Dipartimento di Informatica e
Scienze dell'Informazione

Spatial Operations on Multiresolution Cell Complexes

by

Paola Magillo

Theses Series DISI-TH-1999-03

DISI, Università di Genova

v. Dodecaneso 35, 16146 Genova, Italy http://www.disi.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica e
Scienze dell'Informazione

Dottorato di Ricerca in Informatica

Ph.D. Thesis in Computer Science

Spatial Operations on Multiresolution Cell Complexes

by

Paola Magillo

April, 1999

Dottorato di Ricerca in Informatica
Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova

DISI, Univ. di Genova
via Dodecaneso 35
I-16146 Genova, Italy
`http://www.disi.unige.it/`

Ph.D. Thesis in Computer Science

Submitted by Paola Magillo
DISI, Univ. di Genova
`magillo@disi.unige.it`

Date of submission: December 1998

Title: Spatial Operations on Multiresolution Cell Complexes

Advisor:

Prof. Leila De Floriani
DISI, Univ. di Genova
`deflo@disi.unige.it`

Ext. Reviewers:

Prof. Jorg-Rudiger Sack
School of Computer Science, Carleton University, Ottawa, Canada
`sack@scs.carleton.ca`

Prof. Hanan Samet
Computer Science Department, Univ. of Maryland, College Park, USA
`hjs@umiacs.umd.edu`

Abstract

Cell complexes are used for modeling geometric entities in several application fields. In this approach, an object of arbitrary shape is represented through an aggregation of simple, basic shapes, called cells. A high number of cells is necessary to obtain accurate descriptions, thus involving high costs for managing a complex. Costs can be reduced by locally adapting the resolution of a complex to the needs of each task. A multiresolution complex is a structure which encodes alternative representations of a geometric entity at different resolution, and can be inquired by the user to retrieve any of such complexes. Several multiresolution models for cell complexes have been proposed in the literature, but the field still lacks of a unifying framework. In this thesis, we propose a general model for multiresolution cell complexes, which encompasses most existing proposals, and we define standard access operations for such a model. We specialize the model and its operations for case-study applications, with the design of data structures and algorithms: selected applications include the representation of terrains, of free-form surfaces, of three-dimensional scalar fields, and of the volume of solid objects. Experimental results from prototype implementations are reported.

To Bunny, a happy animal who probably does not know
to have been turned into a triangle mesh (see Figure 5.14).

Acknowledgements

I wish to thank to my supervisor Prof. Leila De Floriani for her suggestions, to Prof. Enrico Puppo for helpful discussions, to the external reviewers Prof. Hanan Samet and Prof. Jorg-Rudiger Sack. I also want to thank to students and collaborators of the Department of Computer and Information Science of the University of Genova for their contributions to the work presented here, especially for implementing parts of the software library; among them a special thank to Dr. Luca Altea, Dr. Vittorio Bertocci, Dr. Raffaella Pelizza and Dr. Daniele Smoquina.

Table of Contents

Chapter 1

Introduction

The need for representing and manipulating models of spatial objects arises in many applications of computer science. Acquired data from physical or medical sciences need to be interpreted, analyzed, and displayed in some suitable graphical format; models of geometric objects acquired through scanning equipments or generated by CAD systems need to be manipulated (e.g., edited), analyzed, and visualized, possibly interactively, in real-time. Multiresolution geometric models support representation and processing of spatial entities at different levels of detail. Such representations have received recently much of attention in the literature because of their potential impact on applications, such as terrain modeling in geographic information systems, scientific data visualization, virtual reality, etc.

The internal representation for geometric entities must be discrete (in order to be processed by a digital computer), i.e., it must consist of a finite number of values of simple type. Roughly speaking, we can have two opposite approaches to discrete representations: either providing a single analytical expression describing the whole entity, or decomposing the entity into many parts, called *cells*, each described separately and independently of the others. The two approaches are illustrated in Figure 1.1 (a) and (b) in a very simple case, i.e., the construction of a model for a curve that is known to pass through a given set of points: in Figure 1.1 (a), we use a unique interpolating function of high degree; in Figure 1.1 (b), we decompose the curve into cells consisting of segments and use a linear interpolant on each cell.

