) data structure. In any case, we can conjecture that $R_{IG}^{2D} < R_{IA}^{2D}$, since we must traverse triangles incident at a vertex in order to identify partial co-boundary relations $R_{0,2}^*$ in the IA$^*$ (2D) data structure. Our tests show that, on average, $R_{IG}^{2D}$ is 23% smaller than $R_{IA}^{2D}$. Note that constructing the TS data structure is completely equivalent to constructing the IA$^*$ data structure, except for the different encoding of non-manifold adjacency along edges, as discussed in Section 7.1.3. Our tests shows that, on average, $R_{IA}^{2D}$ is 4% smaller than $R_{TS}$. Moreover, $R_{SIG}^{2D} \approx 2.2 \times R_{TS}$.

As a consequence, we can summarize our results, regarding the construction of topological data structures representing a simplicial 2-complex:

$$R_{IG}^{2D} < R_{IA}^{2D} < R_{TS} < R_{SIG}^{2D} < R_{IS}^{2D}$$

Now, we analyze running times of construction algorithms on simplicial 3-complexes described
by the IG(3D), IS(3D), SIG(3D), IA∗(3D) and NMIA data structures. We denote running times required for constructing these data structures as $R_{3D}^{IG}$, $R_{3D}^{IS}$, $R_{3D}^{SIG}$, $R_{3D}^{IA}$, and $R_{NM}$, respectively. Average running times for the construction algorithms are summarized in Table 7.17.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$R_{3D}^{IA}$</th>
<th>$R_{NM}$</th>
<th>$R_{3D}^{IG}$</th>
<th>$R_{3D}^{SIG}$</th>
<th>$R_{3D}^{IS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basket</td>
<td>912</td>
<td>1513</td>
<td>2566</td>
<td>4244</td>
<td>16624</td>
</tr>
<tr>
<td>Cylinder</td>
<td>944</td>
<td>2018</td>
<td>3249</td>
<td>5749</td>
<td>24109</td>
</tr>
<tr>
<td>Gargoyle</td>
<td>2240</td>
<td>4242</td>
<td>7120</td>
<td>14688</td>
<td>67428</td>
</tr>
<tr>
<td>Rings</td>
<td>1745</td>
<td>3431</td>
<td>4809</td>
<td>11898</td>
<td>49343</td>
</tr>
<tr>
<td>Torus 3D</td>
<td>2069</td>
<td>4693</td>
<td>6800</td>
<td>16920</td>
<td>74306</td>
</tr>
<tr>
<td>Arc</td>
<td>31</td>
<td>52</td>
<td>95</td>
<td>135</td>
<td>424</td>
</tr>
<tr>
<td>Balloon</td>
<td>393</td>
<td>530</td>
<td>1213</td>
<td>1330</td>
<td>3625</td>
</tr>
<tr>
<td>Bucket</td>
<td>22</td>
<td>29</td>
<td>34</td>
<td>48</td>
<td>157</td>
</tr>
<tr>
<td>Chime</td>
<td>61</td>
<td>113</td>
<td>231</td>
<td>341</td>
<td>1191</td>
</tr>
<tr>
<td>Flasks</td>
<td>1035</td>
<td>1393</td>
<td>2121</td>
<td>3981</td>
<td>15010</td>
</tr>
<tr>
<td>Halves</td>
<td>186</td>
<td>327</td>
<td>541</td>
<td>838</td>
<td>2974</td>
</tr>
<tr>
<td>Sierpinski</td>
<td>8567</td>
<td>9739</td>
<td>11480</td>
<td>54402</td>
<td>468879</td>
</tr>
<tr>
<td>Teapot</td>
<td>2158</td>
<td>3266</td>
<td>4314</td>
<td>10502</td>
<td>41999</td>
</tr>
<tr>
<td>Wheel</td>
<td>243</td>
<td>519</td>
<td>779</td>
<td>975</td>
<td>3471</td>
</tr>
</tbody>
</table>

Table 7.17: Average running times for constructing the IA∗(3D) ($R_{3D}^{IA}$), NMIA ($R_{NM}$), IG(3D) ($R_{3D}^{IG}$), SIG(3D) ($R_{3D}^{SIG}$), and IS(3D) ($R_{3D}^{IS}$) data structures, which describe digital 3D shapes.

We exploit the same dimension-independent algorithms used previously. As a consequence, constructing the IG(3D) data structure is an operation, which is faster than building the IS(3D) and SIG(3D) data structures. In fact, our tests show that, on average, $R_{3D}^{SIG} \approx 1.98 \times R_{3D}^{IG}$ and $R_{3D}^{IS} \approx 4 \times R_{3D}^{SIG}$.

In the tridimensional case, it is interesting to study what happens while constructing the IA∗(3D) and IG(3D) data structures. As discussed in Section 6.3.7, we generate and sort raw faces related only to faces bounding $s_2$ top triangles, and to triangles bounding $s_3$ tetrahedra. Conversely, in the IG(3D) data structure, we generate raw faces related to all the subfaces of each top simplex (except vertices). Thus, we handle a wide number of raw faces. Top simplices are a small subset of all the simplices in a simplicial complex. Hence, we can conjecture that $R_{3D}^{IA} < R_{3D}^{IG}$. Our tests show that $R_{3D}^{IG} \approx 2.7 \times R_{3D}^{IA}$: specifically, the IA∗(3D) data structure is the representation which can be built faster, according to our tests. The NMIA data structure can be built as discussed in Section 7.2.4. Boundary relations can be retrieved as in the IA∗(3D) data structure. Adjacency relation $R_{3,3}$ and relations $R_{2,cl}$ and $R_{3,cl}$ can be quickly retrieved by analyzing raw faces. Conversely, for each vertex $v$, we identify top edges and vertex-based clusters incident at $v$ by using a variant of the algorithm used in the IS(3D) data structure. Thus, we can conjecture that $R_{3D}^{IA} < R_{NM}$. Our tests show that, on average, $R_{NM} \approx 1.6 \times R_{3D}^{IA}$, and $R_{3D}^{IG} \approx 1.6 \times R_{NM}$.

As a consequence, we can summarize our results, regarding the construction of topological data structures representing a simplicial 3-complex:

$$R_{3D}^{IA} < R_{NM} < R_{3D}^{IG} < R_{3D}^{SIG} < R_{3D}^{IS}$$
Summary of Experimental Results

In this chapter, we have performed quantitative comparisons on the efficiency of the five queries, plus the construction algorithm, for all the data structures implemented within our Mangrove TDS framework. We have represented simplicial 2-complexes through the IA∗(2D), TS, IS(2D), IG(2D), and SIG(2D) data structures. Conversely, simplicial 3-complexes have been represented through the IA∗(3D), NMIA, IS(3D), IG(3D), and SIG(3D) data structures.

Our tests show that the IA∗ and IS data structures offer an optimal compromise regarding expressive power, storage cost, and efficiency of all the queries for simplicial 2- and 3-complexes. Specifically, the IA∗ data structure can be considered as the most efficient data structure among those we have analyzed. Recall that the IA∗ data structure encodes only vertices and top simplices in an abstract simplicial complex of any dimension.

The large majority of vertex-based queries are most efficient on the IA∗ data structure, like, for instance, the STAR and LINK queries. As a consequence, the IS_MANIFOLD query is also efficient on vertices. Conversely, the ADJACENCY query is slightly slower than on the IG data structure, except for maximal simplices. Recall that, for a p-simplex, the Incidence Graph directly encodes boundary relation \( R_{p,p-1} \) and \( R_{p-1,p} \), thus the time complexity of adjacency relation \( R_{p,p} \) can be considered as almost constant. In the other data structures (including the IA∗ data structure), it is necessary to extract these relations. In any case, our tests show that the IA∗ data structure is the most efficient for doing it, except the Incidence Graph.

One of the bottlenecks in the IA∗ data structure consists of recognizing non-manifold \((p-1)\)-simplices \( \tau \) at which no top \( p \)-simplex is incident, since partial co-boundary relation \( R_{p-1,p}^* \) is empty. However, the overhead needed for retrieving the number of clusters in \( St(\tau) \) is quite limited. In addition, constructing the IA∗ data structure in the 2D case is slightly slower than building the Incidence Graph, since we must consider raw 1-faces related to all the edges in the input simplicial 2-complex. However, in the 3D case, constructing the IA∗ data structure becomes faster than building the Incidence Graph, since we consider only immediate subfaces of each top simplex instead of all the simplices.

The IA∗ data structure does not encode all the simplices, thus its expressive power may be reduced with respect to the IS data structure. However, our tests also show that the GhostSimplexPointer references improve expressive power of local mangroves, like the IA∗ data structure. In fact, they provide an effective representation of ghost simplices, and allow for a fast retrieval of topological queries. For instance, in the IA∗ data structure, it is possible to retrieve the combinatorial boundary of ghost simplices with a limited overhead with respect to the IS data structure. In addition, the combinatorial boundary of a top simplex is faster than in the IS data structure. Generally speaking, the GhostSimplexPointer references allow retrieving topological relations with a limited overhead with respect to global mangroves, like the IG and IS data structures.

Conversely, our tests show that the IS data structure is less efficient than the IA∗ data structure, although the IS data structure is represented by a global mangrove, and all the simplices are
directly encoded. There are two bottlenecks in the IS data structure. Specifically, for a $p$-simplex $\sigma$, the retrieval of co-boundary relation requires analysis of the complete star $St(\sigma)$, since the IS data structure encodes partial co-boundary relation $R^*_p,p+1$. This can be an issue while retrieving the adjacency relation, which becomes an inefficient operation. In addition, our tests show that constructing the IS data structure is the slowest operation, since we must visit the complete star of a simplex in order to retrieve partial co-boundary relation. However, this construction algorithm may be improved and parallelized. In general, our tests show that the performances of the IS data structures are better than in the SIG data structure for most of the tests.

In other words, we can state that the issues with the IS data structure have been overcome in the IA* data structure, which also results in a very compact representation.
Chapter 8

The Manifold-Connected Decomposition

The topological structure of any non-manifold shape can be analyzed through a decomposition into a collection of almost manifold components, having a common intersection formed by several non-manifold singularities. In this chapter, we consider the Manifold-Connected Decomposition (MC-Decomposition). The basic concepts underlying this decomposition, but limited to 2D and 3D simplicial complexes, have been introduced in [HDF07a, HDF07b]. A non-manifold simplicial shape is decomposed into nearly-manifold components, known as the Manifold-Connected components (MC-components). The MC-Decomposition is completely dimension-independent, and it can be defined and computed for non-manifold simplicial shapes of any dimension.

A suitable representation for the MC-Decomposition is a graph-based data structure, where the nodes describe MC-components, and the arc describe the connectivity among several MC-components, sharing a common set of non-manifold singularities. In [CDF11] we have proposed two graph-based representations of the MC-Decomposition, which can be combined with any topological data structure. The external level of these graph-based data structures consists of a collection of MC-components, while the internal level consists of a unique topological data structure. In this context, each MC-component is completely defined in terms of references to simplices in the input shape, as it happens for spatial indices [Sam06]. Similarly, each arc contains a reference for each non-manifold simplex shared by several MC-components.

These graph-based data structures are suitable for different applications. Specifically, we can decouple the representation of the MC-components, provided by any topological data structure, from the structural model of the input shape, provided by the MC-Decomposition. For instance, if we combine one of these graph-based data structures with the IS data structure [DFHPC10], discussed in Section 4.1, then we obtain a complete description of any shape, and we can access each simplex. Conversely, if we use the IA* data structure [CDFW11], discussed in Section 5.1, then the resulting description is more compact. Also, these representations allow having the same
vertex ordering for all the MC-components. This condition is mandatory in several applications, like the computation of simplicial homology, which we will describe in Chapter 9.

In Section 8.1, we define the MC-Decomposition, and we review several properties of MC-components in two and three dimensions, which have been demonstrated in [HDF07a]. In Section 8.2, we describe two algorithms for computing the MC-Decomposition of non-manifold simplicial shapes, and relating MC-components and non-manifold singularities. In Section 8.3, we describe three graph-based data structures for representing the MC-Decomposition. Two of these representations have been proposed in [CDF11]. Finally, in Section 8.4, we provide our experimental comparisons for each graph-based representation, combined with all the topological data structures in our Mangrove Topological Data Structure (Mangrove TDS) framework, discussed in Chapters 6 and 7.

8.1 Design and Properties of the Manifold-Connected Decomposition

In this section, we describe the Manifold-Connected Decomposition (MC-Decomposition), initially proposed in [HDF07a, HDF07b]. In Section 8.1.1, we review several properties of the Manifold-Connected components (MC-components), which are the basic elements of this decomposition. In Section 8.1.2, we completely define the MC-Decomposition of any arbitrary shape discretized by a simplicial complex.

8.1.1 Manifold-Connected components

In this section, we describe the Manifold-Connected components (MC-components), which are the basic elements in the MC-Decomposition of a non-manifold shape discretized by any simplicial complex. Also, we discuss several properties of the MC-components, which have been introduced in [HDF07a, HDF07b].

Our starting point consists of several concepts introduced in Sections 2.1 and 2.3, specifically the concepts of path and manifold simplex in a simplicial complex. Note that these definitions can be also generalized to cell complexes. Given a simplicial d-complex Σ, an h-path, with $h \leq d$, is a sequence of $(h+1)$-simplices $(\sigma_i)_{i=0}^m$ in Σ, such that two consecutive $h$-simplices $\sigma_{i-1}$ and $\sigma_i$ in this sequence share a $(h-1)$-simplex. Any $(d-1)$-simplex $\sigma$ in a regular simplicial d-complex Σ is manifold if just one or at most two d-simplices of Σ are incident at $\sigma$.

Two d-simplices $\sigma$ and $\sigma'$ in a regular simplicial d-complex Σ are manifold-connected if and only if there exists a $(d-1)$-path joining $\sigma$ and $\sigma'$ such that any pair of consecutive d-simplices share a manifold $(d-1)$-simplex. We call a $(d-1)$-path with these properties a manifold $(d-1)$-path. A regular d-complex Σ, in which every pair of d-simplices is manifold-connected, is a Manifold-Connected complex (MC-complex) of dimension d. Also, Σ is said to be incident at any d-simplex $\sigma$ in Σ and at all the faces of $\sigma$. We denote as $MC_d$ the class containing all the MC-complexes of
dimension $d$. Figure 8.1 show two MC-complexes of dimension 2, where triangles are connected through a manifold edge.

It is interesting to understand how MC-complexes are related to other types of complexes discussed in this thesis, namely the manifold, the combinatorial pseudo-manifold (shorten in pseudo-manifold), and the IQM complexes. These relations have been introduced in [HDF07a].

In Section 2.3, we have discussed manifold and combinatorial pseudo-manifold complexes. Recall that a simplicial $d$-complex $\Sigma$ is manifold if and only if the link of any $p$-simplex $\sigma$ in $\Sigma$, with $0 \leq p \leq d$, is combinatorially equivalent either to $B^h_\Sigma$ or to $S^h_\Sigma - 1$, for some $h \leq d - p - 1$. We define the class $M_d$ containing all the manifold $d$-complexes. Also, a regular $(d - 1)$-connected simplicial complex $\Sigma$ is a pseudo-manifold of dimension $d$ if and only if the star of any $(d - 1)$-simplex consists of just one or at most two $d$-simplices. We denote as $PSDM_d$ the class containing all the combinatorial pseudo-manifold $d$-complexes. Also, in Section 3.4.3, we have briefly reviewed the Initial Quasi-Manifold (IQM) complexes [DFMPP03, Mor03], which are regular simplicial $d$-complexes such that the star of any vertex is an MC-complex of dimension $d$. We denote as $IQM_d$ the class containing all the IQM $d$-complexes.

First, we focus our attention on simplicial 2-complexes embedded in the Euclidean space $\mathbb{E}^3$. Note that any simplicial 2-complex may contain non-manifold vertices and edges. By definition, a manifold 2-complex $\Sigma$ does not contain any non-manifold singularity. Thus, the link of any vertex $v$ in $\Sigma$ is combinatorially equivalent either to $B^h_\Sigma$, with $h \leq 1$, or to $S^0_\Sigma$. Conversely, the link of any edge $e$ in $\Sigma$ consists of at most two vertices, since at most two triangles are incident at $e$. In any IQM 2-complex $\Sigma$, the star of any vertex $v$ is a manifold 2-complex, and at most two triangles are incident at each edge in $St(v)$, hence the link of any vertex $v$ is a manifold 2-complex. As a consequence, at most two triangles are incident at each edge in $St(v)$, hence the link of any vertex $v$ is a manifold 2-complex. Therefore, any vertex $v$ is manifold, and there is no non-manifold edge in $\Sigma$. An IQM 2-complex is the same as a manifold 2-complex, and $M_2 \equiv IQM_2$.

A pseudo-manifold 2-complex $\Sigma$ does not contain any non-manifold edge, since the star of any edge consists of at most two triangles. Thus, an IQM 2-complex is a pseudo-manifold 2-complex. The reverse is not true, since a pseudo-manifold 2-complex may contain non-manifold vertices, as shown in Figure 8.1(a). Here, a pseudo-manifold 2-complex is pinched at a non-manifold vertex $v$, in such a way that $St(v)$ (in purple) is formed by two MC 2-complexes. Therefore, $IQM_2 \nsubseteq PSDM_2$.

Similarly, a pseudo-manifold 2-complex is also an MC-complex of dimension 2. The reverse is not true, since any MC-complex may contain non-manifold edges, as shown in Figure 8.1(b). Here, an MC-complex of dimension 2 is pinched at two non-manifold edges $e_1$ and $e_2$, shared by more than two triangles, respectively. Note that any MC-complex of dimension 2 can contain both non-manifold vertices and non-manifold edges. For instance, vertices bounding edges $e_1$ and $e_2$ in Figure 8.1(b) are non-manifold. Therefore, $PSDM_2 \nsubseteq MC_2$.

As a consequence, we can state that $M_2 \equiv IQM_2 \nsubseteq PSDM_2 \nsubseteq MC_2$. 192
Figure 8.1: (a) A pseudo-manifold of dimension 2, which is not an IQM 2-complex. In fact, it contains a non-manifold vertex $v$ such that $St(v)$ (in purple) is formed by two MC 2-complexes. (b) An MC-complex of dimension 2, which is pinched at two non-manifold edges $e_1$ and $e_2$. Hence, it cannot be a pseudo-manifold of dimension 2. Figures courtesy of [HDF07a, DFPH09].

Now, we can concentrate our attention on simplicial 3-complexes embedded in the Euclidean space $\mathbb{E}^3$. In this case, any simplicial 3-complex may contain only non-manifold vertices and edges, since any triangle is shared by at most two tetrahedra. It is clear that an MC-complex of dimension 3 is the same as a pseudo-manifold 3-complex (and vice versa), since a triangle is on the boundary of at most two tetrahedra. Hence, $PSDM_3 \equiv MC_3$.

In a manifold 3-complex $\Sigma$, the link of a vertex $v$ is combinatorially equivalent to either $B^2_v \Sigma$ or to $S^1_v \Sigma$. As a consequence, the link of $v$ is a manifold 2-complex. As demonstrated before, a manifold 2-complex is the same as an MC-complex of dimension 2, thus the star of $v$ is an MC-complex of dimension 3. Hence, a manifold 3-complex is the same as an IQM 3-complex. Unlike the two-dimensional case, the reverse is not true, since any IQM 3-complex may contain several non-manifold vertices, as the IQM 3-complex in Figure 8.2(a). Hence, $M_3 \not\subseteq IQM_3$.