In the first approach, the problem may easily become intractable because of the high number of coefficients required to define the solution, especially for large data sets. In the second approach, we must define both the entity decomposition and each of the independent entity descriptions; here, the main concern will be the representation of the cells, since usually cells with a simple shape and simple local descriptions are used. An important advantage of this approach is that the representation inside each cell is independent of the others. In the example of the curve depicted in Figure 1.1, the first approach would require the use of the whole data set to evaluate the function at a single point, or within a window; the second approach only

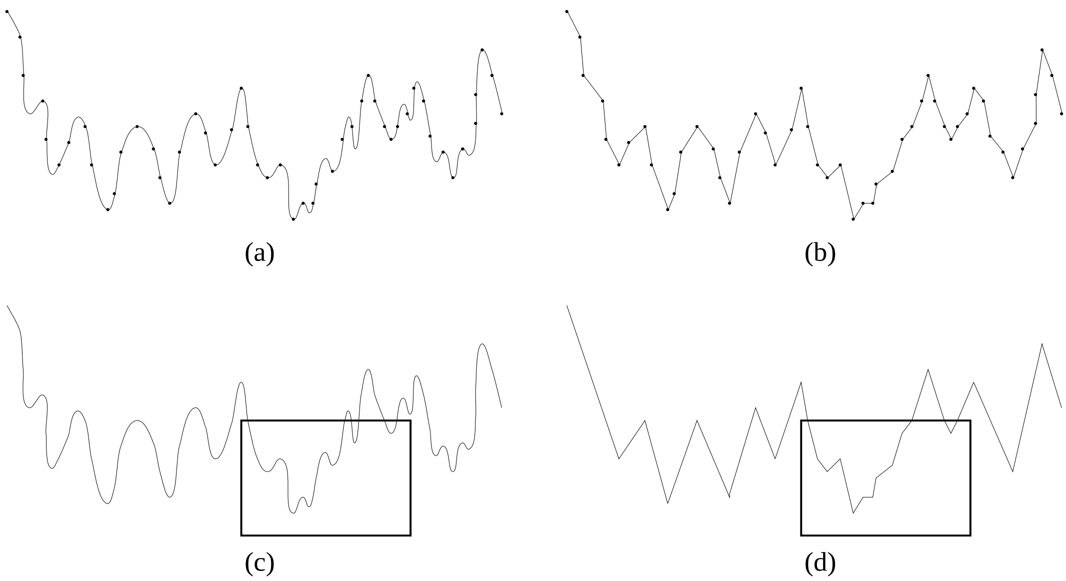


Figure 1.1: A curve interpolating a set of data points represented (a) as a single polynomial curve, and (b) piecewise as a connected sequence of straight-line segments. Focusing the attention inside the rectangular window still requires using all data in the first case (c), while only the relevant segments are used in the second case (d): outside the window we could, for instance, represent the line at a coarser resolution.

needs the endpoints of the relevant segment(s) (see Figure 1.1 (c) and (d)).

However, also the second approach may be inefficient for very large data sets, due to the high resolution (i.e., the large number of cells) of the subdivision. Typically, an application using a geometric model focuses just on some parts of the object at each time. For instance, in real-time visualization of a virtual environment, only the parts of the scene lying inside the view frustum are of interest. Multiresolution techniques have been developed as tools for adapting the resolution of a geometric model to the changing needs of an application, in real-time.

This thesis deals with *multiresolution models of spatial entities based on cell complexes*. In the remainder of this introduction we briefly explain the basic concepts used in the thesis (cell complexes, variable resolution, multiresolution models), illustrate motivations and goals, outline its main contributions, and, finally, provide an overview of all chapters.

1.1 Some Basic Concepts

A *cell complex* (or, simply, a *complex*, for short) is a compound geometric object formed by the aggregation of a finite number of basic entities, called *cells*, embedded in a d-dimensional space. Cell complexes have an impact on a variety of applications where they can be used as *discrete models* of physical objects or phenomena for which a continuous representation is either not available, or is too complicated to be manipulated directly.