In an IQM 3-complex, the star of each vertex $v$ is an MC-complex of dimension 3, thus it is a pseudo-manifold 3-complex. Hence, $St(v)$ is regular and 2-connected, and each triangle in $St(v)$ is shared between at most two tetrahedra. Thus, $Lk(v)$ is 1-connected, and there is no non-manifold edge in $St(v)$. As a consequence, an IQM 3-complex is the same as a pseudo-manifold 3-complex. The reverse is not true, since there exist pseudo-manifold 3-complexes, which are not IQM 3-complex, as the pseudo-manifold 3-complex in Figure 8.2(b). The link of any vertex $v$ consists of more than one connected component, or it is 0-connected. In the latter case, the pseudo-manifold 3-complex contains a non-manifold edge, as in Figure 8.2(b). Hence, $IQM_3 \not\subseteq PSDM_3$.

As a consequence, we can state that $M_3 \not\subseteq IQM_3 \not\subseteq PSDM_3 \equiv MC_3$.

In general, we have that $MC_d$, namely the class of MC-complexes of dimension $d$, is a superclass of $M_d$, namely the class of manifold $d$-complexes. Note that $M_d$ is not decidable for $d \geq 6$, as demonstrated in [Nab96], while this is not true for MC-complexes. We conjecture that $PSDM_d \equiv MC_d$ for simplicial $d$-complexes embedded in the Euclidean space $\mathbb{E}^d$.

193
8.1.2 Design of the Manifold-Connected Decomposition

In this section, we completely characterize the MC-Decomposition of any arbitrary shape discretized by a simplicial complex. First, we define the MC-Decomposition of a regular simplicial complex. Then, we adapt this definition to a non-regular simplicial complex.

Here, the challenge is to decompose any regular simplicial $d$-complex $\mathcal{X}$ into a collection of components, which are MC-complexes of dimension $d$. We denote them as MC-components of $\mathcal{X}$. Note that we define any MC-component of $\mathcal{X}$ in terms of its top $d$-simplices. Also, the manifold-connectivity relation among top $d$-simplices in $\mathcal{X}$ defines an equivalence relation. MC-components of $\mathcal{X}$ are the equivalence classes of top $d$-simplices of $\mathcal{X}$ with respect to the manifold-connectivity relation. In other words, a top $d$-simplex $\sigma$ in $\mathcal{X}$ is equivalent to top $d$-simplices in $\mathcal{X}$, which are reachable from $\sigma$ through a manifold $(d-1)$-path. Note that all the top simplices in $\mathcal{X}$ completely characterize $\mathcal{X}$. Hence, any top $d$-simplex belongs to only one MC-component. The collection of all the MC-components in $\mathcal{X}$ forms the Manifold-Connected Decomposition (MC-Decomposition) of $\mathcal{X}$. Several MC-components in the MC-Decomposition of a regular simplicial $d$-complex $\mathcal{X}$ may have a common intersection, which is a subcomplex of $\mathcal{X}$ of dimension lower than $d$. Figure 8.3(a) shows the MC-Decomposition of a regular simplicial 2-complex (namely the “Chair” shape [GGG09]), which consists of six MC-components of dimension 2 connected through chains of non-manifold edges.

Any non-regular simplicial $d$-complex $\Sigma$ is uniquely decomposed into a collection of maximal regular complexes $\Sigma^r_k$, formed by top $k$-simplices, with $1 \leq k \leq d$. Thus, the Manifold-Connected Decomposition (MC-Decomposition) of a non-regular simplicial $d$-complex $\Sigma$ is formed by MC-Decompositions of the maximal regular subcomplexes $\Sigma^r_k$, namely the collections of top $k$-simplices in $\Sigma$. Note that each top $k$-simplex $\sigma$ in $\Sigma$ belongs to only one subcomplex $\Sigma^r_k$, thus the MC-Decomposition of $\Sigma$ is unique. This decomposition provides a discrete counterpart of the stratification for analytics sets, defined in [Whi65]. Figure 8.3(b) shows the MC-Decomposition of a non-regular simplicial 3-complex, namely the “Flasks” shape [GGG09], which consists of several MC-complexes of dimension 2 and 3 connected through chains of non-manifold edges.
Figure 8.3: MC-Decompositions for several non-manifold simplicial shapes. (a) The MC-Decomposition of a regular simplicial 2-complex, namely the “Chair” shape [GGG09], which consists of six MC-components of dimension 2 connected through chains of non-manifold edges. (b) The MC-Decomposition of a non-regular simplicial 3-complex, namely the “Flasks” shape [GGG09], which consists of several MC-complexes of dimension 2 and 3 connected through chains of non-manifold edges. Figure courtesy of [DFPH09].

As shown in Section 8.1.1, a manifold simplicial $d$-complex $\Sigma'$ is the same as an MC-complex of dimension $d$. As a consequence, the MC-Decomposition of $\Sigma'$ consists of only one MC-component of dimension $d$, since there is not any non-manifold singularity. Clearly, the reverse is not true, since an MC-complex may contain any non-manifold singularity. For instance, the MC-Decomposition of the regular 2-complex in Figure 8.1(a) consists of only one MC-component of dimension 2, but it contains a non-manifold vertex $v$.

8.2 Computing the Manifold-Connected Decomposition

In this section, we describe two algorithms for computing the MC-Decomposition of non-manifold simplicial shapes, and relating MC-components and non-manifold singularities. Specifically, in Section 8.2.1, we prove that the MC-Decomposition is completely dimension-independent, and it can be computed for non-manifold simplicial shapes of any dimension. Conversely, in Section 8.2.2, we show that it is possible to design an algorithm for studying relations among MC-components and non-manifold singularities, which is restricted only to a subclass of simplicial complexes.

8.2.1 A Dimension-Independent Algorithm

In this section, we define a dimension-independent algorithm, which retrieves the MC-Decomposition for non-manifold simplicial shapes of any dimension.

Computation of the MC-Decomposition of $\Sigma$ starts by cutting it into regular subcomplexes $\Sigma_k$, with $1 \leq k \leq d$, formed by all the top $k$-simplices in $\Sigma$. Then, we compute the MC-Decomposition
of each subcomplex $\Sigma^t_k$. Thus, the challenge is to identify the MC-components of dimension $k$ in each $\Sigma^t_k$. These MC-components can be identified through a labeling algorithm.

We perform a traversal of each subcomplex $\Sigma^t_k$, starting from a not visited top $k$-simplex $\sigma$, and retrieve all the top $k$-simplices in $\Sigma^t_k$ which are reachable from $\sigma$ through a manifold $(k-1)$-path. In other words, this path is formed by top $k$-simplices connected through manifold $(k-1)$-simplices. All the top $k$-simplices $\sigma'$ reachable from $\sigma$ belong to the same MC-component, identified by an integer label $C$. Each top simplex $\sigma'$ and its faces are marked with the same label $C$. At the end of this traversal, all the top simplices in $\Sigma^t_k$ have been assigned to one MC-component.

This algorithm is completely dimension-independent, and, at each step, we do not check whether a simplex is manifold. In fact, we only check how many top $k$-simplices are incident at any $(k-1)$-simplex in order to navigate on a manifold $(k-1)$-path. In other words, for each $(k-1)$-simplex, we retrieve partial co-boundary relation $R^*_k$, restricted to top $k$-simplices, which we directly encode in our IA$^*$ data structure [CDFW11]. Thus, this test is always decidable for any $k$.

Thus, the problem of retrieving the MC-Decomposition is always decidable. As a consequence, the class $MCd$ is a decidable superclass of $Md$ without any restriction on the dimension of simplicial shapes to be decomposed. Hence, the MC-Decomposition is completely dimension-independent, and it can be defined and computed for non-manifold simplicial shapes of any dimension.

### 8.2.2 Manifold-Connected Components and Non-manifold Singularities

In this section, we provide an algorithm for studying relations among MC-components and non-manifold singularities. This algorithm can be applied only on a subclass of simplicial complexes, since it is based on the recognition of non-manifold singularities. Recall that the class $M_d$ of combinatorial manifold $d$-complexes is not decidable for $d \geq 6$ [Nab96].

Algorithm described in Section 8.2.1 provides MC-components for any non-manifold simplicial shape $\Sigma$. In any case, it is interesting to analyze relations among MC-components and non-manifold singularities in $\Sigma$. Specifically, this operation is necessary, for instance, to construct any graph-based representation for the MC-Decomposition of $\Sigma$.

At the end of this algorithm, a simplex $\sigma''$ in $\Sigma$, marked with several labels, is a non-manifold singularity. In any case, the number of MC-components incident at $\sigma''$ does not completely characterize $\sigma''$ as a non-manifold simplex. For instance, we cannot recognize non-manifold singularities, like non-manifold vertex $v$ in Figure 8.1(a). Here, we identify only an MC-component of dimension 2, and $v$ is marked by only one label, as other manifold vertices. In this case, it is mandatory to exploit an explicit recognition of non-manifold singularities.

Specifically, non-manifold singularities bound any top $k$-simplex $\sigma'$ belonging to a manifold $(k-1)$-path, and, thus, to one MC-component of dimension $k$. In this context, all the top $k$-simplices in any MC-component of dimension $k$ belong to the transitive closure of the $(k-1)$-adjacency relation along manifold $(k-1)$-faces, restricted to top $k$-simplices.
Algorithm 8.1 provides a pseudo-code description of this traversal. Here, we explicitly check if a simplex $\sigma''$ bounding a top $k$-simplex $\sigma'$ is manifold. As a consequence, this algorithm is not completely dimension-independent, and provides a not decidable solution for simplicial complexes of dimension $d \geq 6$ [Nab96]. In any case, it can be applied on the most common shapes, discretized by simplicial 2D and 3D complexes.

**Algorithm 8.1 RETRIEVE_MC_COMPONENTS($\Sigma$)**

**Input:** a simplicial $d$-complex $\Sigma$, described through any topological data structure

**Output:** the MC-components of $\Sigma$ and the array $L$ containing the collection $\Sigma_n$

```
1: let $L := \emptyset$
2: for all $k = 1, \ldots, \dim(\Sigma)$ do
3:   for all top $k$-simplex $\sigma$ in $\Sigma$ do
4:     if $\sigma$ is not visited then
5:       generate a new MC component associated with a new label $C$
6:       let $q$ be an empty queue
7:       enqueue $\sigma$ in $q$
8:       while $q$ is not empty do
9:         dequeue $\sigma'$ from $q$
10:        if $\sigma'$ is not visited then
11:          mark $\sigma'$ as visited and with label $C$
12:          add $\sigma'$ to the MC-component related to $C$
13:        for all $\sigma''$ in $b(\sigma')$ do
14:          if $\sigma''$ is not manifold in $\Sigma$ then
15:            $L[\sigma''] := L[\sigma''] \cup \{C\}$
16:          else if $\dim(\sigma'') = k - 1$ then
17:            enqueue $R^*_k,k(\sigma')$ along $\sigma''$ in $q$
18:        end if
19:      end for
20:    end if
21:  end while
22: end for
23: return $L$
```

In this algorithm we store non-manifold singularities in an array $L$ defined as follows: for a non-manifold simplex $\sigma''$, location $L[\sigma'']$ contains the labels of the MC-components incident at $\sigma''$.

It is clear that a simplicial $d$-complex $\Sigma$ must be represented through a topological data structure, which allows for an efficient navigation on top $k$-simplices, with $1 \leq k \leq d$. The traversal of each subcomplex $\Sigma'_k$ is a process which is linear in the number of top simplices in $\Sigma$, namely $s_k$. At each step, we analyze all the faces of any top $k$-simplex $\sigma'$, looking for non-manifold singularities.

Thus, the time complexity of Algorithm 8.1 depends on the time required for retrieving combinatorial boundary $b(\sigma')$ and partial adjacency relation $R^*_k,k$ of a top $k$-simplex $\sigma'$ (restricted to the top $k$-simplices adjacent to $\sigma'$), and on the time required for checking the manifold condition. In
the best case, the input simplicial $d$-complex is represented by a topological data structure such that the time complexity of these operations is optimal, i.e. $O(1)$, thus the time complexity of Algorithm 8.1 is $\Theta(S_\Sigma)$, where $S_\Sigma$ is the number of all the simplices in $\Sigma$. However, the time complexity of Algorithm 8.1, combined with a generic topological data structure, is $\Omega(S_\Sigma)$.

It is clear that some data structures are more suitable to be used in this algorithm than others. It is interesting to evaluate the time complexity of this algorithm, combined with the various data structures in our Mangrove TDS framework, discussed in Chapters 6 and 7. Note that, in these data structures, the combinatorial boundary of a simplex can be always extracted in $O(1)$. Thus, the retrieval of adjacency relation $R_{k,k}$, and the recognition of non-manifold simplices are critical.

As discussed in Section 5.1.1, partial adjacency relation $R^*_{k,k}$ is directly encoded in the I$A^*$ data structure, thus its time complexity is $O(1)$. Conversely, the time complexity required for recognizing non-manifold singularities depends on the complexity of non-manifold configurations. As shown in Section 6.3.6, given a simplex $\sigma$, the time complexity of the IS,MANIFOLD query may be $O(1)$ in the best case, and $O(\sigma^*_t)$ in the worst case, where $\sigma^*_t$ is the number of top simplices incident at $\sigma$. As a consequence, the time complexity of Algorithm 8.1, when combined with the I$A^*$ data structure, is $\Theta(S_\Sigma)$ in the best case, while, in the worst case, it is equal to:

$$O\left( s_d + \sum_{p=0}^{d-1} \sum_{\sigma \in \Sigma^p} \sigma^*_t \right)$$

where $\Sigma^p$ and $\sigma^*_t$ are, respectively, the collection of $p$-simplices in $\Sigma$, and the number of top simplices incident at $\sigma$. However, note that in most of shapes used in real applications, $\sigma^*_t$ may be bounded by a constant value, for any simplex $\sigma$. Thus, we can state that the time complexity tends to be linear in $S_\Sigma$. Note that we obtain the same results of the I$A^*$ data structure also with the TS and NMIA data structures. Recall that the TS and NMIA data structures are dimension-specific representations for simplicial 2- and 3-complexes embedded in the Euclidean space $E^3$, respectively. Conversely, the I$A^*$ data structure is a dimension-independent representation for abstract simplicial complexes.

As discussed in Section 6.2.6, in the IS data structure, the time complexity of the IS,MANIFOLD query is the same as the I$A^*$ data structure. However, given a top $k$-simplex $\sigma$, partial adjacency relation $R^*_k$ must be retrieved by combining $R_{k,k-1}$ and $R_{k,k-1}$ relations, thus it is local. As a consequence, the time complexity of Algorithm 8.1, when combined with the IS data structure, tends to be linear in $S_\Sigma$. Note that we obtain the same results of the IS data structure also with the SIG data structure.

Conversely, the IG data structure does not make the recognition of non-manifold simplices easy, since we need to retrieve all the connected components in the link of any simplex, as shown in Section 7.4.2. As a consequence, the time complexity of Algorithm 8.1, when combined with the IG data structure, is $\Omega(S_\Sigma)$.

In Section 8.4, we provide quantitative comparisons for all the data structures implemented in the Mangrove TDS Library, regarding efficiency of Algorithm 8.1. Our tests confirm the efficiency of
this algorithm, if combined with the IA\textsuperscript{*} data structure.

8.3 Graph-based Representations

In this section, we describe three graph-based data structures for representing the MC-Decomposition of any simplicial $d$-complex $\Sigma$. Two of these representations have been proposed in [CDF11].

Here, we represent the MC-Decomposition of any simplicial $d$-complex $\Sigma$ as a graph-based data structure, and we will show that it acts as an index on the data structure encoding the underlying simplicial complex. The most external level of these data structures consists of MC-components of $\Sigma$, while the internal level consists of a topological data structure describing $\Sigma$. Here, any MC-component $C$ of dimension $k$ (with $1 \leq k \leq d$) is described by a node, which consists of references to top $k$-simplices of $C$ in the input data structure. An arc $a$ describes the intersection of several MC-components, which consists of references to non-manifold singularities related to $a$ in the input data structure. Here, we assume that any simplex may be referred by a unique integer value in the input topological data structure.

In Section 8.3.1, we introduce the Exploded Manifold-Connected Graph (Exploded MC-Graph), while, in Section 8.3.2, we describe the Pairwise Manifold-Connected Graph (Pairwise MC-Graph). Finally, in Section 8.3.3, we introduce the Compact Manifold-Connected Graph (Compact MC-Graph). The first two graph-based data structures have been introduced in [CDF11].

Note that these graph-based data structures are defined on the same set $N_\Sigma$ of nodes, which corresponds to the collection of all the MC-components in a given simplicial $d$-complex $\Sigma$.

8.3.1 The Exploded Manifold-Connected Graph

In this section, we provide a complete description of the Exploded Manifold-Connected Graph (Exploded MC-Graph), which we have introduced in [CDF11].

The Exploded MC-Graph encodes the MC-Decomposition of any simplicial $d$-complex $\Sigma$ as a hyper-graph $G_E^{\Sigma} = (N_\Sigma, A_E^{\Sigma})$, where a node in $N_\Sigma$ corresponds to an MC-component in the MC-Decomposition of $\Sigma$, while an hyper-arc in $A_E^{\Sigma}$ corresponds to a non-manifold singularity $\sigma$ in $\Sigma$, and connects all the nodes which describe MC-components incident at $\sigma$.

Figure 8.4 shows the Exploded MC-Graph for a non-regular simplicial 3-complex, namely the “Flasks” shape [GGG09], which we have already shown in Figure 8.3(b). This screenshot is taken by the GCViewer program [DFF09], which provides a 3D representation of decompositions of simplicial shapes, embedded in the Euclidean space $E^3$. Each MC-component is represented by a point of $E^3$ corresponding to its center of gravity, while arcs among MC-components connect their centers of gravity. Blue squares along arcs correspond to the center of gravity of intersection complexes related to each arc.
Figure 8.4: (a) The Exploded MC-Graph for a non-regular simplicial 3-complex, namely (b) the “Flasks” shape [GGG09], which we have already shown in Figure 8.3(b). This screenshot is taken by the GCViewer program [DFPH09], which provides a 3D representation of decompositions of simplicial shapes, embedded in the Euclidean space $\mathbb{E}^3$. Each MC-component is represented by a point of $\mathbb{E}^3$ corresponding to its center of gravity, while arcs among MC-components connect their centers of gravity. Blue squares along arcs correspond to the center of gravity of intersection complexes related to each arc.

For each node in $\mathcal{N}_\Sigma$, corresponding to an MC-component $c$ of dimension $k$ in $\Sigma$, with $0 < k \leq d$, we encode:

- the dimension $k$ of all the top simplices belonging to $c$;
- $s_c$ references to all the top $k$-simplices belonging to $c$;
- $a^c_\Sigma$ references to all the hyper-arcs incident at the current node.

Thus, the storage cost of a node $c$ is equal to $1 + s_c + a^c_\Sigma$ integer values. In the following, we denote as $c$ both the node of the Exploded MC-Graph and the MC-component described by this node.

For each hyper-arc $a$ in $\mathcal{A}^E_\Sigma$, we encode:

- a reference to the non-manifold singularity $\sigma$ associated with $a$;
- $l_\sigma$ references to all the nodes describing MC-components incident at $\sigma$.

Thus, the storage cost of an hyper-arc $a$, related to a non-manifold singularity $\sigma$, is equal to $1 + l_\sigma$ integer values.

Let $n^E$ and $a^E$ be the numbers of nodes and hyper-arcs in the Exploded MC-Graph $\mathcal{G}^E_\Sigma$, respectively. Then, the storage cost $S_E$ of the Exploded MC-Graph is equal to:

$$S_E = n^E + a^E + S_\Sigma^E + \sum_{c \in \mathcal{N}_\Sigma} a^c_\Sigma + \sum_{\sigma \in \Sigma_n} l_\sigma$$

(8.1)
where $S^t$ and $\Sigma^n$ are, respectively, the total number of top simplices, and the set of non-manifold simplices in $\Sigma$.

Now, we can introduce an algorithm for retrieving the Exploded MC-Graph of any non-manifold simplicial shape $\Sigma$ in two steps. First, we apply Algorithm 8.1, which retrieves the MC-components in $\Sigma$, plus the connections among MC-components and non-manifold singularities in the array $L$. At the second step, we generate all the hyper-arcs in the Exploded MC-Graph from the content of the array $L$. Given a non-manifold simplex $\sigma$, each location $L[\sigma]$ contains $l_\sigma$ labels for all the MC-components incident at $\sigma$. As a consequence, we can directly generate an hyper-arc $a$ for each location $L[\sigma]$, where $a$ connects all the nodes corresponding to MC-components incident at $\sigma$. Thus, the time complexity of this step is $O(\sum_{\sigma\in \Sigma} l_\sigma)$. Note that the Step 2 of this algorithm does not depend on which topological data structure we are using.

Generally speaking, the time complexity required for constructing the Exploded MC-Graph is $\Omega(S^t + \sum_{\sigma\in \Sigma} l_\sigma)$, if combined with a generic data structure.

As discussed in Section 8.2.2, the time complexity of Step 1 depends on which topological data structure we use. Generally speaking, it is $\Omega(S^t)$, where $S^t$ is the number of all the simplices in $\Sigma$. However, it can be improved: for instance, it becomes $\Theta(S^t)$, if combined with the IA* data structure, introduced in Section 5. In this case, the time complexity required for constructing the Exploded MC-Graph becomes $O(S^t + \sum_{\sigma\in \Sigma} l_\sigma)$. Recall that we obtain the same results for simplicial 2- and 3-complexes, by combining the Exploded MC-Graph with the TS and NMIA data structures, respectively.

If a location $L[\sigma]$ contains only one label $C_i$, then we can generate a special kind of hyper-arc, which we call a self-loop arc. This arc connects twice the node corresponding to the MC-component identified by label $C_i$. Following [LDFH09], these arcs correspond to a squeezed configuration related to a non-manifold intrinsic connection. For instance, the Exploded MC-Graph for simplicial 2-complex in Figure 8.1(a) is formed by only one node with a self-loop arc, which corresponds to the non-manifold vertex $v$.

### 8.3.2 The Pairwise Manifold-Connected Graph

In this section, we provide a complete description of the Pairwise Manifold-Connected Graph (Pairwise MC-Graph), which we have introduced in [CDF11]. We have exploited the Pairwise MC-Graph in our Mayer-Vietoris (MV) algorithm for computing the homological information of a simplicial shape, which we will describe in Chapter 9.

The Pairwise MC-Graph encodes the MC-Decomposition of a simplicial $d$-complex $\Sigma$ as a graph $G^\Sigma_P = (N^\Sigma_P, A^\Sigma_P)$, where a node in $N^\Sigma_P$ corresponds to an MC-component in $\Sigma$, while an arc in $A^\Sigma_P$ describes the intersection between only two MC-components.

Figure 8.5 shows the Pairwise MC-Graph for a non-regular simplicial 3-complex, namely the “Flasks” shape [GGG09], which we have already shown in Figure 8.3(b).
Figure 8.5: (a) The Pairwise MC-Graph for a non-regular simplicial 3-complex, namely (b) the “Flasks” shape [GGG09], which we have already shown in Figure 8.3(b). Also this screenshot is taken by the GCViewer program [DFPH09], where exploit the same conventions used in Figure 8.4.

For each node in $N_\Sigma$, corresponding to an MC-component $c$ of dimension $k$ in $\Sigma$, we encode:

- the dimension $k$ of all the top simplices belonging to $c$;
- $s_c$ references to all the top $k$-simplices belonging to $c$;
- $a_c^p$ references to all the arcs incident at the current node.

Thus, the storage cost of a node corresponding to an MC-component $c$ is equal to $1 + s_c + a_c^p$ integer values. In the following, we denote as $c$ both the node of the Pairwise MC-Graph and the MC-component described by this node.

For each arc $a$ in $A^P_\Sigma$, we encode:

- two references to the MC-components $c_1$ and $c_2$ connected by arc $a$;
- $s_p^a$ references to singularities belonging to the intersection subcomplex between $c_1$ and $c_2$.

Thus, the storage cost of an arc $a$ connecting two nodes is equal to $2 + s_p^a$ integer values.

Let $a^P$ be the numbers of arcs in the Pairwise MC-Graph $G^P_\Sigma$. Recall that the number of nodes in this graph is equal to $n^E$, namely the number of nodes in the Exploded MC-Graph (see Section 8.3.1). Then, the storage cost $S_P$ of the Pairwise MC-Graph is equal to:

$$S_P = n^E + 2a^P + S^L_\Sigma + \sum_{c \in N_\Sigma} a_c^p + \sum_{a \in A_\Sigma^P} s_p^a$$  \hspace{1cm} (8.2)

where $S^L_\Sigma$ is the total number of top simplices in $\Sigma$.

Now, we can introduce an algorithm for retrieving the Pairwise MC-Graph of any non-manifold simplicial shape $\Sigma$ in two steps. First, we apply Algorithm 8.1, which retrieves the MC-components
in $\Sigma$, plus the connections among MC-components and non-manifold singularities in the array $L$.

As discussed in Section 8.2.2, the time complexity of Step 1 depends on which topological data structure we use. Generally speaking, it is $\Omega(S_{\Sigma})$, where $S_{\Sigma}$ is the number of all the simplices in $\Sigma$. However, it can be improved: for instance, it becomes $\Theta(S_{\Sigma})$, if combined with the IA$^*$ data structure, introduced in Section 5.1.

Recall that, given a non-manifold simplex $\sigma$, each location $L[\sigma]$ contains $l_{\sigma}$ identifiers $C_i$, with $0 \leq i < l_{\sigma}$, of all the MC-components which are incident at $\sigma$.

Arcs of the Pairwise MC-Graph are then retrieved as follows, independently on which topological data structure we are using:

1. for each non-manifold singularity $\sigma$, consider the number of labels in $L[\sigma]$. If there are at least two labels, sort them in increasing order, generate all the possible pairs $(C_i, C_j)$ formed by elements in $L[\sigma]$, and store tuples $(\sigma, C_i, C_j)$ in an array $B$. Otherwise, we generate a tuple $(\sigma, C_i, C_i)$, if $C_i$ is the unique label in $L[\sigma]$.

2. Sort the tuples in $B$ by using the lexicographic order on the pairs $(C_i, C_j)$: tuples related to the same pair of labels are stored in consecutive locations of $B$, and correspond to the intersection between a pair of MC-complexes.

3. Create a new arc for each unique pair of nodes identified in step 2, and complete all the missing data in the involved nodes and arcs.

In Step 1, for each non-manifold singularity $\sigma$, we generate all the possible $l_{\sigma}$ intersections among labels in $L[\sigma]$, namely $l_{\sigma} = l_{\sigma}/(l_{\sigma} - 2)!$. The time complexity of this algorithm is dominated by Step 2, where all the elements of $B$ are sorted, thus the time complexity is $O(\sum_{\sigma \in \Sigma} l_{\sigma} \log l_{\sigma})$, where $\Sigma_n$ is the set of non-manifold simplices in $\Sigma$.

As a consequence, the time complexity of this algorithm is $\Omega(S_{\Sigma} + \sum_{\sigma \in \Sigma} l_{\sigma} \log l_{\sigma})$, including Step 1. Note that it becomes $O(S_{\Sigma} + \sum_{\sigma \in \Sigma} l_{\sigma} \log l_{\sigma})$, if we exploit our IA$^*$ data structure, as discussed in Section 8.2.2. Recall that we obtain the same results for simplicial 2- and 3-complexes, by combining the Pairwise MC-Graph with the TS and NMIA data structures, respectively.

Also in the Pairwise MC-Graph, there may be self-loop arcs, which are related to pinched configurations (see Section 8.3.1). In this case, we identify these arcs at Step 1, when we analyze a location $L[\sigma]$ containing only one label.

### 8.3.3 The Compact Manifold-Connected Graph

In this section, we introduce the Compact Manifold-Connected Graph (Compact MC-Graph), which is more compact and robust than the other graph-based data structures, discussed, respectively, in Sections 8.3.1 and 8.3.2.
The Compact MC-Graph encodes the MC-Decomposition of any simplicial $d$-complex $\Sigma$ as a hyper-graph $G^C = (N^C, A^C)$, where a node in $N^C$ corresponds to an MC-component in the MC-Decomposition of $\Sigma$, while a hyper-arc $a$ in $A^C$ corresponds to the maximal set of singularities in $\Sigma$, which are shared by a subset of MC-components $C_1, \ldots, C_k$. In other words, an hyper-arc $a = (C_1, \ldots, C_k)$ satisfies the following properties:

i) the intersection of MC-complexes connected by $a$, namely $\Sigma^a = \bigcap_{i=1}^{k} C_i$, is a subcomplex, not necessarily connected, of $\Sigma$;

ii) it does not exist a MC-component $C_s$ of $\Sigma$, with $C_s \neq C_i$, for all $i = 1, \ldots, k$, such that $\Sigma^a = \left( \bigcap_{i=1}^{k} C_i \right) \cap C_s$.

Figure 8.6(a) shows a non-manifold simplicial 2-complex formed by three MC-components of dimension 2, namely $C_1, C_2, C_3$, plus an MC-complex $C_4$ of dimension 1. Its MC-Decomposition can be represented through the related Compact MC-Graph, shown in Figure 8.6(b). Note that there are two hyper-arcs. An hyper-arc connects MC-components $(C_1, C_2, C_3)$ and it is related to non-manifold singularities $v_1$ and $e_1$. The other hyper-arc connects MC-components $(C_1, C_2, C_3, C_4)$ and it is related to non-manifold vertex $v_2$.

For each node in $N^C$, corresponding to an MC-component $c$ of dimension $k$ in $\Sigma$, we encode:

- the dimension $k$ of all the top simplices belonging to $c$;
- $s_c$ references to all the top $k$-simplices belonging to $c$;
- $a^c$ references to all the hyper-arcs incident at the current node.

Thus, the storage cost of a node corresponding to an MC-component $c$ is $1 + s_c + a^c$ integer values.

In the following, we denote as $c$ both the node of the Compact MC-Graph and the MC-component described by this node.

For each hyper-arc $a$ in $A^C$, we encode:
• $l_a$ references to MC-components which share the intersection subcomplex related to arc $a$;

• $s'^a_a$ references to singularities belonging to intersection subcomplex described by arc $a$.

Thus, the storage cost of an hyper-arc $a$ is equal to $l_a + s'^a_a$ integer values.

Let $a^C$ be the number of arcs in the Compact MC-Graph $G^C$. Recall that the number of nodes in this graph is equal to $n^E$, namely the number of nodes in the Exploded MC-Graph (see Section 8.3.1). Then, the storage cost $S_C$ of the Compact MC-Graph is equal to:

$$S_C = n^E + 2a^C + \sum_{a \in A^E} s'^a_a + \sum_{c \in \mathcal{N}^E} a^c_C$$

where $S^E_C$ is the total number of top simplices in $\Sigma$.

Now, we can introduce an algorithm for retrieving the Compact MC-Graph of any non-manifold simplicial shape $\Sigma$ in two steps. The key idea is to combine together hyper-arcs of the Exploded MC-Graph, which connect to the same subset of MC-components, like shown in Figure 8.6(c). First, we apply Algorithm 8.1, which retrieves MC-components in $\Sigma$, plus connections among MC-components and non-manifold singularities in the array $L$, as performed while constructing the Pairwise MC-Graph (see Section 8.3.2). Then, arcs of the Compact MC-Graph are retrieved as follows, independently of which topological data structure we are using:

1. for each non-manifold singularity $\sigma$, consider the number of labels stored in $L[\sigma]$. If there are at least two labels, sort them in increasing order as a new list $l_{\sigma}$ of labels, and store the tuples $(\sigma, l_{\sigma})$ in an array $B$. Otherwise, we generate a tuple $(\sigma, C_i, C_i)$, if $C_i$ is the unique label in $L[\sigma]$.

2. Sort the tuples in $B$ with respect to the corresponding list of labels. Given two lists $l$ and $l'$ of labels (corresponding to the MC-components), we say that $l < l'$ if and only if either the size of $l$ is smaller than the size of $l'$, or their size is the same and $l < l'$ with respect to the lexicographic order of their labels. Tuples related to the same subset $S$ of MC-components are stored in consecutive locations of $B$, and correspond to singularities shared by MC-components in $S$.

3. Create a new arc for each unique subset of MC-components identified in Step 2, and complete all the missing data in the involved nodes and arcs.

In Step 1, for each non-manifold singularity $\sigma$, we sort the $l_{\sigma}$ labels for all the MC-components incident at $\sigma$ (see Section 8.3.1), and generate a new tuple. The time complexity of this operation is $O(l_{\sigma} \log l_{\sigma})$, for each non-manifold singularity. In Step 2, we sort all the $S^\Sigma_2$ tuples in $B$, one for each location $L[\sigma]$, and thus for each non-manifold singularity $\sigma$ in $\Sigma$. Hence, the time complexity of this step is $O(S^\Sigma_2 \log S^\Sigma_2)$, where $S^\Sigma_2$ is the number of non-manifold singularities in $\Sigma$. Note that the lists of labels in these tuples do not have necessarily the same number of elements. The time
complexity required for identifying the hyper-arcs in the Compact MC-Graph is:

$$\mathcal{O}\left( S^n_\Sigma \log S^n_\Sigma + \sum_{\sigma \in \Sigma_n} l_\sigma \log l_\sigma \right)$$

As a consequence, the time complexity required for constructing the Compact MC-Graph is $\Omega(S^n_\Sigma + S^n_\Sigma + \sum_{\sigma \in \Sigma_n} l_\sigma \log l_\sigma)$, including the identification of MC-components (see Algorithm 8.1). Note that it becomes:

$$\mathcal{O}\left( S^n_\Sigma + S^n_\Sigma + S^n_\Sigma \log S^n_\Sigma + \sum_{\sigma \in \Sigma_n} l_\sigma \log l_\sigma \right)$$

if we exploit our IA$^*$ data structure, as discussed in Section 8.2.2. Recall that we obtain the same results for simplicial 2- and 3-complexes, by combining the Pairwise MC-Graph with the TS and NMIA data structures, respectively.

Also in the Compact MC-Graph, there may be self-loop arcs, which are related to pinched configurations (see Section 8.3.1). In this case, we identify these arcs at Step 1, when we analyze a location $L[\sigma]$ containing only one label.

### 8.4 Experimental Comparisons

In this section, we present qualitative and quantitative results about the MC-Decomposition and our graph-based data structures, discussed in Section 8.3. Specifically, we provide experimental comparisons for each graph-based representation, combined with all the topological data structures implemented in our Mangrove Topological Data Structure (Mangrove TDS) framework, discussed in Chapters 6 and 7.

All the digital shapes used in our tests are freely available [GGG09]. We tested our implementations on a workstation with 1.8 Ghz Intel®Core 2 Duo processor and 3 Gb of RAM. All the timings presented in this section are expressed in milliseconds, and are retrieved through a standard timer in the platform-independent QT Library® [QT08].

First, we show experimentally two of the most important properties of the MC-Decomposition. Specifically, we analyze the size of MC-components in the MC-decomposition, and the size of the intersection complexes, corresponding to singularities shared by two MC-components. Table 8.1 shows our results on a subset of simplicial shapes we have already analyzed in Chapters 4 and 5.

We evaluate the size $MS_c$ of the largest MC-component (including top simplices and their faces), and the size $MA_a$ of the largest intersection between only two MC-components. Our tests show that $MA_a$ is very small with respect to the number $S_\Sigma$ of simplices in the input shape: specifically, $MA_a$ does not exceed 5% of $S_\Sigma$ in most of tests. Also, the maximum value of $MS_c$ is 97% of $S_\Sigma$ (see the “Arc” 3D shape), and it is clearly smaller than $S_\Sigma$. It is also clear that we do not impose any limitation on the size of MC-components, which is extremely variable, as shown by our tests. For instance, in manifold shapes, or in shapes with only pinched configurations (like the
“Cylinders” shape), an MC-component can be as large as the input shape. Conversely, it can be formed by only one vertex, like in the “Sierpinski” shape. We can also deduce that $S^t_c$ is about 30% for simplicial 2-complexes, while $S^t_c$ is about 20% for simplicial 3-complexes.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$S^t_c$</th>
<th>$S^c_c$</th>
<th>$S^c_n$</th>
<th>$n^E$</th>
<th>$a^E$</th>
<th>$a^P$</th>
<th>$a^C$</th>
<th>$M S_a$</th>
<th>$M A_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armchair</td>
<td>31.8k</td>
<td>10.6k</td>
<td>113</td>
<td>12</td>
<td>113</td>
<td>17</td>
<td>12</td>
<td>38.4%</td>
<td>0.07%</td>
</tr>
<tr>
<td>800-Cubes</td>
<td>22.5k</td>
<td>9.6k</td>
<td>5.7k</td>
<td>2.8k</td>
<td>5.7k</td>
<td>36.2k</td>
<td>4.8k</td>
<td>0.17%</td>
<td>0.04%</td>
</tr>
<tr>
<td>Cylinders</td>
<td>590</td>
<td>200</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>100%</td>
<td>0.8%</td>
</tr>
<tr>
<td>Pinched-pie</td>
<td>5.9k</td>
<td>2.3k</td>
<td>936</td>
<td>120</td>
<td>936</td>
<td>1.36k</td>
<td>192</td>
<td>2.7%</td>
<td>0.55%</td>
</tr>
<tr>
<td>Twist</td>
<td>7.1k</td>
<td>2.4k</td>
<td>56</td>
<td>4</td>
<td>56</td>
<td>5</td>
<td>2</td>
<td>65%</td>
<td>0.8%</td>
</tr>
<tr>
<td>Robot</td>
<td>19.8k</td>
<td>6.7k</td>
<td>916</td>
<td>45</td>
<td>916</td>
<td>65</td>
<td>64</td>
<td>79.5%</td>
<td>0.48%</td>
</tr>
<tr>
<td>Balance</td>
<td>24k</td>
<td>8k</td>
<td>8</td>
<td>10</td>
<td>8</td>
<td>12</td>
<td>8</td>
<td>31.4%</td>
<td>0.004%</td>
</tr>
<tr>
<td>Carter</td>
<td>23.8k</td>
<td>7.9k</td>
<td>641</td>
<td>45</td>
<td>641</td>
<td>79</td>
<td>48</td>
<td>40.6%</td>
<td>0.49%</td>
</tr>
<tr>
<td>Chandelier</td>
<td>24k</td>
<td>8.1k</td>
<td>616</td>
<td>130</td>
<td>616</td>
<td>328</td>
<td>96</td>
<td>11.8%</td>
<td>0.11%</td>
</tr>
<tr>
<td>Frame</td>
<td>4.3k</td>
<td>1.3k</td>
<td>165</td>
<td>132</td>
<td>165</td>
<td>285</td>
<td>156</td>
<td>8%</td>
<td>0.05%</td>
</tr>
<tr>
<td>Tower</td>
<td>55.2k</td>
<td>18.6k</td>
<td>1.4k</td>
<td>169</td>
<td>1.4k</td>
<td>12.9k</td>
<td>165</td>
<td>56.8%</td>
<td>0.6%</td>
</tr>
<tr>
<td>Tower-wir</td>
<td>48.8k</td>
<td>16.8k</td>
<td>2.1k</td>
<td>970</td>
<td>2.1k</td>
<td>10.7k</td>
<td>649</td>
<td>75%</td>
<td>0.52%</td>
</tr>
<tr>
<td>Arc</td>
<td>1k</td>
<td>171</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>97%</td>
<td>0.2%</td>
</tr>
<tr>
<td>Balloon</td>
<td>9.5k</td>
<td>2.5k</td>
<td>48</td>
<td>66</td>
<td>48</td>
<td>192</td>
<td>48</td>
<td>2.9k</td>
<td>0.02%</td>
</tr>
<tr>
<td>Bucket</td>
<td>428</td>
<td>86</td>
<td>38</td>
<td>7</td>
<td>38</td>
<td>11</td>
<td>7</td>
<td>75.5%</td>
<td>7.5%</td>
</tr>
<tr>
<td>Chime</td>
<td>2.4k</td>
<td>376</td>
<td>29</td>
<td>27</td>
<td>29</td>
<td>47</td>
<td>28</td>
<td>32.4%</td>
<td>0.12%</td>
</tr>
<tr>
<td>Flasks</td>
<td>19.6k</td>
<td>3.96k</td>
<td>76</td>
<td>8</td>
<td>76</td>
<td>10</td>
<td>6</td>
<td>47.2%</td>
<td>0.16%</td>
</tr>
<tr>
<td>Halves</td>
<td>4.2k</td>
<td>0.8k</td>
<td>19</td>
<td>2</td>
<td>19</td>
<td>1</td>
<td>1</td>
<td>50.3%</td>
<td>0.45%</td>
</tr>
<tr>
<td>Sierpinski</td>
<td>213k</td>
<td>16.4k</td>
<td>32.76k</td>
<td>16.4k</td>
<td>32.76k</td>
<td>32.76k</td>
<td>32.76k</td>
<td>0.007%</td>
<td>0.0005%</td>
</tr>
<tr>
<td>Teapot</td>
<td>45.2k</td>
<td>12.5k</td>
<td>11.5k</td>
<td>2.9k</td>
<td>1.2k</td>
<td>18k</td>
<td>1k</td>
<td>65%</td>
<td>0.28%</td>
</tr>
<tr>
<td>Wheel</td>
<td>6.3k</td>
<td>1.2k</td>
<td>136</td>
<td>115</td>
<td>136</td>
<td>520</td>
<td>88</td>
<td>50.2%</td>
<td>0.05%</td>
</tr>
</tbody>
</table>

Table 8.1: Statistics about our graph-based data structures, which represent the MC-Decomposition of simplicial shapes containing $S^t_c$ top simplices and $S^c_n$ non-manifold simplices. Their corresponding Exploded, Pairwise, and Compact MC-Graph have $n^E$ nodes, and, respectively, $a^E$, $a^P$, and $a^C$ arcs. It is interesting to compare the size $M S_a$ of the largest MC-component (including top simplices and their faces) and size $M A$ of the largest intersection between two MC-components, both expressed as a percentage of the total number of simplices $S^t_c$ in the input shape. The first group of shapes is formed by simplicial 2-complexes, while the second one by simplicial 3-complexes. Here, we do not report results on manifold shapes.