For example, two-dimensional complexes with polygonal cells are used for describing the boundary of solid objects. In particular, two-dimensional simplicial complexes (i.e., triangle meshes) are used for representing surfaces in rendering, and for representing terrains in geographic applications. In solid modeling, three-dimensional complexes are used for representing the interior of an object. d-Dimensional simplicial complexes are used as approximate representations of scalar or vector fields (for instance, in domain discretization techniques in finite element analysis), and so on.

The use of cell complexes in geometric modeling fits in the well-known approach of solving a difficult problem (i.e., the one of describing a complicated shape) by decomposing it into several instances of a simpler problem (i.e., the one of representing a cell).

The *accuracy* of a cell complex in representing a spatial object depends on the size of its cells: a parameter that we call the *resolution* of the complex. In order to obtain accurate descriptions, a high resolution, and thus a high number of cells, is needed. This may lead to high costs for storing and/or processing the complex at high resolution.

Not all application tasks accessing a geometric entity, which is represented as a cell complex, require the *maximum* accuracy in *every part* of the entity. This suggests that costs can be reduced by *locally adapting* the resolution of a complex to the specific needs of each task within an application. The basic principle is that the parts of the object considered of low importance for the application can be represented by fewer and hence bigger cells. However, the important parts should still be described by using a sufficiently high number of cells. This helps to reduce the overall number of cells of the complex which must be handled (also called the *size* of the complex), and thus the resources needed for its manipulation.

In an ideal situation, each application task, at each instant in time, should handle a cell complex that describes the original object with a suitable accuracy and with an acceptable number of cells. This cell complex should implement the best compromise between the need of having a *sufficiently high accuracy*, at least in the important parts of the object, and the need of having a *sufficiently small number of cells* to manage the problem (e.g., in rendering and analysis) with the available resources (memory space, execution time constraints, etc.). For example, in a flight simulator, a terrain must be rendered with many details only close to the observer. Since the viewpoint is continuously moving, the required accuracy is changing from frame to frame, and visualization must occur at frame rate in order to support interactivity (real time).

There are two alternative ways to approximate this ideal situation:

1. *On-the-fly construction of a complex.*

   A new cell complex is built on-line, from scratch, whenever the application requirements concerning accuracy and size of the complex change.

2. *Use of a multiresolution model.*

   A comprehensive structure is built off-line, which organizes several cell complexes representing the same object at different resolutions, and is capable of extracting on-line the complex which best reflects some given requirements in accuracy and size.

The process of building a cell complex from scratch, starting from the available data for an object, is usually time-consuming. Sophisticated techniques are required, for instance, to decide where to place the vertices, how to optimize the shape of the cells, how to estimate and bound the approximation error. Besides the computational time, an additional drawback is the need to permanently maintain the original data along with the current complex.

When using the approach based on multiresolution models, alternative representations of an object at several different resolutions are precomputed and stored. Thus, most of the time-consuming operations are performed off-line in a preprocessing step. In an on-line phase, one simply searches the cell complex which best adapts to the current application requirements *among those available from the model.*

Since the choice is somehow limited, the cell complex obtained from a multiresolution model may not be as "good" as that built on-the-fly according to the same application requirements: it may either over-refine areas which are not of interest (in order to sufficiently refine interesting ones), or under-refine interesting areas (in order to remain within the maximum number of cells allowed). On the other hand, querying a multiresolution model is much faster than building a complex from scratch, so that the time subsequently spent by the application in processing possible extra cells, needed to achieve the same accuracy, becomes negligible (see Figure 1.2). Different multiresolution models show different capabilities of adapting the resulting complexes to the given requirements of accuracy and size.

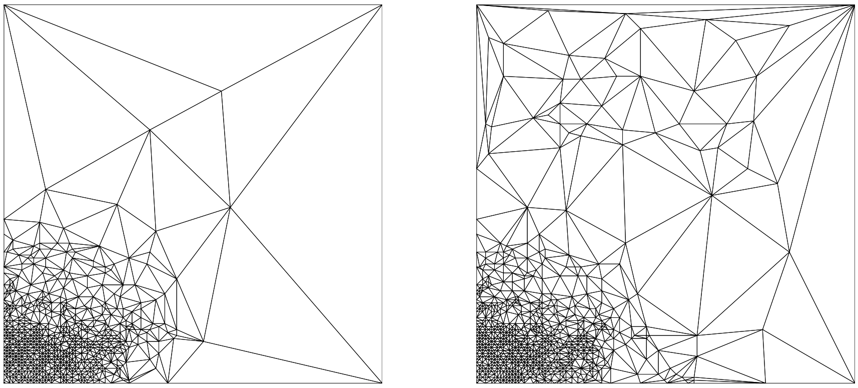
1.2 Motivations and Goals

Several multiresolution models for cell complexes have been proposed in the literature, but the field still lacks a unifying framework, as:

- Most existing models are designed for specific applications, or classes of applications.

  Several models have been proposed for representing surfaces for the purpose of real-time visualization in applications such as virtual reality and interactive geometric modeling [Hop96, KS97, XV96, XESV97, Hop97, GTLH98, SZL92, CVM+96, BBCS96, CCMS97].

  Other models have been proposed in the field of terrain modeling [Sam90a, SS92, VHB87, DFP95a, SP92, DWS+97, EKT97, HK95, MMS97, DWS+97, LKR+96, DF89, CPS97, dBD95, Pup96, Pup98].



(a) **(b)**

Figure 1.2: (a) A triangle mesh representing terrain at variable resolution built from scratch, and (b) one extracted from a multiresolution model for the same accuracy requirements. The triangulation in (a) has 1378 triangles, the one in (b) has 1524 triangles, but it is obtained in 1/10 of the time.

A few proposals of multiresolution models of higher-dimensional objects also exist, which extend two-dimensional ones [WvG94, Heb94, BBDFP94, PH97].

- Existing models usually rely on specific construction techniques.

  In most existing multiresolution models, the available representations at variable resolution are constrained to have certain characteristics. Typically, they are the complexes that can be obtained from an initial cell complex by applying a *specific type* of transformation operator. In general, each multiresolution model assumes a certain operator (e.g., vertex insertion / removal [DF89, CDFM$^+$94, dBD95, BDFM95b, KS97], vertex split / edge collapse [Hop96, XV96, XESV97, Hop97, MMS97, GTLH98, KCVS98], etc.).

- Most proposed models are a direct abstraction of the data structure used to implement them.

  An abstract notion is still missing of what a multiresolution cell complex intrinsically is, independently of how it is encoded.

- Access operations are defined based on the application, and are strictly tied to the algorithms which implement them on the data structure.

  Many existing models are designed ad-hoc for certain operations (i.e., the ones typical of the application for which they have been developed), while other operations are not supported at all.

  For instance, multiresolution models developed in the field of terrain modeling [Sam90a, GG79, Ore91] are well suited as spatial indexes for operations such as point location, nearest neighbor search, etc., but they fail in providing a global representation of the terrain at a variable resolution, or require ad-hoc solutions in order to do that [VHB87, SS92, DFMP94].

In the absence of a general specification of the fundamental characteristics, properties and operations of a multiresolution cell complex, it is even hard to provide a comparison of existing models. For instance, it does not make much sense to say that a certain model is more compact than others, or it is faster in supporting certain operations, when, for instance, the variable-resolution cell complexes available in it have different characteristics (e.g., in the two dimensional case some models provide Delaunay triangulations, others provide meshes of right triangles, etc.), or the set of operations supported is different (e.g., some models are designed for efficient rendering, others for efficient spatial search, etc.).

The fundamental goal of this thesis is the definition of *a general framework for multiresolution cell complexes* by providing a model for multiresolution cell complexes and standard, application-independent operations acting on it. The general framework can then be specialized in the context of each application by defining appropriate instances of the model and of the operations. Most existing models reduce to instances of the general framework. The second goal of the thesis is proving the effectiveness of the model by presenting specialized implementations of the general framework within a set of case-study applications.