Now, we evaluate the storage costs of our graph-based data structures, which represent the MC-Decomposition of a simplicial $d$-complex $\Sigma$. Table 8.2 summarizes the storage costs $S_C$, $S_P$, and $S_E$, of, respectively, the Compact, Pairwise, and Exploded MC-Graphs. Our tests show that, on average, $S_P \approx 1.7 \times S_C$, and $S_E$ is about 4% larger than $S_P$, in most of tests, although there are special cases, which we will analyze in the remainder of this section. In any case, we can state that $S_C < S_P < S_E$.

However, in some situations, the Exploded MC-Graph may become more compact than the Pairwise MC-Graph, for instance in the “800-Cubes” 2D shape, or in the “Teapot” 3D shape, just to mention few. Thus, it is interesting to analyze their behaviors. Both the graph-based data structures encode references to top simplices, and this does not give a contribution to their differ-
Table 8.2: Statistics about storage costs of our graph-based data structures representing the MC-Decomposition of simplicial 2- and 3-complexes. Specifically, we denote storage costs of the Compact, Pairwise, and Exploded MC-Graphs as, respectively, $S_C$, $S_P$, and $S_E$. Also, the minimum, average, and maximum number of MC-components incident at a non-manifold singularity $\sigma$ are denoted, respectively, as $l_{\sigma}^m$, $l_{\sigma}^a$, and $l_{\sigma}^M$.

In the Exploded MC-Graph, we generate only one hyper-arc related to $\sigma$, which connects $l_{\sigma}$ MC-components incident at $\sigma$. Conversely, in the Pairwise MC-Graph, we generate all the possible $l_{\sigma}$ intersections among MC-components incident at $\sigma$, namely $\overline{l_{\sigma}} = l_{\sigma}/(l_{\sigma} - 2)!$. Clearly, if $l_{\sigma} > 2$, we generate a large number of arcs in the Pairwise MC-Graph, increasing its storage cost $S_P$. For instance, this happens with a vertex belonging to a wire-web, namely a 0-connected maximal
component of top edges. The Tower and “Teapot” shapes contain a wire-web, and $S_P \approx 3.18 \times S_E$ and $S_P \approx 4 \times S_E$, respectively. In this case, $l_σ^M = 160$ and $l_σ^M = 64$, respectively. Moreover, this situation becomes more interesting if the same set of non-manifold singularities is shared by more than two MC-components, like the “Pinched-pie” and “800-Cubes” shapes, where $l_σ = 4.08$ and $l_σ = 6.4$, respectively. In other words, the Pairwise MC-Graph tends easily to be strongly-connected. Conversely, if $l_σ \approx 2$ for each non-manifold singularity $σ$, singularities can be grouped in connected components (in this case chains of non-manifold edges), which describe intersection of only two MC-components. This situation happens with the “Flasks” shape, where hyper-arcs of its Exploded MC-Graph, shown in Figure 8.4(a), are “grouped” in one arc of the Pairwise MC-Graph, shown in Figure 8.5(a). Moreover, in the “Sierpinski” shape, all the MC-components consist of one tetrahedron, which shares a non-manifold vertex with an other tetrahedron. All the singularities are non-manifold vertices, thus our graph-based data structures coincide.

Our tests also show that the Compact MC-Graph does not depend on the complexity of non-manifold singularities. Note that it is always more compact than the other graph-based representations. Compact MC-Graph minimizes the number of redundancies regarding the encoding of intersection subcomplexes. For instance, we consider a 2D shape formed by two hemispheres connected through a disk, as shown in Figure 8.7(a). The MC-Decomposition of this shape is formed by three MC-components, connected through non-manifold vertices and edges bounding the disk (in black). As shown in Figure 8.7(b), an MC-component is one of hemispheres (in green), while the other two MC-components are, respectively, the common disk (in black) and the second hemisphere (in blue), as shown in Figure 8.7(c). In this case, the Pairwise MC-Graph consists of three nodes, one for each MC-component, and three arcs, one for each pair of intersections between MC-components. Non-manifold vertices and edges bounding the disk, are encoded for each arc, thus three times. Conversely, the Compact MC-Graph consists of three nodes, and only one hyper-arc, which connects all the nodes, and it is related to non-manifold singularities shared by the three MC-components.

![Figure 8.7](image)

Figure 8.7: (a) A 2D shape formed by two hemispheres connected by a disk. Its MC-Decomposition is formed by three MC-components, namely (a) a hemisphere (in green), and (b) a disk (in black), and an other hemisphere (in blue). Figures courtesy of [HDF07a].

Now, we show that our graph-based data structures are effective tools for representing a non-manifold simplicial shape. We concentrate our attention on the Compact MC-Graph, since it
is robust with respect to the complexity of non-manifold connections. Specifically, we combine the Compact MC-Graph with all the topological data structures implemented in the Mangrove TDS framework. In this way, we decouple the representation of the MC-components, provided by any topological data structure, from the structural model of the input shape, provided by the MC-Decomposition. In this way, we obtain a two-level representation, which explicitly exposes singularities, and the connectivity of MC-components. In any case, this two-level representation is more expensive than a topological data structure.

Recall that we have discussed the storage costs of our topological data structures in Sections 4.4 and 5.4. Specifically, we denote the resulting storage cost of the Compact MC-Graph, combined with the IA* and TS (only for simplicial 2-complexes) data structures, as $S_{IA*}^C$ and $S_{TS}^C$, respectively. Storage costs obtained with the NMIA (only for simplicial 3-complexes) and IS data structures, are denoted as $S_{NM}^C$ and $S_{IS}^C$, respectively. Finally, the storage cost of the Compact MC-Graph, combined with the SIG data structure, is denoted as $S_{SIG}^C$. We denote storage cost of the IG data structure as $S_{IG}$. Table 8.3 summarizes our comparisons.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$S_{IA*}^C$</th>
<th>$S_{TS}^C$</th>
<th>$S_{NM}^C$</th>
<th>$S_{IS}^C$</th>
<th>$S_{SIG}^C$</th>
<th>$S_{IG}$</th>
<th>$S_C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armchair</td>
<td>80k</td>
<td>80.2k</td>
<td>-</td>
<td>111.6k</td>
<td>111.6k</td>
<td>127.2k</td>
<td>10.9k</td>
</tr>
<tr>
<td>800-Cubes</td>
<td>138.4k</td>
<td>153.3</td>
<td>-</td>
<td>144.5k</td>
<td>144.5k</td>
<td>140.4k</td>
<td>63.4k</td>
</tr>
<tr>
<td>Cylinders</td>
<td>1.51k</td>
<td>1.53k</td>
<td>-</td>
<td>2.1k</td>
<td>2.1k</td>
<td>2.4k</td>
<td>0.2k</td>
</tr>
<tr>
<td>Pinched-pie</td>
<td>21.6k</td>
<td>23.5k</td>
<td>-</td>
<td>25.5k</td>
<td>25.5k</td>
<td>25.4k</td>
<td>5k</td>
</tr>
<tr>
<td>Twist</td>
<td>18k</td>
<td>18.1k</td>
<td>-</td>
<td>24.9k</td>
<td>24.9k</td>
<td>28.4k</td>
<td>2.4k</td>
</tr>
<tr>
<td>Robot</td>
<td>52.9k</td>
<td>54.4k</td>
<td>-</td>
<td>71.5k</td>
<td>71.5k</td>
<td>80.2k</td>
<td>8k</td>
</tr>
<tr>
<td>Balance</td>
<td>59.9k</td>
<td>59.9k</td>
<td>-</td>
<td>83.9k</td>
<td>83.9k</td>
<td>96k</td>
<td>8k</td>
</tr>
<tr>
<td>Carter</td>
<td>66.3k</td>
<td>67.3k</td>
<td>-</td>
<td>89.2k</td>
<td>89.2k</td>
<td>95k</td>
<td>13.9k</td>
</tr>
<tr>
<td>Chandelier</td>
<td>139.9k</td>
<td>140.7k</td>
<td>-</td>
<td>194.1k</td>
<td>194.1k</td>
<td>220.2k</td>
<td>19.6k</td>
</tr>
<tr>
<td>Frame</td>
<td>10.3k</td>
<td>10.3k</td>
<td>-</td>
<td>14.3k</td>
<td>14.3k</td>
<td>15.4k</td>
<td>2.2k</td>
</tr>
<tr>
<td>Tower</td>
<td>142.9k</td>
<td>144.8k</td>
<td>-</td>
<td>196.2k</td>
<td>196.2k</td>
<td>221.2k</td>
<td>21k</td>
</tr>
<tr>
<td>Tower-wir</td>
<td>132.06k</td>
<td>135.06k</td>
<td>-</td>
<td>177.66k</td>
<td>177.66k</td>
<td>193.8k</td>
<td>23.36k</td>
</tr>
</tbody>
</table>

Table 8.3: Storage costs of the Compact MC-Graph combined with our topological data structures, implemented in the Mangrove TDS framework. Specifically, we denote the resulting storage cost of the Compact MC-Graph, combined with the IA* and TS (only for simplicial 2-complexes) data structures, as $S_{IA*}^C$ and $S_{TS}^C$, respectively. Storage costs obtained with the NMIA (only for simplicial 3-complexes) and IS data structures, are denoted as $S_{NM}^C$ and $S_{IS}^C$, respectively. Finally, storage cost of the Compact MC-Graph, combined with the SIG data structure, is denoted as $S_{SIG}^C$. We denote storage cost of the IG data structure as $S_{IG}$. 

210
Our results show that, in most cases, the Compact MC-Graph, combined with our topological data structures, provides a representation, which is more compact than the Incidence Graph. The “800-Cubes” shape does not satisfy this property, since $S_{IA^*}^C$ is about 38% larger than $S_{IG}$. Also, the “Pinched-pie” shape does not partially satisfy this property, since $S_{IS}^C$ is less than 1% larger than $S_{IG}$. For simplicial 2-complexes, our tests show that, on average, $S_{IG} \approx 1.44 \times S_{IA^*}^C$, while $S_{IS}^C$ is about 7% smaller than $S_{IG}$. Conversely, for simplicial 3-complexes, $S_{IG} \approx 2.6 \times S_{IA^*}^C \approx 1.26 \times S_{IS}^C$.

Now, we can evaluate running times needed to construct our graph-based data structures representing the MC-Decomposition of a simplicial $d$-complex $\Sigma$. This operation can be performed in two steps. In the first step, we exploit Algorithm 8.1, discussed in Section 8.2.2, which depends on the topological data structure we are using. In the second step, we identify arcs in the Exploded, Pairwise, and Compact MC-Graphs by exploiting algorithms described in Section 8.3. These algorithms do not depend on the topological data structure we are using.

Table 8.4 summarizes our results regarding execution times of the construction algorithms for our graph-based data structures, combined with all the data structures discussed in Chapters 4 and 5. Specifically, we denote running times of Algorithm 8.1, combined with the IA$^*$ and TS (only for simplicial 2-complexes) data structures, as $I_{IA^*}$ and $I_{TS}$, respectively. Also, running times, obtained with the NMIA (only for simplicial 3-complexes) and IS data structures, are denoted as $I_{NM}$ and $I_{IS}$, respectively. Finally, running times, obtained with the SIG and IG data structures, are denoted as $I_{SIG}$ and $I_{IG}$, respectively. Moreover, we denote the running times of algorithms which identify arcs in the Exploded, Pairwise, and Compact MC-Graphs as $I_E$, $I_P$, and $I_C$, respectively.

These results strictly depend on the performances of topological data structures, which we have analyzed in Section 7.5. Specifically, we concentrate our attention on the retrieval of the combinatorial boundary $b(\sigma)$, and partial adjacency relation $R_{k,k}^*(\sigma)$, for any top $k$-simplex $\sigma$, plus the recognition of non-manifold singularities.

Our tests show that the IA$^*$ data structure is the most suitable data structure for supporting the retrieval of the MC-Decomposition. In our implementation of the IA$^*$ data structure, described in Section 6.3, the combinatorial boundary of a simplex can be efficiently retrieved without a traversal of the IA$^*$ boundary graph (see Section 7.5.1), partial adjacency relation $R_{k,k}^*$ is directly encoded, and the recognition of non-manifold singularities can be efficiently performed (see Section 7.5.5).

The main differences between the IA$^*$ and the TS data structures (only for simplicial 2-complexes), and between the IA$^*$ and NMIA data structures (only for simplicial 3-complexes) consist of the different encoding of non-manifold adjacency, related to partial adjacency relation $R_{k,k}^*$ (see Sections 5.2 and 5.3, respectively) with $1 < k \leq 3$. In the IA$^*$ data structure, this adjacency relation is encoded only once, while it can be retrieved by visiting several records in the TS and NMIA data structures. Our tests show that, on average, $I_{TS}$ is 18% larger than $I_{IA^*}$, and $I_{NM} \approx 1.53 \times I_{IA^*}$. Note that $I_{TS} \approx 2.17 \times I_{IA^*} > I_{SIG}$ with the “800-Cubes” 2D shape, since the number of records related to partial co-boundary relation is quite large, as already discussed in Section 5.4.1. Note
Table 8.4: Running times, expressed in milliseconds, of algorithms which retrieve our graph-based representations of the MC-Decomposition. Specifically, we denote running times of Algorithm 8.1, combined with the IA∗ and TS (only for simplicial 2-complexes) data structures, as \(I_{IA^*}\) and \(I_{TS}\), respectively. Also, running times, obtained with the NMIA (only for simplicial 3-complexes) and IS data structures, are denoted as \(I_{NM}\) and \(I_{IS}\), respectively. Finally, running times, obtained with the SIG and IG data structures, are denoted as \(I_{SIG}\) and \(I_{IG}\), respectively. Moreover, we denote the running times of algorithms which identify arcs in the Exploded, Pairwise, and Compact MC-Graphs as \(I_E\), \(I_P\), and \(I_C\), respectively.

<table>
<thead>
<tr>
<th>Shape</th>
<th>(I_{IA^*})</th>
<th>(I_{TS})</th>
<th>(I_{NM})</th>
<th>(I_{IS})</th>
<th>(I_{SIG})</th>
<th>(I_{IG})</th>
<th>(I^E)</th>
<th>(I^P)</th>
<th>(I^C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Armchair</td>
<td>4010</td>
<td>4030</td>
<td>–</td>
<td>6324</td>
<td>10789</td>
<td>179650</td>
<td>1155</td>
<td>1155</td>
<td>1350</td>
</tr>
<tr>
<td>800-Cubes</td>
<td>1412</td>
<td>3064</td>
<td>–</td>
<td>1686</td>
<td>2349</td>
<td>22029</td>
<td>367</td>
<td>386</td>
<td>451</td>
</tr>
<tr>
<td>Cylinders</td>
<td>50</td>
<td>52</td>
<td>–</td>
<td>62</td>
<td>90</td>
<td>1233</td>
<td>4</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Pinched-pie</td>
<td>260</td>
<td>385</td>
<td>–</td>
<td>411</td>
<td>520</td>
<td>6780</td>
<td>58</td>
<td>71</td>
<td>83</td>
</tr>
<tr>
<td>Twist</td>
<td>621</td>
<td>678</td>
<td>–</td>
<td>709</td>
<td>1169</td>
<td>18526</td>
<td>66</td>
<td>69</td>
<td>81</td>
</tr>
<tr>
<td>Robot</td>
<td>1622</td>
<td>1762</td>
<td>–</td>
<td>2643</td>
<td>4579</td>
<td>63064</td>
<td>182</td>
<td>183</td>
<td>214</td>
</tr>
<tr>
<td>Balance</td>
<td>2440</td>
<td>2552</td>
<td>–</td>
<td>4093</td>
<td>7157</td>
<td>108075</td>
<td>9</td>
<td>22</td>
<td>35</td>
</tr>
<tr>
<td>Carter</td>
<td>4264</td>
<td>4569</td>
<td>–</td>
<td>7689</td>
<td>13451</td>
<td>187490</td>
<td>17</td>
<td>23</td>
<td>37</td>
</tr>
<tr>
<td>Chandelier</td>
<td>8074</td>
<td>8090</td>
<td>–</td>
<td>15329</td>
<td>27651</td>
<td>414758</td>
<td>106</td>
<td>125</td>
<td>200</td>
</tr>
<tr>
<td>Frame</td>
<td>212</td>
<td>217</td>
<td>–</td>
<td>283</td>
<td>431</td>
<td>5367</td>
<td>9</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>Tower</td>
<td>8125</td>
<td>8508</td>
<td>–</td>
<td>15286</td>
<td>24667</td>
<td>440840</td>
<td>49</td>
<td>122</td>
<td>195</td>
</tr>
<tr>
<td>Tower-wir</td>
<td>5776</td>
<td>6682</td>
<td>–</td>
<td>11384</td>
<td>18942</td>
<td>293723</td>
<td>146</td>
<td>270</td>
<td>432</td>
</tr>
<tr>
<td>Arc</td>
<td>99</td>
<td>–</td>
<td>168</td>
<td>197</td>
<td>352</td>
<td>12989</td>
<td>30</td>
<td>40</td>
<td>55</td>
</tr>
<tr>
<td>Balloon</td>
<td>995</td>
<td>–</td>
<td>1467</td>
<td>4847</td>
<td>6914</td>
<td>116748</td>
<td>30</td>
<td>43</td>
<td>60</td>
</tr>
<tr>
<td>Bucket</td>
<td>35</td>
<td>–</td>
<td>66</td>
<td>84</td>
<td>140</td>
<td>3064</td>
<td>10</td>
<td>13</td>
<td>19</td>
</tr>
<tr>
<td>Chime</td>
<td>219</td>
<td>–</td>
<td>354</td>
<td>516</td>
<td>615</td>
<td>19195</td>
<td>14</td>
<td>20</td>
<td>28</td>
</tr>
<tr>
<td>Flasks</td>
<td>2421</td>
<td>–</td>
<td>4253</td>
<td>6704</td>
<td>10139</td>
<td>383473</td>
<td>10</td>
<td>14</td>
<td>20</td>
</tr>
<tr>
<td>Halves</td>
<td>649</td>
<td>–</td>
<td>1163</td>
<td>1305</td>
<td>1860</td>
<td>86915</td>
<td>13</td>
<td>19</td>
<td>27</td>
</tr>
<tr>
<td>Sierpinski</td>
<td>2916</td>
<td>–</td>
<td>2430</td>
<td>7608</td>
<td>44660</td>
<td>537199</td>
<td>115</td>
<td>161</td>
<td>225</td>
</tr>
<tr>
<td>Teapot</td>
<td>6428</td>
<td>–</td>
<td>8338</td>
<td>22350</td>
<td>42419</td>
<td>1006944</td>
<td>166</td>
<td>231</td>
<td>323</td>
</tr>
<tr>
<td>Wheel</td>
<td>1288</td>
<td>–</td>
<td>1691</td>
<td>1791</td>
<td>2307</td>
<td>103598</td>
<td>131</td>
<td>183</td>
<td>256</td>
</tr>
</tbody>
</table>

that the recognition of a non-manifold edge \(e\) in the NMIA data structure is always performed in \(O(1)\), while in the IA∗ data structure, this operation can be performed in \(O(1)\) if and only if there is at least one top triangle in \(St(e)\). For instance, \(I_{IA^*} > I_{NM}\), for regular simplicial 3-complexes like the “Sierpinski” 3D shape, where \(I_{IA^*} \approx 1.2 \times I_{NM}\).