1.3 Contributions of the Thesis

The main contributions of this thesis are in two directions:

- From a theoretical point of view, we formally define a general model for multiresolution cell complexes and its basic access operations in a way that does not depend on the application context or on the specific algorithms or data structures that are intended to be used.

  The model is an extension of the one proposed first by Puppo [Pup96, Pup98] for two-dimensional triangulations. The basic idea underlying our model, called the *Multi-Complex* (MC), is that of having a partially ordered set of modification patterns specifying how an initial coarse cell complex can be iteratively refined in order to obtain more and more accurate representations. The partial order captures dependency relations between modifications that affect the same portion of the complex. Modifications involving distinct parts of the complex are independent. This framework permits to select subsets of modification patterns which are consistent with the partial order so that their application to the initial cell complex gives a variety of complexes at different and, in general, variable resolutions.

  The Multi-Complex is independent of the specific type of modification pattern. Existing multiresolution models reduce to special cases of MCs where modification patterns have certain characteristics and/or the partial order is encoded in a certain type of data structure.

We define basic access operations on an MC is a modular way. At the user-interface level, we pose two general query operations. Intuitively, the first query permits to extract a cell complex which describes the spatial object represented in the MC at a certain, user-specified, resolution, accuracy, or size. The second query permits, in addition, to obtain a representation restricted to a certain, user-specified, area of space. The wide range of operations used in the various application domains are obtained by varying the parameters of the query, i.e., the interesting area and the resolution constraints.

- From an application point of view, we propose data structures and algorithms for implementing the Multi-Complex and using it in real-life situations. We also propose techniques for building an MC starting from data sets typical of a number of case-study applications.

  We present and discuss various approaches to the design of data structures for Multi-Complexes, and related algorithms operating on them, addressing both general data structures, compressed data structures for certain classes of MCs, and organization of an MC on secondary storage.

  We provide a modular approach to the construction of an MC, which clearly separates the two phases of generation of the set of modification patterns and of construction of the partial order. The set of modification patterns can be easily obtained as a byproduct of existing iterative algorithms for mesh simplification.

  A relevant part of the ideas exposed in this thesis have been implemented and tested, and we show results from our prototype implementations.

1.4 Thesis Outline

The core of this thesis consists of seven chapters, numbered from 2 to 8, and it is organized as follows.

In **Chapter 2**, we introduce some background notions that will be used throughout the thesis.

Based on classical concepts from geometry and topology, we define the kind of geometric objects that we intend to model through cell complexes at multiple resolutions, and we briefly discuss cell complexes and their basic properties. Special kinds of cell complexes (such as simplicial complexes, and grids) are introduced for their relevance in applications. In particular, *simplicial* cell complexes reveal to be the most suitable for representing arbitrarily shaped objects. We introduce reconstruction and discretization problems as the two main application contexts for geometric modeling through cell complexes. We also define what we mean for resolution and accuracy of a cell complex in representing a given spatial entity. In the last part of the chapter, we describe a number of *applications* of cell complexes: these include terrain models in Geographic Information Systems (GISs), models of surfaces and solid objects in Computer Aided Design (CAD) and Virtual Reality (VR), and models of multi-dimensional fields in scientific visualization. We also treat the problem of the compression of a complex, for which a possible solution relies on the use of multiresolution techniques, and therefore it is strictly related to the subject of this thesis.

Chapter 3 provides an overview of previous and related work existing in the literature.

We start by reviewing existing methods for building *approximated* cell complexes for a given spatial entity. These methods are considered in the thesis since many of them (namely, the ones using an iterative approach) are the basis for the development of multiresolution models. The central part of the chapter is devoted to existing proposals for multiresolution models of cell complexes. These are described and discussed following a classification into layered models, nested models, and evolutionary models. Finally, we review existing literature on compression of triangle meshes.

In **Chapter 4**, we present our general-purpose multiresolution model, called the *Multi-Complex* (MC).