In the IS data structure, the combinatorial boundary of a simplex is retrieved by visiting the IS boundary graph (see Section 4.1.3.1). Partial adjacency relation \(R_{k,k-1}\) is retrieved by combining relations \(R_{k,k-1}\) and \(R_{k-1,k}\), and this operation is less efficient than in the IA∗ data structure (see Section 7.5.3). Similarly, identification of non-manifold singularities is slower than in the IA∗ data structure (see Section 7.5.5). As a consequence, the IS representation is less efficient than the IA∗ data structure for identifying the MC-Decomposition. Specifically, for simplicial 2-complexes, our tests show that, on average, \(I_{IS} \approx 1.6 \times I_{IA^*}\), and \(I_{IS} \approx 1.4 \times I_{TS}\). Conversely, for simplicial 3-complexes, \(I_{IS} \approx 3.2 \times I_{IA^*}\), and \(I_{IS} \approx 2.1 \times I_{NM}\).

Similarly, the SIG data structure is less efficient than the IA∗ data structure, since involved
primitives are less efficient than in the IA* data structure. For simplicial 2-complexes, our tests show that, on average, $I_{SIG} \approx 2.6 \times I_{IA^*}$, and $I_{SIG} \approx 1.6 \times I_{IS}$. Conversely, for simplicial 3-complexes, $I_{SIG} \approx 6.4 \times I_{IA^*}$, and $I_{SIG} \approx 2 \times I_{IS}$.

Finally, our tests show that the IG data structure does not support efficiently the identification of MC-Decomposition. Specifically, in the IG data structure, identification of non-manifold singularities is extremely slow and inefficient, since we must compute connected components of the link for a simplex each time (see Section 7.5.5). Also, combinatorial boundary and partial adjacency relation of a simplex can be retrieved by exploiting the same algorithms used in the IS data structure, which are slower than in the IA* data structure. For simplicial 2-complexes, our tests show that, on average, $I_{IG} \approx 36 \times I_{IA^*}$, and $I_{IG} \approx 14 \times I_{SIG}$. Conversely, for simplicial 3-complexes, $I_{IG} \approx 192 \times I_{IA^*}$, and $I_{IG} \approx 30 \times I_{SIG}$. Thus, this algorithm, combined with the IG data structure, is really inefficient, and it must not be used in real applications.

Now, we can summarize our results, regarding the execution of Algorithm 8.1:

- $I_{IA^*} < I_{TS} < I_{IS} < I_{SIG} < I_{IG}$, for simplicial 2-complexes;
- $I_{IA^*} < I_{NM} < I_{IS} < I_{SIG} < I_{IG}$, for simplicial 3-complexes.

Now, we can evaluate running times of algorithms which identify arcs in the Exploded, Pairwise, and Compact MC-Graphs, denoted in Table 8.4 as $I_E$, $I_P$, and $I_C$, respectively. They do not depend on which topological data structure we are using, and on the dimension of the simplicial complex we are using. Our tests, summarized in Table 8.4, show that $I_P \approx 1.4 \times I_E$ and $I_C \approx 1.4 \times I_P$. Thus, $I_E < I_P < I_C$, according to their time complexities, which we have proposed in Section 8.3.
Chapter 9

The Manifold-Connected Decomposition and Homology Computation

Recently, the problem of computing topological features of a shape has drawn much attention, because of its applications in several disciplines, including shape analysis and understanding, shape retrieval, and finite element analysis [AIM04, GM05, ZSG09, EH10]. Unlike geometric features, such as curvature, which are only invariant under rigid transformations, topological features are invariant under continuous deformations. Thus, they provide global quantitative and qualitative information about a shape, such as the number of its connected components, and the number of holes and tunnels. Topological features are the core descriptors to extend geometric modelers with non-manifold shapes processing. For instance, the generation of simulation models still lacks capabilities for processing non-manifold shapes, like idealized representations [TCMT56, VL97, VL98, VL01, CDMM04, LF05, TBG09]. Homological information on arbitrary shapes can strongly support new modeling capabilities, because constructive modeling techniques are often used. Also, topological features are very important, especially while analyzing high dimensional data, where pure geometric tools are usually not sufficient.

Simplicial homology is one of the most useful and algorithmically computable topological invariants. It characterizes a simplicial $d$-complex through the notion of homological descriptor. Homological descriptors are defined in any dimension, and are related to non-trivial $k$-cycles which have intuitive geometrical interpretations up to dimension 2. In dimension 0, they are related to the connected components of the complex. Conversely, in dimension 1, they are related to the tunnels and holes, and in dimension 2, to the shells surrounding voids or cavities.

Classical techniques for computing simplicial homology of any arbitrary shape exploit a reduction of incidence matrices to a canonical form, known as the Smith Normal Form (SNF) [Ago76]. Hence, these techniques exploit an algebraic approach, which does not yield to an iterative algo-
rithm [Ser94]. Conversely, the Constructive Homology Theory [Ser99, SR06] offers an elegant way for iteratively computing homology of a simplicial complex from homology of its subcomplexes and of their intersections.

In this chapter, we describe the Mayer-Vietoris (MV) algorithm [BCMA+11], which exploits the Constructive Homology Theory. Our MV algorithm can be defined on any decomposition of a simplicial shape. Specifically, we have combined the MV algorithm with the MC-Decomposition of any arbitrary shape. We have encoded the MC-Decomposition by the Pairwise Manifold-Connected Graph (Pairwise MC-Graph) [CDF11], combined with the IS data structure [DFHPC10], as discussed in Chapter 8. In this case, the MV algorithm relates homology of each MC-component, and of their intersections. Here, the intersection between a pair of MC-components is given by non-manifold singularities, and their number is usually limited. The size of MC-components is smaller than the size of the input shape. We have experimentally shown that these properties of the MC-Decomposition increase efficiency of our MV algorithm, which is more efficient, in terms of storage cost and running times, than the classical SNF algorithm.

Since MC-Decomposition is always decidable and its definition is dimension-independent (see Section 8.2), our MV algorithm is completely dimension-independent. In any case, it is a modular algorithm, since we do not introduce limitation regarding the number of top simplices in MC-components.

In Section 9.1, we review some background notions on simplicial homology. In Section 9.2, we discuss the state of the art on homology computation. In Section 9.3, we introduce basic concepts of the Constructive Homology Theory, and, in Section 9.4, we describe the Homological Smith Reduction, which is the key tool for our MV algorithm. In Section 9.5, we provide a detailed description of the MV algorithm. Finally, in Section 9.6, we present experimental results based on our implementation of the MV algorithm.

### 9.1 Background Notions on Simplicial Homology

In this section, we review some background notions on simplicial homology, which we will use throughout this thesis. Simplicial homology exploits the combinatorial structure of simplicial complexes, and reformulates an homological problem into an algebraic one. Specifically, we concentrate our attention on Euclidean simplicial complexes, which we have discussed in Section 2.2. Simplicial homology can be defined in a more general context: an interested reader can refer to [Ago76, Mun99, Ago05] for more details.

**Simplicial Complexes** Recall that any Euclidean $k$-simplex $\sigma$ is the convex hull of a set $V_\sigma = [v_0, \ldots, v_k]$ of $k+1$ linearly independent points in any Euclidean space $\mathbb{R}^n$, with $k+1 \leq n$. Any $k$-simplex $\sigma$ is denoted as $\sigma = [v_0, \ldots, v_k]$. All the points in $V_\sigma$ are known as vertices of $\sigma$. Given any $V_{\sigma'} \subseteq V_\sigma$ formed by $h+1$ vertices, with $h \leq k$, the Euclidean simplex $\sigma'$ generated by $V_{\sigma'}$.
is a \( h \)-face of \( \sigma \). Any \( k \)-simplex \( \sigma \) can be oriented by assigning a linear ordering to its vertices in \( V_\sigma \). As a consequence, we impose an orientation on \( \sigma \). Two orientations of \( \sigma \) are the same ones if they differ by an even number of transpositions. Opposite orientation of \( \sigma \) is given by reversing the order of all the vertices of \( \sigma \), and it is denoted as \( -\sigma \). Each \((k-1)\)-face \( \sigma_i \) of any \( k \)-simplex \( \sigma \) is defined as \( \sigma_i = [\nu_0, \ldots, \nu_i, \ldots, \nu_k] \), where we discard vertex \( \nu_i \) of \( \sigma \). Thus, the oriented boundary \( d_k(\sigma) \) of any \( k \)-simplex \( \sigma \) is defined as a linear combination of its \((k-1)\)-faces, namely \( d_k(\sigma) = \sum_{i=0}^{k} (-1)^i \sigma_i \). In other words, we can define a boundary operator \( d_k \), which associates any \( k \)-simplex \( \sigma \) with its \((k-1)\)-faces.

Recall that a finite collection \( \Sigma \) of Euclidean simplices forms a Euclidean simplicial complex if all the \( k \)-faces of any Euclidean simplex \( \sigma \) in \( \Sigma \) belong to \( \Sigma \), and the intersection of two simplices \( \sigma' \) and \( \sigma'' \) in \( \Sigma \) is either empty, or a common face in \( \Sigma \). In the remainder of this chapter, we denote a Euclidean simplex and a Euclidean simplicial complex as a simplex and a simplicial complex, respectively.

**Chain-complexes** For each simplicial \( d \)-complex \( \Sigma \), simplicial homology builds an algebraic object \( C_\ast \), known as the chain-complex of \( \Sigma \), on which the homological problem for \( \Sigma \) is resolved by using linear algebra. We assume that all the simplices in \( \Sigma \) are oriented.

In this context, we can define a \( k \)-chain in any simplicial complex \( \Sigma \) as a linear combination \( a_k \) of oriented \( k \)-simplices \( \sigma^k \) in \( \Sigma \), namely \( a^k = \sum \lambda_i \sigma^k_i \), with \( \lambda_i \) in \( \mathbb{Z} \). We can define addition operator between a pair of \( k \)-chains by adding coefficients \( \lambda_i \) simplex by simplex. The set of \( k \)-chains in \( \Sigma \), together with addition operation on \( k \)-chains, forms the chain group \( C_k \). These chain groups are Abelian and finitely generated [Mun99], thus, all the oriented \( k \)-simplices form the canonical basis \( \beta^k_0 \) for \( C_k \). Each boundary operator \( d_k \) can be linearly extended to any \( k \)-chain \( a^k \) as the sum of boundaries for all the \( k \)-simplices in \( a^k \), namely \( d_k(a^k) = \sum \lambda_i d_k(\sigma^k_i) \). The chain-complex of any simplicial \( d \)-complex \( \Sigma \), denoted as \( C_\ast = (C_k,d_k) \), with \( 0 \leq k \leq d \), provides an algebraic description of \( \Sigma \), and is a sequence of chain groups \( C_k \) connected by boundary operator \( d_k \) as follows:

\[
(C_\ast, d_\ast) : 0 \leftarrow C_0 \overset{d_0}{\leftarrow} C_1 \overset{d_1}{\leftarrow} \cdots \overset{d_{d-1}}{\leftarrow} C_{d-1} \overset{d_{d-1}}{\leftarrow} C_d \overset{0}{\leftarrow} 0
\]

Any chain-complex \( C_\ast \) can be encoded as a pair \((\beta^0_0, D_k)\), for \( 0 \leq k \leq d \), where \( D_k \) is an integer matrix, known as the incidence matrix of dimension \( k \). The matrix \( D_k \) expresses the boundary operator \( d_k \) with respect to the basis \( \beta^k_{k-1} = [\sigma^k_0^{-1}, \ldots, \sigma^k_p^{-1}] \) and \( \beta^k_{k} = [\sigma^k_0, \ldots, \sigma^k_k] \). Each element \( \eta^k_{j,i} \) of \( D_k \), with \( 0 \leq j \leq p \) and \( 0 \leq i \leq l \), is defined as follows:

\[
\eta^k_{j,i} = \begin{cases} 
0 & \text{if } \sigma^{k-1}_j \text{ is not in the boundary of } \sigma^k_i; \\
1 & \text{if } \sigma^{k-1}_j \text{ is in the boundary of } \sigma^k_i; \\
-1 & \text{if } -\sigma^{k-1}_j \text{ is in the boundary of } \sigma^k_i.
\end{cases}
\]

We denote the canonical basis \( \beta_\ast \) of the chain-complex \( C_\ast \) as the union of all the basis \( \beta^k_0 \) of the chain group \( C_k \), for \( 0 \leq k \leq d \). Given two chain-complexes \( C_\ast \) and \( C'_\ast \), a chain-complex morphism \( f : C_\ast \rightarrow C'_\ast \) is a collection of linear morphisms \( f_k : C_k \rightarrow C'_k \), for \( 0 \leq k \leq d \). Also, the homology operator \( h : C_\ast \rightarrow C'_\ast \) is a collection of linear maps \( h_k : C_k \rightarrow C'_{k+1} \), for \( 0 \leq k < d \). 216
Homology groups Given any chain-complex $C_*$, we recognize two subgroups of the chain groups $C_k$ defined by the boundary operators $d_k$, namely the group of $k$-cycles $Z_k = \ker d_k = \{c \in C_k \mid d_k(c) = 0\}$, and the group of $k$-boundaries $B_k = \text{img} d_{k+1} = \{c \in C_k \mid \exists a \in C_{k+1} : c = d_{k+1}(a)\}$. Here, we say that a $k$-cycle in $Z_k$ bounds if it is also in $B_k$. Two cycles are homologous if they differ by a cycle which bounds. Clearly, $B_k \subseteq Z_k \subseteq C_k$. For each dimension $0 \leq k \leq d$, we define the homology group $H_k$ as the quotient of $Z_k$ over $B_k$, namely $H_k = Z_k/B_k$. Hence, elements of $H_k$ are equivalence classes related to all the $k$-cycles which are not $k$-boundaries. As demonstrated in [Mun99], a chain group $C_k$ is an Abelian group, then $H_k$ is isomorphic to:

$$\frac{\text{free part}}{\text{torsion part}}\left(\mathbb{Z} \oplus \ldots \oplus \mathbb{Z} \oplus \mathbb{Z}/\lambda_1 \mathbb{Z} \oplus \ldots \oplus \mathbb{Z}/\lambda_p \mathbb{Z}\right)$$

The number of occurrences of $\mathbb{Z}$ in the free part is known as the $k^{\text{th}}$ Betti number $\beta_k$, which corresponds to the maximal number of independent $k$-cycles which do not bound. Values $\lambda_1, \ldots, \lambda_p$ are known as the torsion coefficients, and satisfy two conditions, namely $\lambda_1 \geq 2$, and $\lambda_i$ divides $\lambda_{i+1}$, for $1 \leq i < p$. A set of homologous $k$-cycles can be associated with each group $\mathbb{Z}/\lambda_i \mathbb{Z}$ in $H_k$. These $k$-cycles are not the boundary of any $(k+1)$-chain, but if taken $\lambda_i$ times, they become the boundary of any $(k+1)$-chain. They are known as weak-boundaries. Each homology group $H_k$ is generated by $q$ independent equivalence classes $C_1 \ldots C_q$, then any set $\{g_1, \ldots, g_q\} \subseteq C_1, \ldots, C_q$ is known as the set of generators for $H_k$. In the remainder of this chapter, we denote the complete homological information of any simplicial complex $\Sigma$, including Betti numbers, torsion coefficients, and generators, as the $\mathbb{Z}$-homology of $\Sigma$.

Mayer-Vietoris Sequence The Mayer-Vietoris Sequence is an algebraic tool, which allows studying homology of any space $X$ by splitting it into two subspaces $A$ and $B$ such that $A \cap B \neq \emptyset$. Clearly, homology groups of $A$ and $B$ have to be easier to compute. This sequence relates the chain-complex $(A \cup B)_*$, corresponding to $A \cup B$, to the chain-complexes of the disjoint sum $A_* \oplus B_*$ and the intersection $(A \cap B)_*$. The chain-complex $A_* \oplus B_*$ is called sum of the chain-complexes $A_*$ and $B_*$, and contains chains $(\sigma, \bar{\sigma})$, with $\sigma$ in $A_*$ and $\bar{\sigma}$ in $B_*$. In other words, the chain-complex $A_* \oplus B_*$ is originated from the disjoint union of the subcomplexes $A$ and $B$, namely $A \cup B$. Hence, we can define the chain-complex morphism $j = j_A \circ j_B : (A \oplus B)_* \to (A \cap B)_*$, such that $j(\sigma) = \sigma - \bar{\sigma}$. We can also define the chain-complex morphism $i = i_A \oplus i_B : (A \cap B)_* \to (A \oplus B)_*$, such that $i(\sigma) = (\sigma, \bar{\sigma})$. Note that the chain-complex morphism $i$ is the inclusion of a chain of $A \cap B$ on $A$ and $B$. Hence, the Mayer-Vietoris Sequence can be defined as follows:

$$0 \overset{0}{\leftarrow} (A \cup B)_* \overset{j}{\leftarrow} (A \oplus B)_* \overset{i}{\leftarrow} (A \cap B)_* \overset{0}{\leftarrow} 0$$

Note that this sequence is exact, since $\text{img}(i) = \ker(j)$, therefore, $(A \cap B)_* \cong \ker(j)$ and $(A \cup B)_* \cong (A \oplus B)_*/\text{img}(i)$. As a consequence, this sequence is also known as the Exact Short Sequence of Mayer-Vietoris. As demonstrated in [Mun99], we can build the Long Exact Sequence of Mayer-Vietoris for the homology groups, defined as follows:

$$\ldots \leftarrow H_{k-1}((A \cap B)_*) \overset{\partial}{\leftarrow} H_k((A \cup B)_*) \overset{j}{\leftarrow} H_k((A \oplus B)_*) \overset{i}{\leftarrow} H_k((A \cap B)_*) \overset{\partial}{\leftarrow} H_{k+1}((A \cap B)_*) \leftarrow \ldots$$
where the chain-complex morphism $\partial$ is known as the *connection morphism*, built through the diagram chasing, as suggested in [SR06].

In some cases, homology of $(A \oplus B)_n$ can be deduced from this sequence, but it is not always possible to decide. This problem is known as the *extension problem*. Moreover, there is no way to give the generators of the homology group, because this method is non-constructive [SR06]. Thus, the classical Mayer-Vietoris sequence is known as a purely theoretical tool, and is useful only for computations by hand. In Section 9.3, we provide the constructive version of the Mayer-Vietoris sequence (see Definition 9.3.4).

**Smith Normal Form (SNF) algorithm** Let $D_k$ be the incidence matrix of dimension $k$, which relates basis $\beta^k = [\sigma^k_0, \ldots, \sigma^k_l]$ and $\beta^{k-1} = [\sigma^{k-1}_0, \ldots, \sigma^{k-1}_p]$ of the chain groups $C_k$ and $C_{k-1}$, respectively. As demonstrated in [Mun99], the incidence matrix $D_k$ can be decomposed as $D_k = P_{k-1}N_kP_k$, where matrix $N_k$ is the *Smith Normal Form (SNF)* of $D_k$, while matrices $P_{k-1}$ and $P_k$ describe the basis change operated by this decomposition. The matrix $N_k$ is:

$$
N_k = \begin{pmatrix}
\sigma^k_0 & \ldots & \sigma^k_l \\
0 & \lambda & 0 \\
0 & 0 & \text{Id} \\
0 & 0 & 0
\end{pmatrix}
$$

where $\lambda$ is a diagonal matrix formed by values $\lambda_i$ in $\mathbb{Z}$, such that $\lambda_i > 1$, and $\lambda_i$ divides $\lambda_{i+1}$.

Matrix $N_k$ is expressed into different basis $\beta^k_{s-1}$ and $\beta^k_s$, which form the *Smith basis* $\beta_s$. Matrices $P_k$ and $P_{k-1}$ encode the basis changes $P_k : C_k[\beta_s] \rightarrow C_k[\beta_s]$ and $P_{k-1} : C_{k-1}[\beta_s] \rightarrow C_{k-1}[\beta_s]$, respectively. Initially, $P_k = P_{k-1} = \text{Id}$, but each operation on the rows and the columns of $D_k$ is translated into an operation on $P_k$ and $P_{k-1}$. Hence, matrices $P_k$ and $P_{k-1}$ tell us how to express an element of the Smith basis $\beta_s$ in terms of the canonical basis of $C_k$ and $C_{k-1}$.

The $\mathbb{Z}$-homology of any simplicial $d$-complex $\Sigma$ is computed by using two consecutive incidence matrices in Smith Normal forms, namely $N_k$ and $N_{k+1}$, with $0 \leq k < d$. Rank of the subgroup $Z_k = \ker N_k$ is equal to the number of zero-columns of $N_k$, which correspond to the $k$-cycles. Rank of $B_k = \text{img } N_{k+1}$ is equal to the number of non-zero rows of $N_{k+1}$. Generators, expressed in the canonical basis $\beta_c$, are obtained by computing the image of each generator $\gamma_i$ from the Smith basis $\beta_s$ by $P_k$.

### 9.2 State of the Art

In this section, we discuss several techniques in the literature, related to the homology computation.

The classical approach to compute the $\mathbb{Z}$-homology of any simplicial complex is based on the *Smith Normal Form (SNF)* [Ago76, Mun99]. Although this method is theoretically valid in any dimension and for any kind of simplicial complex, it has some inherent limitations regarding the size of incidence matrices and high complexity of the reduction algorithm. The best available
reduction algorithms have super-cubical complexity [Sto96, DHSV03], thus they are suitable only for small simplicial complexes. Another well-known problem is the appearance of huge integers during the reduction process [HM91].

In the literature, several optimizations of the SNF algorithm have been developed. Stochastic methods [Gie96] are efficient on sparse integer matrices, but they do not provide generators. Deterministic methods [KB79, Sto96] perform computations modulo an integer value chosen by a determined criterion, but information about torsion coefficients is lost with this strategy. Another way to improve computation times is to reduce the input complex without changing its topology by applying iterative simplifications, and by computing homology when no more simplifications are possible. This reduction approach has been mainly investigated in the context of homology computation from 3D voxel images [KMS98, DPF06, MPZ08, PIK+09]. Other reduction approaches apply the Discrete Morse Theory to homology computations [For98], since one expects the Morse complex built on the original simplicial complex is much smaller than this latter.

Another approach for homology computation is based on the persistent homology [EH10]. In this framework, the input simplicial complex is filtered, according to any real function, in order to study which homological attributes appear, disappear, and are maintained by nesting. Pertinent information is encapsulated by a pairing of critical values in the function, which are visualized by points forming a diagram in the plane. Since filtration is done by adding only one simplex at a time, it can be considered as a special case of the Mayer-Vietoris sequence.

These methods are usually designed for simplicial complexes with dimensional restrictions in most of cases. In [ELZ02] the authors define an algorithm which computes pairs from an ordering of simplices in any triangulation, and exhibits a cubic worst-case time in the size of the input complex. In [CSEM06] the authors propose an algorithm which maintains the pairing in worst-case linear time per transposition in the ordering. A nearly linear algorithm for computing only the Betti numbers for simplicial 3-complexes is proposed in [DE93].

In [DG96] the authors propose an algorithm for computing homological generators of any manifold simplicial complex embedded in the Euclidean space $E^3$. This technique can be extended to arbitrary simplicial complexes by a thickening process. The algorithm presented in [GKM+07] computes non-contractible 1-cycles of manifold surfaces. The technique described in [DLSC08] retrieves two types of 1-cycles, which identify handles and tunnels in manifold surfaces. In [DSW10] the authors propose an algorithm for computing loops from point data which sample a manifold surface. This technique approximates the shortest basis of the 1-dimensional homology group in a manifold shape, and minimizes the length of its generators. Geometric properties of generators have been also addressed in [ZC08, CF10]. However, the persistence of a feature depends on the chosen filtering function. At the moment, it is still an open problem to find geometrically meaningful functions for non-manifold simplicial complexes.

In [DFLM09] the author propose a first definition of the Mayer-Vietoris formula for persistent homology in the context of shape recognition in presence of occlusions. However, this work is based on the classical version of the Mayer-Vietoris sequence, and the proposed formula cannot
be used in practice, since it does not lead to an algorithm.

Finally, there exist also a few methods based on the Constructive Homology Theory [Ser94, Ser99, SR06], which provides an original algorithmic approach for computing homology. Concepts borrowed from the Constructive Homology Theory have been used in [ADF+09, GDJMR09] for computing homology of images. To the best of our experience, none of the existing algorithms exploits the Constructive Exact Short Sequence of Mayer-Vietoris, which offers an elegant way for computing homology of a simplicial complex from homology of its subcomplexes and of their intersections.

9.3 Constructive Homology Theory

The Mayer-Vietoris algorithm we present in this chapter is an application of the Constructive Homology Theory [Ser94, Ser99, SR06]. This theory has been developed in order to solve the non-constructiveness of classical homology from its roots. Within this framework, based on the Constructive Mathematics [TvD88], homological concepts are reformulated into concepts with a computational nature, thus yielding to effective implementable algorithms. Constructive Homology Theory has been developed to handle homology computations over chain-groups of infinite dimension, and its validity has been proven by using functional programming [DRSS08]. This section is devoted to review basic notions of the Constructive Homology Theory, presented in [SR06].

Specifically, the Constructive Homology Theory offers an elegant way for computing homology of a simplicial complex from homology of its subcomplexes and of their intersections. Here, we concentrate our attention on several key concepts, namely the reduction, and the cone of a morphism. We also present three constructive theorems, namely the Short Exact Sequence (SES), the Cone Reduction, and the Cone Equivalence theorems.

**Definition 9.3.1 (Reduction).** Given two chain-complexes $\tilde{C}_*$ and $C_*$, a reduction $\rho : \tilde{C}_* \Rightarrow C_*$ can be defined by the following diagram:

$$
\rho = \tilde{C}_* \xrightarrow{h} C_* \\
g \downarrow \quad \quad \quad \downarrow \quad \quad \quad \downarrow \quad \quad \quad \downarrow
\tilde{C}_* \quad f \\
C_*
$$

where $f$ and $g$ are chain-complex morphisms such that $f \circ g = \text{id}_{\tilde{C}_*}$. Here, $h : \tilde{C}_* \rightarrow \tilde{C}_*$ is an homotopy operator such that $f \circ h = h \circ g = h \circ h = 0$ and $g \circ f + d \circ h + h \circ d = \text{id}_{\tilde{C}_*}$, where $d$ is the boundary operator, defined as the sequence of boundary operators $d_k$.

Any reduction $\rho : \tilde{C}_* \Rightarrow \tilde{C}_*$ such that chain-complex morphisms $f$ and $g$ are the identity morphisms, and the homotopy operators $h$ are 0-morphisms, is known as the trivial reduction.

Any reduction relates two chain-complexes with equivalent homologies in such a way that, if homology of a chain-complex is known, then homology of the other chain-complex can be found by the reduction. Intuitively, it relates a large chain-complex $\tilde{C}_*$ to a small chain-complex $C_*$. 
which contains the same homological information. In other words, any reduction $\rho$ provides a compact and convenient form for the diagram depicted in Figure 9.1, which implicitly provides a decomposition of any chain group $\hat{C}_k$.

![Figure 9.1: Any reduction $\rho$ implicitly provides a decomposition of any chain group $\hat{C}_k$. Figure courtesy of [SR06].](image)

Specifically, each chain group $\hat{C}_k$ is decomposed as the direct sum of three components, namely $\hat{C}_k = A_k \oplus B_k \oplus C'_k$. Note that there exists a bijection between $A_{k+1}$ and $B_k$ through the boundary operator $d$ and the homotopy operator $h$ for each $k$. Therefore, each component $A_{k+1}$ is a collection of $(k+1)$-cycles such that their boundaries belong to $B_{k-1}$. These cycles are known as the pre-boundaries. Each component $B_k$ is a collection of $k$-cycles, which are known as the $k$-boundaries. Each component $C'_k$ is a copy of $C_k$ and, thus, $C'_* \cong \hat{C}_*$. As a consequence, a large chain-complex $\hat{C}_*$ is the direct sum of one small chain-complex $C'_*$ and $A_* \oplus B_*$, where the last component does not play a key role from the homological point of view.

In Section 9.4, we introduce the Homological Smith Reduction, which is the key tool for our MV algorithm. This reduction is based on the Smith Normal Form of incidence matrices [Mun99].

Now, we introduce the concept of reduction equivalence between two chain-complexes.

**Definition 9.3.2 (Reduction equivalence).** A reduction equivalence $\epsilon : C_* \leftrightarrow D_*$ between two chain-complexes $C_*$ and $D_*$, is a pair of reductions $\rho_L : \hat{C}_* \Rightarrow C_*$ and $\rho_R : \hat{C}_* \Rightarrow D_*$, which connect $C_*$ and $D_*$ through a third chain-complex $\hat{C}_*$, as shown in the following diagram:

![Diagram showing reduction equivalence](image)
In this context, chain groups where matrices $D$ boundary operator is given by the matrix:

$$D = \begin{bmatrix} D_Y & f_{k-1} \\ 0 & -D_{X_{k-1}} \end{bmatrix}.$$ 

In other words, a reduction equivalence provides a composition of two reductions. Reductions $\rho_L$ and $\rho_R$ relate chain-complexes $C_\ast$ and $D_\ast$, which contain the same homology information, and a big object $\hat{C}_\ast$. This operation is convenient when $D_\ast$ is smaller than $C_\ast$, while the big object $\hat{C}_\ast$ is required only to link $C_\ast$ and $D_\ast$. Note that this equivalence implies that homology groups of $D_\ast$ and $C_\ast$ are isomorphic.

Now, we can introduce another key concept in the Constructive Homology Theory, namely the cone of a morphism. Informally, it represents a chain-complex morphism $f$ relating two chain-complexes $X_\ast$ and $Y_\ast$ as a new chain-complex. Informally, this representation makes possible to build an object homologically equivalent to $X_\ast$ and $Y_\ast$.

**Definition 9.3.3 (Cone of a morphism).** Let $f : X_\ast \rightarrow Y_\ast$ be a chain-complex morphism between two chain-complexes $X_\ast$ and $Y_\ast$. The cone of the chain-complex morphism $f$ is a chain-complex, denoted as $Cone(f)_\ast$. For each dimension $k$, $Cone(f)_k = Y_k \oplus X_{k-1}$, and the related boundary operator is given by the matrix:

$$D_{Cone(f)_k} = \begin{bmatrix} D_{Y_k} & f_{k-1} \\ 0 & -D_{X_{k-1}} \end{bmatrix},$$

where matrices $D_{Y_k}$ and $D_{X_{k-1}}$ are the incidence matrices of chain-complexes $Y_\ast$ and $X_\ast$.

In this context, chain groups $Y_k$ and $Y_k$ and $X_{k-1}$ are considered as disjoint. For each dimension $k$, a basis of $Cone(f)_k$ is formed by a basis of $Y_k$ and a basis of $X_{k-1}$.

At this point, we can introduce the Constructive Exact Short Sequence of Mayer-Vietoris between the chain-complexes of two simplicial complexes $A$ and $B$ with a non-empty intersection $A \cap B$. This is the constructive version of the Mayer-Vietoris sequence, and allows defining homology of $A \cup B$ in terms of homologies of $A$, $B$, and $A \cap B$. This sequence is defined as follows.

**Definition 9.3.4 (Constructive Exact Short Sequence of Mayer-Vietoris).** Let $A$, $B$ be two simplicial complexes with a non-empty intersection $A \cap B$, then the following diagram defines the Constructive Exact Short Sequence of Mayer-Vietoris for their chain-complexes:

$$0 \xleftarrow{i} (A \cup B)_\ast \xrightarrow{\nu} A_\ast \oplus B_\ast \xrightarrow{\rho} (A \cap B)_\ast \xrightarrow{0} 0,$$

where $i = i_A \oplus i_B$ and $j = j_A \oplus j_B$ are chain-complex morphisms defined in the same way as in the Exact Short Sequence of Mayer-Vietoris, discussed in Section 9.1. Conversely, $\rho : A_\ast \oplus B_\ast \rightarrow (A \cap B)_\ast$ and $\nu : (A \cup B)_\ast \rightarrow A_\ast \oplus B_\ast$ are graded module morphisms [CE56] such that $\rho \circ i = id_{(A \cap B)_\ast}$, $i \circ \rho + \nu \circ j = id_{A_\ast \oplus B_\ast}$, and $j \circ \nu = id_{(A \cup B)_\ast}$.
In order to complete homological computations, we need several chain-complex morphisms, provided by the following theorems.

Specifically, the Short Exact Sequence (SES) Theorem is probably the most important result on which our algorithm is based. It allows establishing an homological equivalence between the cone of the morphism inclusion \( i : (A \cap B)_* \rightarrow A_* \oplus B_* \) and the chain-complex of \((A \cup B)_*\).

**Theorem 9.3.5 (Short Exact Sequence (SES) Theorem).** The Constructive Exact Short Sequence of Mayer-Vietor, provides the reduction \( \rho : \text{Cone}(i) \Rightarrow (A \cup B)_* \), defined as follows:

\[
\begin{array}{ccc}
\text{Cone}(i)_* & \overset{\rho}{\longrightarrow} & (A \cup B)_* \\
\nu \circ d_{A \oplus B}^\ast \downarrow & \downarrow j & \downarrow \nu \circ d_{A \oplus B}^\ast \\
(A \cap B)_* & \rightarrow & \text{Cone}(i)_*
\end{array}
\]

The chain morphisms are \( f = j \) and \( g = \nu \circ d_{A \oplus B}^\ast \circ \nu \), while the homotopy operator is \( h = \rho \).

Thus, if the homology of the cone of \( i \) is known, then we can retrieve homology of \((A \cup B)_*\) by computing the image of each element of the homology of \( \text{Cone}(i)_* \) with the chain-complex morphism \( f = j_A \odot j_B \). However, \( \text{Cone}(i)_* \) is much larger than \((A \cup B)_*\), thus it would be extremely inefficient to compute homology on this huge object. In order to simplify computations, we need a small intersection between two simplicial complexes \( A \) and \( B \).

In any case, the Cone Reduction Theorem gives us another reduction of chain-complex \( \text{Cone}(i)_* \), and allows building a reduction equivalence between the chain-complex \((A \cup B)_*\) and one very small chain-complex, which is homologically equivalent to \( \text{Cone}(i)_* \).

**Theorem 9.3.6 (Cone Reduction Theorem).** Let \( i : (A \cap B)_* \rightarrow (A \oplus B)_* \) be a chain-complex morphism and two reductions \((A \oplus B)_* \Rightarrow E_A \oplus EB_* \) and \((A \cap B)_* \Rightarrow E(A \cap B)_* \). Then, we can define a reduction \( \rho = (f_c, g_c, h_c) : \text{Cone}(i)_* \Rightarrow \text{Cone}(Ei)_* \), defined by the following diagram:

\[
\begin{array}{ccc}
(A \oplus B)_* & \overset{h_{A \oplus B}}{\longrightarrow} & (A \cap B)_* \\
f_{A \oplus B} & \downarrow & \downarrow h_{A \cap B} \\
E_A \oplus EB_* & \overset{f_{A \cap B}}{\longrightarrow} & E(A \cap B)_* \\
g_{A \oplus B} & \downarrow & \downarrow g_{A \cap B} \\
& \downarrow f_c & \downarrow g_c \\
& \text{Cone}(i)_* & \Rightarrow \text{Cone}(Ei)_*
\end{array}
\]

where chain-complex morphisms \( f_c \) and \( g_c \) of reduction \( \rho \) are:

\[
f_c = \begin{bmatrix}
    f_{A \oplus B} & -f_{A \oplus B} & i \\
    0 & f_{A \cap B}
\end{bmatrix},
\]

\[
g_c = \begin{bmatrix}
    g_{A \oplus B} & -g_{A \oplus B} & (i \circ (h_{A \cap B})) \\
    0 & g_{A \cap B}
\end{bmatrix},
\]

while homotopy operator \( h_c \) of reduction \( \rho \) is:

\[
h_c = \begin{bmatrix}
    h_{A \oplus B} & -h_{A \oplus B} & (i \circ (h_{A \cap B})) \\
    0 & -h_{A \cap B}
\end{bmatrix}
\]

223
Note that reduction \((A \oplus B)_* \Rightarrow E A_* \oplus E B_*\) is defined as the formal sum of the reductions of \(A_*\) and \(B_*\). Moreover, \(\text{Cone}((f_{A \oplus B}) \circ (g_{A \cap B}))_* = E A_* \oplus E B_* \oplus E(A \cap B)_*\), where chain-complexes \(E A_*\), \(E B_*\), and \(E(A \cap B)_*\) are the reduced chain-complexes of, respectively, \(A_*\), \(B_*\) and \((A \cap B)_*\), and contain only their homological information. Therefore, we can efficiently compute homology on this small chain-complex by using the SNF algorithm. As a consequence, we obtain the following reduction equivalence:

\[
\begin{array}{c}
\text{Cone}(i)_* \xrightarrow{(j \circ g_{A \cap B})_*} (A \cup B)_* \xrightarrow{j_*} \text{Cone}(Ei)_* \\
\end{array}
\]

This reduction equivalence demonstrates that \((A \cup B)_*\) has the same homology as \(\text{Cone}(Ei)_*\). Therefore, Betti numbers and torsion coefficients of \((A \cup B)_*\) are provided directly from homology of \(t \text{Cone}(Ei)_*\). Generators of \(H_k((A \cup B)_*)\) can be obtained by computing the image of each cycle \(c\) in \(H_k(\text{Cone}(Ei)_*)\) by \((j \circ g_{A \cap B})_*(c)\).

Finally, we introduce the \textit{Cone Equivalence Theorem}, which is the key result for designing the iterative version of our MV algorithm, which we introduce in Section 9.5.2.

\textbf{Theorem 9.3.7 (Cone Equivalence Theorem).} Let \(i: (A \cap B)_* \rightarrow (A \oplus B)_*\) be a chain-complex morphism, which belongs to the reduction equivalences \((A \cap B)_* \xleftarrow{\text{E}} (A \cap B)_* \xrightarrow{\text{E}} E(A \cap B)_*\) and \((A \oplus B)_* \xleftarrow{\text{E}} (A \oplus B)_* \xrightarrow{\text{E}} E(A \oplus B)_*\), as shown in the following diagram:

\[
\begin{array}{c}
(A \cap B)_* \xrightarrow{i_*} E(A \cap B)_* \xrightarrow{h_*} \text{Cone}(i)_* \xrightarrow{j_*} (A \cup B)_* \xrightarrow{\rho} \text{Cone}(i)_* \xrightarrow{g_*} (A \oplus B)_* \xrightarrow{\rho'} \text{Cone}(Ei)_* \\
\end{array}
\]

Then, we can define the reduction equivalence \(\text{Cone}(i)_* \xleftarrow{\tilde{i}} \text{Cone}(Ei)_* \xrightarrow{\text{E}} \text{Cone}(Ei)_*\), where \(\tilde{i} = (l_{g'} \circ i \circ (lf)) \circ (r_{f'} \circ (lg') \circ i \circ lf) \circ (rg)\).

This theorem retrieves a reduction equivalence between \((A \cap B)_*\) and \(E \text{Cone}(Ei)_*\), from which we can extract the required homological information. Betti numbers and torsion coefficients can be directly accessed in \(E \text{Cone}(Ei)_*\), while generators can be obtained by computing the image of each homological generator in \(E \text{Cone}(Ei)_*\) by chain-complex morphisms of the reduction equivalence.

\section{9.4 Homological Smith Reduction}

In this section, we introduce a specific reduction, which we call the \textit{Homological Smith Reduction}. It will be used to compute homology of each subcomplex of the input simplicial complex in our
Mayer-Vietoris algorithm, which we introduce in Section 9.5. Most of topics discussed in this section are detailed in [BMALH10].

This reduction relates a chain-complex, $X_*$ and a very small chain-complex, $EX_*$, which contains only the homological information of $X_*$. This information is computed by the SNF algorithm [Ago76, Mun99], discussed in Section 9.1, which transforms each incidence matrix $D_k$ into its Smith Normal Form $N_k$. In order to describe $EX_*$, we need a basis for each dimension and a boundary matrix. A basis is defined as a subset of the Smith basis $\beta_s$ of $X_*$, while a boundary matrix is a submatrix of $N_k$. Chain-complex morphisms $f$ and $g$ and the homotopy operator $h$ are defined by matrices $P_k$ and $P_{k-1}$ which describe the basis changes. Thus, we need to classify elements of the Smith basis provided by the SNF algorithm in order to find a basis of $EX_*$. Then, we can construct the reduced chain-complex $EX_*$ from the chain-complex $X_*$.  

**Basis classification** Let $N_k$ and $N_{k+1}$ be two consecutive incidence matrices in Smith Normal Form. We need to classify elements of the Smith basis $\beta^*_k$ in which the columns of $N_k$ and rows of $N_{k+1}$ are expressed in order to find a basis of the small chain-complex $EX_*$. We clarify the basis classification algorithm through the example in Figure 9.2, where $\beta^*_k=\{\gamma_1, \ldots, \gamma_{10}\}$.

Now, we consider the basis of $k$-cycles in $\ker d_k = [\gamma_1, \ldots, \gamma_7]$, which correspond to the zero columns of $N_k$. This basis is the union of three basis, namely $w^k$, $b^k$, and $c^k$. Basis $w^k = \{\gamma_1, \gamma_2\}$ corresponds to the rows of $N_{k+1}$ with coefficient $\lambda_i > 1$, and is formed by elements related to weak-boundaries. Basis $b^k = \{\gamma_3, \gamma_4, \gamma_5\}$ corresponds to the rows of $N_{k+1}$ with coefficient $\lambda_i = 1$, and is formed by the elements related to boundaries. Finally, the remaining kernel basis $c^k = \{\gamma_6, \gamma_7\}$ corresponds to non-trivial $k$-cycles.

We complete the basis classification with the $k$-chains which are not $k$-cycles. All the elements $pb^k = \{\gamma_9, \gamma_{10}\}$ corresponding to the columns of $N_k$ with coefficients $\lambda_i = 1$ are known as the pre-boundaries. Note that these chains do not carry homological information. Finally, all the elements $pw^k = \{\gamma_8\}$ corresponding to the columns of $N_k$ with coefficients $\lambda_i > 1$ are known as the pre-weak boundaries, and are related to torsion coefficients.

Given any simplicial $d$-complex $\Sigma$, the basis classification for vertices and for $d$-simplices must be treated as special cases, since these boundary morphisms are zero morphisms. Hence, in the basis of dimension $d$, there are only cycles, pre-boundaries, and possibly pre-weak boundaries, but not weak-boundaries nor boundaries. In the vertices basis, there are only cycles and boundaries.

**Reduced chain-complexes** Basis classification allows constructing the reduced chain-complex $EX_*$ from $X_*$. Note that the basis classification is equivalent to the decomposition of $X_k$ into three groups, namely $X_k = A_k \oplus B_k \oplus C'_k$, as shown in Figure 9.2.

Component $A_k = [pb^k]$ is generated by $k$-chains which do not play a key role in homology computations. Chain-group $B_k = [b^k]$ is generated by $k$-cycles which are known to be boundaries. Note that the group generated by pre-boundaries $[pb^k]$ is isomorphic to the group generated by boundaries
Figure 9.2: Classification of the Smith basis $\beta_k^s$ by using two consecutive incidence matrices $N_k$ and $N_{k+1}$ in Smith Normal Form. First, we consider $k$-cycles in $\ker d_k$. These cycles can be classified in weak-boundaries $w^k$, boundaries $b^k$, and non-trivial cycles $c^k$. Remaining $k$-chains can be classified in pre-boundaries $pb^k$ and pre-weak boundaries $pw^k$.

$[b^{k-1}]$. Hence, homology of $X_k$ is given by the reduced chain-complex $EX_k = C'_k = [w^k, c^k, pw^k]$. For each dimension $1 \leq k \leq d$, the boundary matrix $EN_k$ of $EX_k$ is:

$$EN_k = \begin{pmatrix} w^{k-1} & c^{k-1} & pw^{k-1} \\ 0 & 0 & \lambda \\ 0 & 0 & 0 \end{pmatrix}.$$

It is immediate to prove that $EN_{k-1}EN_k = 0$ for $1 \leq k \leq d$, hence $EN_{k_*}$ is a chain-complex. Hence, we can define the Homological Smith Reduction as follows.

**Definition 9.4.1 (Homological Smith Reduction).** Let $X_*$ be a chain-complex $X_*$, then its Homological Smith Reduction $\rho : X_* \Rightarrow EX_*$ is defined by the following diagram:

$$\rho = \begin{array}{ccc} X_* & \xrightarrow{h} & X_* \\ \downarrow g & & \downarrow f \\ EX_* = [w^*, c^*, pw^*] & & \end{array}$$

where, for each dimension $k$, morphisms $f_k$ and $g_k$ in the chain-complex morphisms $f : X_* \to EX_*$ and $g : EC_* \to X_*$ are, respectively, $f_k = P_k^{k-1}|[w^k, c^k, pw^k]$ and $g_k = P_k|[w^k, c^k, pw^k]$. Each linear map $h_k$ in the homotopy operator $h : X_* \to X_{k+1}$ is $h_k = P_k|\rho h_k = P_k^{-1}|b^{k-1}$. Each linear map $h_k$ is defined by combining restrictions of the matrices $P_k$ and $P_k^{-1}$ of the Smith basis to the basis related to pre-boundaries $pb^k$ and boundaries $b^{k-1}$. Chain-complex morphisms $f$ and $g$ are inverse isomorphisms between $EX_*$ and a subchain of $X_*$, which contains homological information of $X_*$. The restriction of each linear map $h_k : B_k \to A_{k+1}$ and the restriction of the boundary operator $d_k : A_{k+1} \to B_k$ are isomorphisms among boundaries and pre-boundaries. This means that, given...
any boundary $\sigma^k$ in $B_k$, the linear map $h_k$ gives us the $(k+1)$-chain of $A_{k+1}$ for which $\sigma^k$ is the boundary. Intuitively, homotopy operator $h$ captures only the information about boundaries and their pre-boundaries. It can be seen as the constructive version of boundary.

Algorithm 9.1 summarizes operations needed for computing the Homological Smith Reduction of a chain-complex $X_*$.  

**Algorithm 9.1 Building the Homological Smith Reduction**  
**Input:** A chain-complex $X_*$.  
**Output:** The Homological Smith Reduction $X_* \Rightarrow E X_*$.  

1: for all $1 \leq k \leq \dim(X)$ do  
2: Compute the SNF of the incidence matrix $D_k = P_{k-1} N_k P_k^{-1}$  
3: Classify the Smith basis $[w^k, b^k, c^k, pw^k, pb^k]$  
4: Compute the chain-complex $EX_k$ as the restriction $EX_k = N_k |_{[w^k, c^k, pw^k]}$  
5: Compute the chain-complex morphisms $f$ and $g$ by the restrictions:  
6: $f_k = P_{k-1}^{-1} |_{[w^k, c^k, pw^k]}$  
7: $g_k = P_k |_{[w^k, c^k, pw^k]}$  
8: Compute the linear maps $h_k$ in the homotopy operator $h$ as $h_k = P_k |_{[pb^k]} * P_{k-1}^{-1} |_{[b^k-1]}$  
9: end for

### 9.5 The Mayer-Vietoris Algorithm

In this section, we introduce our Mayer-Vietoris (MV) algorithm, which is based on the Constructive Homology Theory, discussed in Section 9.3, and on the Homological Smith Reduction, defined in Section 9.4. The key idea of our approach consists of combining the Homological Smith Reduction with tools provided by the Constructive Homology Theory.

Specifically, in Section 9.5.1, we explain how the Constructive Exact Short Sequence of Mayer-Vietoris yields to an effective algorithm for homology computation. Here, given two simplicial complexes $A$ and $B$ with a non-empty intersection, we design an algorithm which computes homology of the simplicial complex $A \cup B$ in terms of homologies of $A$, $B$, and $A \cap B$. In Section 9.5.2, we introduce the iterative version of our dimension-independent Mayer-Vietoris (MV) algorithm for computing the homology of any simplicial complex $\Sigma$, starting from its Pairwise MC-Graph $G^p_{\Sigma}$, introduced in Section 8.3.2.

#### 9.5.1 Computing the Homology of the Union of Two Simplicial Complexes

In this section, we explain how the Constructive Exact Short Sequence of Mayer-Vietoris yields to an effective algorithm for homology computations. Specifically, given two simplicial complexes $A$ and $B$ with a non-empty intersection, we design an algorithm which computes homology of the
simplicial complex \( A \cup B \) in terms of the homologies of \( A_*, B_* \), and \( (A \cap B)_* \). Most of topics discussed in this section are detailed in [BMALH10].

This algorithm can be defined on any pair of simplicial complexes \( A \) and \( B \) with a not empty intersection. Here, we illustrate this algorithm with a running example in Figure 9.3(a), where a simplicial 2-complex \( \Sigma \) is decomposed in two MC-components \( A \) and \( B \) with a non-empty intersection \( A \cap B \) formed by two isolated vertices.

The first step of our algorithm consists of computing the Homological Smith Reductions of chain-complexes \( A_* \), \( B_* \) and \( (A \cap B)_* \), as explained in Section 9.4. We obtain reductions \( A_* \Rightarrow EA_* \), \( B_* \Rightarrow EB_* \) and \( (A \cap B)_* \Rightarrow E(A \cap B)_* \), as shown in Figure 9.3(b).

In the second step of our algorithm, we compute the Constructive Exact Short Sequence of Mayer-Vietoris, as shown in Figure 9.4. Specifically, we compute chain-complex morphisms \( i, j, \rho \) and \( \nu \), following Definition 9.3.4.

At this point, we can apply the SES theorem, namely the Theorem 9.3.5, which builds a reduction \( \text{Cone}(i)_* \Rightarrow (A \cup B)_* \) between the cone of inclusion morphism \( i \) and chain-complex \( (A \cup B)_* \), as illustrated in Figure 9.5(a). This means that the chain-complex \( \text{Cone}(i)_* \) has the same homology as the chain-complex \( (A \cup B)_* \).

In the fourth step of our algorithm, we apply the Cone Reduction Theorem, namely the Theorem 9.3.6, in order to build a new reduction \( \text{Cone}(i)_* \Rightarrow \text{Cone}(Ei)_* \). This reduction establishes a homological equivalence between the large chain-complex \( \text{Cone}(i)_* \) and the chain-complex \( \text{Cone}(Ei)_* \), which relates the reduced chain-complexes \( EA_* \oplus EB_* \) and \( E(A \cap B)_* \), as shown in Figure 9.5(b). Note that chain-complex \( \text{Cone}(Ei)_* \) can be efficiently computed from the reduced...
chains \( EA \oplus EB \) and \( E(A \cap B)_* \), following Definition 9.3.3.

Figure 9.5: (a) The SES and (b) the Cone Reduction Theorems for the example in Figure 9.3(a).

At this point, we establish the reduction equivalence \((A \cup B)_* \Leftrightarrow Cone(Ei)_*\) from the last two reductions. This means that the chain-complex \( Cone(Ei)_* \) has the same homology as the chain-complex \((A \cup B)_*\), since they are related by \( Cone(i)_* \), as shown in Figure 9.6(a). However, \( Cone(Ei)_* \) is much smaller than \((A \cup B)_*\), since it contains only the homological information of \( A, B \) and \( A \cap B \).

In the next step, we compute homology of the small chain-complex \( Cone(Ei)_* \) by the Homological Smith Reduction \( Cone(Ei)_* \xrightarrow{} ECone(Ei)_* \), as shown in Figure 9.6(b).

Figure 9.6: Last steps of the MV algorithm for the example in Figure 9.3(a). We compute (a) the reduction equivalence \((A \cup B)_* \Leftrightarrow Cone(Ei)_*\), (b) the Homological Smith Reduction \( Cone(Ei)_* \xrightarrow{} ECone(Ei)_* \), and (c) the reduction equivalence \((A \cup B)_* \Leftrightarrow ECone(Ei)_*\).

Finally, the MV algorithm composes the last two reductions, and produces as output a reduction equivalence between chain-complexes \((A \cup B)_* \) and \( ECone(Ei)_* \), as shown in Figure 9.6(c), from which we can extract the required homological information. Betti numbers torsion coefficients of \((A \cup B)_* \) can be directly accessed in \( ECone(Ei)_* \), while generators can be obtained by computing the image of each homological generator in \( ECone(Ei)_* \) by the chain-complex morphisms of the reduction equivalence. Main steps of our algorithm are summarized in Algorithm 9.2.
Algorithm 9.2 Computing the Homology of the Union of Two Simplicial Complexes

**Input:** three simplicial complexes $A$, $B$, and $A \cap B \neq \emptyset$

**Output:** the reduction equivalence $(A \cup B) \overset{\prec}{\leftrightarrow} \text{Cone}(\text{E}(i)), \text{Cone}(\text{E}(\text{C}(i))) \overset{\prec}{\leftrightarrow} \text{E}(\text{Cone}(\text{E}(i)))$

1: Compute the chain-complexes $A_\ast$, $B_\ast$, and $(A \cap B)_\ast$, and the Homological Smith Reductions $A_\ast \Rightarrow \text{E}A_\ast$, $B_\ast \Rightarrow \text{E}B_\ast$, and $(A \cap B)_\ast \Rightarrow \text{E}(A \cap B)_\ast$

2: Compute the chain-complexes $(A \oplus B)_\ast$ and $(A \cup B)_\ast$, plus the morphisms $i, j, \rho$ and $\nu$ in the Constructive Exact Short Sequence of Mayer-Vietoris

3: Construct the reduction $\text{Cone}(i)_\ast \Rightarrow (A \cup B)_\ast$ of the morphism inclusion $i$, as provided by the SES Theorem

4: Construct the reduction $\text{Cone}(i)_\ast \Rightarrow \text{Cone}(\text{E}(i))_\ast$ of the morphism inclusion $i$, as provided by the Cone Reduction Theorem

5: Construct the Homological Smith Reduction $\text{Cone}(\text{E}(i))_\ast \Rightarrow \text{E}(\text{Cone}(\text{E}(i)))_\ast$

6: Construct the reduction equivalence $\text{Cone}(i)_\ast \Rightarrow \text{E}(\text{Cone}(\text{E}(i)))_\ast$

9.5.2 The Mayer-Vietoris Iterative Algorithm

In this section, we introduce our iterative and dimension-independent Mayer-Vietoris (MV) algorithm for computing the $\mathbb{Z}$-homology of any simplicial $d$-complex $\Sigma$ starting from its Pairwise MC-Graph $G_p^\Sigma$, introduced in Section 8.3.2.

Since MC-Decomposition is always decidable and its definition is dimension-independent (see Section 8.2.1), our MV algorithm is completely dimension-independent, at least from a theoretical point of view. However, as discussed in Section 8.2.2, the identification of the Pairwise MC-graph requires the recognition of non-manifold singularities, which is not a decidable operation for simplicial $d$-complexes, with $d \geq 6$ [Nab96]. This operation is mandatory in order to recognize “pinched” singularities (see Section 8.2.2) corresponding to self-loops in the Pairwise MC-graph. As we will prove in this section, self-loops are not interesting in the MV algorithm. Hence, we can avoid to completely recognize non-manifold singularities, and exploit only the dimension-independent algorithm, discussed in Section 8.2.1, in order to identify the Pairwise MC-graph.

As a consequence, we can state that our MV algorithm is completely dimension-independent. In any case, it is a modular algorithm, since we do not introduce limitations regarding the number of top simplices in MC-components.

The Pairwise MC-Graph can be combined with any topological data structure. In this application, we combine the Pairwise MC-Graph with the *Incidence Simplicial (IS)* data structure [DFHPC10], introduced in Section 4.1, in order to simplify computation of incidence matrices related to MC-component. Recall that the IS data structure encodes all the simplices in $\Sigma$, plus boundary relation $R_{p,p-1}$, with $0 < p \leq d$, for each $p$-simplex in $\Sigma$. Also, it encodes partial co-boundary relations $R_{p,p+1}$, with $0 \leq p < d$. In the IS data structure, vertices bounding a $p$-simplex $\sigma$ are sorted according to the lexicographic order of their indices while constructing the IS data structure, as discussed in Section 6.2.7. Also, each face of any $p$-simplex $\sigma$ is generated following the schema introduced in Section 6.1.3. Let $V_\sigma = [v_0, \ldots, v_p]$ be the set of vertices bounding a $p$-simplex $\sigma$, then any $(p-1)$-face $\lambda_1$ of $\sigma$ is generated by discarding vertex $v_1$ from $V_\sigma$. As a consequence, all the
$(p-1)$-simplices bounding $\sigma$ are implicitly oriented, thus it is trivial to generate the corresponding $p$-chain. Therefore, any incidence matrix $D_p$ can be easily computed by visiting boundary relations $R_{p,p-1}$ for any $p$-simplex.

Note that any topological data structure described in Chapter 4 and Chapter 5 can be exploited in the MV algorithm, since all the simplices are generated with respect to the schema introduced in Section 6.1.3. In any case, the topological data structure we use as basis of the MC-Decomposition does not play a key role in the MV algorithm.

The key step of our iterative MV algorithm is the computation of homology for the union of two MC-components $A$ and $B$. Here, we can exploit Algorithm 9.2, introduced in Section 9.5.1, to compute $\mathbb{Z}$-homology of the subcomplex $A \cup B$ from homologies of subcomplexes $A$, $B$, and $A \cap B$. This information is directly encoded in the Pairwise MC-Graph. Specifically, two MC-components $A$ and $B$ are described, respectively, by two nodes $n_A$ and $n_B$, which are connected by arc $(n_A, n_B)$. This arc is related to non-manifold singularities shared between $A$ and $B$, namely it contains $A \cap B$. Thus, a step of our MV algorithm corresponds to the analysis of any arc $(n_A, n_B)$. As a consequence, it is clear that a self-loop $(n_A, n_A)$ does not produce any contribution to computations, since $A \cup A \equiv A$. Thus, it can be discarded. Thus, we can exploit a slight variant of the Pairwise MC-graph, where self-loops are not encoded.

In order to simplify computations, we save homological information of the new component $A \cup B$ in order to be reused in the next steps. Specifically, this operation is equivalent to reuse the reduction equivalence provided as output by Algorithm 9.2. The simplest solution is to slightly modify the second step of this algorithm, and associate a reduction equivalence with each MC-component.

At the beginning, we compute, for each MC-component $N$, a reduction equivalence $N_\ast \Leftarrow N_\ast \Rightarrow E(N)_\ast$, where the right reduction $N_\ast \Rightarrow E(N)_\ast$ is the Homological Smith Reduction of the chain-complex $N_\ast$. Conversely, the left reduction $N_\ast \Leftarrow N_\ast$ is the trivial reduction of the chain-complex $N_\ast$. Note that it is possible to compute this reduction equivalence for each MC-component as a pre-processing step and in parallel. In the following steps, this reduction equivalence is replaced by the reduction equivalence currently associated with any MC-component $N$. In other words, this modified version of the Algorithm 9.2 uses the Cone Equivalence Theorem (see Theorem 9.3.7) instead of the Cone Reduction one (see Theorem 9.3.6) in order to reuse intermediate reduction equivalences, which we have already computed in any previous step.

We repeat these operations until each arc has been visited. Each step is equivalent to collapse an arc $(n_A, n_B)$ and generate a new node $n_{AB}$, which describes $A \cup B$. Then, we merge lists of top simplices in $n_A$ and $n_B$, which we store in $n_{AB}$. We also update arcs incident at $n_A$ and $n_B$, which become incident at $n_{AB}$. Hence, at the end of this algorithm, there is only one node $n_\Sigma$, which corresponds to the input simplicial complex $\Sigma$. Hence, the $\mathbb{Z}$-homology of $\Sigma$ can be retrieved from the reduction equivalence associated with $n_\Sigma$, as performed in Algorithm 9.2.

Note that, at each step, the new component resulting from the union of $A$ and $B$ is not manifold-connected, by definition. As a consequence, the decomposition of the input simplicial complex $\Sigma$
we obtain at any intermediate step is not a valid MC-Decomposition. In any case, the intersections of the components is still composed by a limited number of non-manifold simplices.

Algorithm 9.3 summarizes main steps of our Mayer-Vietoris iterative algorithm for homology computation. Here, this algorithm works on a copy \( G \) of the Pairwise MC-Graph \( G^p_\Sigma \). Note that the graph \( G \) is a Pairwise MC-Graph only at the beginning of Algorithm 9.3.

### Algorithm 9.3 Mayer-Vietoris Iterative Computation

**Input:** the Pairwise MC-Graph \( G^p_\Sigma \) of any simplicial complex \( \Sigma \)

**Output:** the \( \mathbb{Z} \)-homology of \( \Sigma \)

1: let \( G := G^p_\Sigma \)
2: Initialize the reduction equivalence for all nodes in \( G \);
3: while there is more than one arc in \( G \) do
   4: Select a random arc \( a = (n_A, n_B) \) in \( G \)
   5: Apply the modified version of Algorithm 9.2 to \( A, B, \) and \( A \cap B \)
   6: Create a new node \( n_{AB} \) in \( G \) describing the new component \( A \cup B \)
   7: Update all the arcs incident at \( n_A \) and \( n_B \)
   8: Associate the new reduction equivalence with \( n_{AB} \)
9: end while
10: Retrieve the \( \mathbb{Z} \)-homology of \( \Sigma \) from the unique node \( n_\Sigma \) in \( G \)

### 9.6 Experimental Results

In this section, we present qualitative and quantitative results about the MC-Decomposition and our MV algorithm, discussed in Section 9.5. All the digital shapes used in our tests are freely available [GGG09]. We tested our implementations on a workstation with 3.2 Ghz Intel® i7 processor and 16 Gb of RAM. All the timings presented in this section are expressed in milliseconds, and are retrieved through a standard timer in the platform-independent QT Library® [QT08]. We have exploited our implementation of the Pairwise MC-Graph, discussed in Section 8.3.2. We have exploited a not optimized version of the SNF algorithm [Dam08].

First, we recall one of the most important properties of the MC-Decomposition, which is critical for efficiency of homology computations. Note that our MV algorithm operates on decomposed shapes and computes homology of the input model from homologies of MC-components and of their intersection through Mayer-Vietoris sequences. Thus, in order to reduce redundancies during homological computation, it is mandatory to use a decomposition which minimizes the number of simplices in intersection subcomplexes. Our experimental results in Section 8.4 show that the size of intersection complexes, corresponding to singularities shared by two MC-components, is very small in comparison with the number \( S_\Sigma \) of simplices in the input shape. In Table 9.1, we propose a subset of shapes where the largest intersection \( MA_\sigma \) between two MC-components never exceeds 5% of \( S_\Sigma \). This property makes the MC-Decomposition suitable as basis for the MV algorithm.

Our MV algorithm has been designed for computing \( \mathbb{Z} \)-homology of any arbitrary shape, including
Table 9.1: Statistics about the Pairwise MC-Graph of several non-manifold 2D shapes formed by $s_0$ vertices, $s_1$ edges, and $s_2$ triangles. Their corresponding Pairwise MC-Graph has $n^P$ nodes and $a^P$ arcs. It is interesting to compare size $MS_c$ of the largest MC-component and size $MA_a$ of the largest intersection between two MC-components, both expressed as a percentage of the total number of simplices $S_Σ = s_0 + s_1 + s_2$ in the input shape.

We have compared our MV algorithm with the classical SNF algorithm, which is the most general method for computing $\mathbb{Z}$-homology. Recall that the SNF algorithm computes incidence matrices from the entire shape. Conversely, our MV algorithm uses the SNF algorithm to compute homology on MC-components. In our current implementation, we exploit the classical SNF algorithm without any optimization [Dam08]. In any case, it is possible to use any other version of the SNF algorithm.
algorithm. Our experimental results, summarized in Table 9.2, show that the MV algorithm is a reasonable tool for computing $Z$-homology on simplicial shapes, since it require less space than the SNF algorithm, and provides a relevant speed-up to computations.

The key point in our storage analysis is the size of incidence matrices of any simplicial $d$-complex $\Sigma$. Recall that any incidence matrix $D_k$ of dimension $k$ relates chain-groups $C_{k-1}$ and $C_k$, with $1 \leq k \leq d$. Thus, it requires $I_k = s_k \times s_{k-1}$ integer values (encoded on 4 bytes), where $s_k$ and $s_{k-1}$ denote, respectively, the number of $k$-simplices and $(k-1)$-simplices in $\Sigma$. The SNF algorithm needs incidence matrices $D_k$, with $1 \leq k \leq d$, thus its storage cost $SNF_s$ is $O(\sum_{1 \leq k \leq d} I_k)$. Conversely, our MV algorithm computes, at each step, homology of the union of two MC-components $A$ and $B$, plus homology of their intersection. Since their intersection is usually very small, we can ignore it. Thus, we operate only on incidence matrices of a MC-component $X$, which we denote as $D_k^X$, with $1 \leq k \leq \dim(X)$. We denote the number of elements in $D_k^X$ as $I_k^X$. In any step of our MV algorithm, we operate only on a pair of MC-components $A$ and $B$, hence we need $O(\sum_{1 \leq k \leq d} I_k^A + I_k^B)$ integer values for encoding their incidence matrices. If we consider the size of incidence matrices for the largest MC-component, denoted as $MI_k$, with $1 \leq k \leq d$, then storage cost of the MV algorithm $MV_k$ becomes $O(\sum_{1 \leq k \leq d} MI_k)$. This property demonstrates that the MV algorithm requires much less space than the SNF algorithm. In our tests, summarized in Table 9.2, we have obtained a reduction $R_s$ of at least 55% of $SNF_s$. In other words, the MV algorithm requires at most 45% of $SNF_s$.

We also provide timing comparisons between our MV algorithm and the SNF one, demonstrating that we obtain a relevant speed-up with our approach. In our tests, the MV algorithm is at least 1.6 times faster than the SNF algorithm, as shown by data in the column $R_t$ in Table 9.2.

<table>
<thead>
<tr>
<th>Shape</th>
<th>$(H_0,H_1,H_2)$</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$SNF_s$</th>
<th>$SNF_t$</th>
<th>$MI_1$</th>
<th>$MI_2$</th>
<th>$MV_s$</th>
<th>$MV_t$</th>
<th>$R_s$</th>
<th>$R_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bi-Twist</td>
<td>$(Z,Z^4,Z^7)$</td>
<td>26</td>
<td>54</td>
<td>80</td>
<td>$1.2 \times 10^7$</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>$3 \times 10^6$</td>
<td>73%</td>
<td>3.8</td>
</tr>
<tr>
<td>Carter</td>
<td>$(Z,Z^{27},Z^7)$</td>
<td>190</td>
<td>377</td>
<td>567</td>
<td>$7.7 \times 10^7$</td>
<td>41</td>
<td>75</td>
<td>116</td>
<td>$1.7 \times 10^7$</td>
<td>79%</td>
<td>4.5</td>
</tr>
<tr>
<td>Chair</td>
<td>$(Z,0,Z^7)$</td>
<td>0.2</td>
<td>0.4</td>
<td>0.6</td>
<td>60</td>
<td>0.03</td>
<td>0.04</td>
<td>0.07</td>
<td>19</td>
<td>88%</td>
<td>3.2</td>
</tr>
<tr>
<td>Twist</td>
<td>$(Z,Z^2,Z^7)$</td>
<td>16</td>
<td>34</td>
<td>50</td>
<td>$2.2 \times 10^6$</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>$1.4 \times 10^6$</td>
<td>55%</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Table 9.2: Comparisons in terms of timings and storage costs between the SNF and MV algorithms, which compute the $Z$-homology $(H_0,H_1,H_2)$ for several arbitrary shapes. Columns $I_1$ and $I_2$ indicate the size (in Mb) of incidence matrices for the entire shape. Columns $SNF_s$ and $SNF_t$ show, respectively, the storage cost (in Mb) and timing (in ms) required by the SNF algorithm. Columns $MI_1$ and $MI_2$ show the size (in Mb) of incidence matrices for the largest MC-component. Columns $MV_s$ and $MV_t$ show, respectively, the storage cost (in Mb) and timing (in ms) required by the MV algorithm. We also provide the reduction of storage cost $R_s$ (expressed as a percentage of $SNF_s$), and the ratio $R_t$ between $SNF_t$ and $MV_t$.

However, advantages introduced by the MV algorithm can be slightly reduced in some cases, as shown in Table 9.2 for the Twist model. Since the MC-Decomposition does not impose any limitation on components size, it is possible to obtain a “large” MC-component. Thus, computing homology of this MC-component through the SNF algorithm may be time consuming.
This issue could be overcome in different ways. For instance, we can reduce size of MC-components through simplification algorithms which preserve its topology and handles non-manifold singularities. In this way, generators of any MC-component can be computed only on a simplified version. However, if this component has to be merged with another MC-component by the MV algorithm in a later iteration, generators have to be expressed in the non-simplified MC-component in order to ensure consistency of the computation during the union. It is also possible to gain in efficiency by using one of the existing optimizations of the SNF algorithm for sparse integer matrices [Gie96, DHSV03, JW09]. As noted before, our framework does not depend on the SNF algorithm, and it can work with any optimized version of this latter.
Chapter 10
Concluding Remarks

We have addressed the problem of representing arbitrary non-manifold shapes, namely non-manifold subsets of the Euclidean space, formed by parts of different dimensions. Modeling arbitrary shapes requires efficient representations for their discretized version as well as topological methods for analyzing their structure. In this context, the objective of our research has been twofold. On one side, we have investigated new effective data structures for modeling arbitrary shapes discretized as simplicial or cell complexes. On the other side, we have considered topological decompositions of arbitrary shapes into manifold or almost manifold parts in order to deal with the intrinsic complexity of arbitrary shapes.

Specifically, we have introduced the Incidence Simplicial (IS) data structure [DFHPC10], and the Generalized Indexed Data Structure with Adjacencies (IA*) [CDFW11], which are dimension-independent data structures for representing abstract simplicial complexes. Specifically, the IS data structure encodes all simplices in any arbitrary shape, plus a subset of incidence relations. Thus, it is suitable for those applications, like CAD and FEM applications, where it is mandatory to access and associate attributes with each simplex. Conversely, the IA* data structure encodes only vertices and top simplices in any arbitrary simplicial complex, plus a subset of adjacency relations. We have compared the storage cost of these data structures with several dimension-specific and dimension-independent representations. Our tests show that the IS and IA* data structures are more compact than existing representations.

Also, we have designed the dimension-independent Mangrove Topological Data Structure (Mangrove TDS) framework, which supports a wide number of topological data structures under a common application interface. In this framework, a topological data structure is described through a graph-based representation, which we call a mangrove. Also, the Mangrove TDS framework provides an implicit description for any simplex not directly encoded in a topological data structure, which we call a ghost simplex. In order to prove the validity of our approach, we have implemented and compared several data structures, including the IS and IA* data structures. Our tests show that ghost simplices can improve the expressive power of several data structures which do not
encode all simplices, as well the IA* data structure. The complete implementation of the Mangrove TDS framework, including all the data structures, is contained in the platform-independent Mangrove TDS Library.

In this thesis, we have also considered the Manifold-Connected Decomposition (MC-Decomposition) of arbitrary simplicial shapes. The basic concepts underlying this decomposition, but limited to 2D and 3D simplicial complexes, have been introduced in [HDF07a, HDF07b]. In [CDF11] we have proposed several two-level graph-based representations of the MC-Decomposition, which can be combined with any topological data structure in the same way as a spatial index overimposed on an arbitrary shape. These graph-based data structures are suitable for several applications. Specifically, we can decouple the representation of MC-components, provided by any topological data structure, from the structural model of the input shape, provided by the MC-Decomposition. Also, these representations allow for a unique vertex ordering for all MC-components. This condition is a requirement in several applications, like computation of simplicial homology, which we have also investigated in this thesis.

Recently, computing topological invariants of a shape has drawn much attention in computational topology, since the homeomorphic information useful for characterizing and understanding shapes. For instance, the generation of simulation models still lacks capabilities for processing non-manifold shapes, like idealized representations [TCMT56, LF05, TBG09]. Homological information on arbitrary shapes can strongly support new modeling capabilities, because constructive modeling techniques are often used. Also, topological features are extremely important in high dimensional data analysis, where pure geometric tools are usually not sufficient.

Classical techniques for computing simplicial homology exploit an algebraic approach [Ago76], which does not yield to an iterative algorithm [Ser94]. Conversely, the Constructive Homology Theory [Ser99, SR06] offers an elegant way for iteratively computing homology of a shape from homologies of its subcomplexes and their intersections. We have exploited this approach in order to design the Mayer-Vietoris (MV) algorithm [BCMA+11]. The MV algorithm retrieves the complete homological information from the MC-Decomposition of any arbitrary shape, relating homologies of MC-components and their intersections. In this case, the intersection among MC-components is given by any non-manifold singularities, and their size is usually limited. We have experimentally shown that MC-Decomposition increases efficiency of the MV algorithm, which results more efficient than classical approaches [Mun99, Ago05].

As a consequence, it is clear that our research may be extended in several directions. Here, we summarize the most relevant ones.

In our current work, we have defined the effect of a simplification operator for simplicial complexes, known as Vertex-Pair Collapse (VPC) [PH97] and defined for simplicial complexes in arbitrary dimensions encoded as an incidence graph, on the IS and IA* data structures. Our current implementation of the Mangrove TDS Library does not support any editing operator on mangroves. We plan to implement the VPC operator in the library. With this modification, it will be possible to define and implement a multi-resolution model for arbitrary shapes, by extending the Non-
**Manifold Multi-Tessellation (NMT)** model [DFMPS04], currently restricted to arbitrary simplicial 2-complexes embedded in the Euclidean space \( \mathbb{E}^3 \).

Moreover, the IS and IA* data structures provide effective representations non-manifold shapes, but they are limited only to simplicial complexes. Specifically, they may be extended to arbitrary shapes discretized by cell complexes. Specifically, there is an increasing interest in structured hexahedral meshes, since they become really important in the FEM and CAD analysis. Therefore, a possible future work should be extending our data structures in order to represent abstract cell complexes. This extension should be quite straightforward for some of our data structures. Moreover, our Mangrove TDS framework is extensible, thus it is possible to add any data structure without changing the structure of the framework.

The MC-Decomposition of any simplicial complex \( \Sigma \) is suitable to retrieve a *semantic decomposition* of \( \Sigma \). Specifically, our current work consists of defining and implementing the *Shell-based Decomposition* of any simplicial complex \( \Sigma \) of dimension up to 3, embedded in the Euclidean space \( \mathbb{E}^3 \). The key idea of this decomposition consists of identifying components which are relevant from the semantic point of view. Specifically, it is possible to combine together into closed components MC-components forming 2-cycles (*shells*) in \( \Sigma \). Preliminary results on this decomposition can be found in [HDF07a, HDF07b]. Shell-based Decomposition may be a basis for applications related to shape reasoning [DFHP06], shape annotation [DFHP+07, PDFH07, PDF09], and computation of topological invariants [DFPH10]. An other possible application is in the context of FEM models generated through idealization operators. In this case, it is possible to detect form features, like protrusions, depressions, handles, or through-holes, by considering combinatorial and topological structure of this decomposition. Preliminary classifications of form features have been presented in [DFHG08, LDFH09].

Finally, also our work on homology computation can be extended in several directions. Specifically, it possible to improve our current implementation of the SNF algorithm in order to increase efficiency of homology computations for each MC-component. Hence, it will be possible to process very large models. For instance, it is possible to use existing optimizations of the SNF algorithm for sparse integer matrices [Gie96, DHSV03, JW09]. Also, an other possible future avenue consists of investigating how to combine the MV algorithm with a different approach for computing homology of MC-components, since these latter can be viewed as almost manifold complexes. Note that there are several techniques for computing homology on manifold shapes, thus more efficient algorithms on manifold shapes could be applied. Finally, several strategies for improving any geometric properties of generators may be applied. The key idea is to provide the shortest set of generators. A possible solution is to minimize the length of such generators, by associating a metric to homology basis, as recently proposed in [DSW10].
Bibliography


102, 1993.

[PDF09] L. Papaleo and L. De Floriani. Semantic-based Segmentation and Annotation of
15th International Conference on Image Analysis and Processing*, number 5716 in

[PDFH07] L. Papaleo, L. De Floriani, and J. Hendler. Bridging Semantic Web and Digital


[PIK+09] S. Peltier, A. Ion, W. Kropatsch, G. Damiand, and Y. Haxhimusa. Directly Com-
puting the Generators of Image Homology using Graph Pyramids. *Image & Vision


1985.

[PTL04] S. Pesco, G. Tavares, and H. Lopes. A Stratification Approach for Modeling Two-


[RC99] J. Rossignac and D. Cardozo. Matchmaker: manifold BReps for Non-manifold R-
sets. In *Proceedings of the 5th ACM Symposium on Solid Modeling and Applications*,


[RO89] J. Rossignac and M. O’Connor. A Dimension-independent Model for Point-sets with

[RSS01] J. Rossignac, A. Safonova, and A. Szymczak. 3D Compression made Simple: Edge-
breaker on a Corner Table. In *Proceedings of the Shape Modeling International*,