We formally define the model and we prove a number of relevant properties of it. In particular, we characterize the set of all cell complexes at variable resolution available in an MC in terms of consistent sub-structures of the model itself, that we call sub-MCs. We define the expressive power of an MC as the number of sub-MCs it provides, and introduce a few desirable properties for a Multi-Complex. In the last part of the chapter, we show how existing multiresolution models, reviewed in Chapter 3, can be seen as special cases of MCs. We show that each model reduces to a class of MCs characterized by a certain expressive power and by certain of the desired properties.

In **Chapter 5**, we introduce general query operations on Multi-Complexes.

Given an MC for a spatial object, the basic query operation, called a *global query*, intuitively consists of extracting a cell complex which represents the whole object at a certain resolution, with a certain accuracy, or with a certain number of cells. We also define a *local query*, which generalizes usual geometric queries performed on single-resolution cell complexes: it consists of retrieving a representation of the portions of the object that interfere with a certain area of interest in space. We show that operations of practical interest for the applications are instances of such general operations. We define two algorithms for answering the global query either *statically* (i.e., computing the solution of each new query from scratch) or *dynamically* (i.e., computing the solution of a new query by updating that of a previous query). For the local query, we define an approach which guarantees that the output cell complex can always be extended to represent the whole object outside the area of interest, and a heuristic approach that provides a representation just inside the area of interest. The chapter ends with a specification, in an object-oriented style, of the basic access operations of a Multi-Complex that will allow the implementation of the query algorithms.

In **Chapter 6**, we propose and discuss various data structures for representing a simplicial Multi-Complex, and the related implementations of basic operations and query algorithms.

We propose *three basic approaches* to the representation of an MC, namely as a DAG, as a sequence, or through adjacency relations among its cells. These three families of data structures capture all types of data structures proposed in the literature to represent multiresolution models. Beside general data structures, we propose and discuss a number of *compressed* structures, which we have designed to encode special subclasses of MCs with reduced storage costs. Finally, we address the important problem of representing Multi-Complexes on secondary storage. For the case of two-dimensional MCs representing terrains, we propose a method for organizing a composable set of MCs which reside on separate files, where two or more of them (i.e., the ones contributing to represent a relevant area of space) can be merged into a single MC when loaded into main memory.

In **Chapter 7**, we address the construction of Multi-Complexes. An MC can be built from *any sequence* of incremental modifications of an initial cell complex.

The specific operator used to modify the current cell complex at each step, and the specific strategy used for deciding where to apply such operator in the current cell complex are relevant for determining the expressive power and the properties ("shape") of an MC. We present and compare a few construction strategies that have been implemented for two-dimensional scalar fields (e.g., terrains).

In **Chapter 8**, we illustrate the application of simplicial Multi-Complexes in a selected set of application contexts, namely *terrain modeling*, *free form surfaces* from range scans or discretization of CAD models, and *volume data* for scientific visualization. This chapter includes experimental results from our prototype implementations.

In the conclusions (**Chapter 9**), we briefly summarize the results of the work, and outline future developments and open problems.

Chapter 2

Background

2.1 Introduction

This thesis deals with multiresolution models of geometric objects based on cell complexes. In this chapter, we introduce background concepts that are used throughout the thesis.

We start with basic concepts from geometry and topology. We introduce the classes of geometric entities that will be our representation domain, as well as cell complexes and their properties. Then, we discuss the use of cell complexes in geometric modeling, both in a general setting, and by presenting specific applications as examples.

We define the *resolution* as an intrinsic characteristic of a cell complex, related to the number, size and density of its cells; we define the *accuracy* as a property of a complex with respect to the object that it represents, depending on the "distance" between the complex and the object. Resolution and accuracy (as defined in this thesis) are two different, but strictly related concepts: typically, a high accuracy requires a high resolution, while a low resolution can only provide inaccurate, raw representations. In the literature, the distinction between the concept of resolution and that of accuracy is not always clear.

Cell complexes have a strong impact in many applications, where they are used as *discrete models* of physical objects for which a continuous representation is not available, or it is not suitable to be manipulated directly.

We present some sample application fields of cell complexes, showing, for each of them, the types of objects and of complexes manipulated, the input data from which models are constructed, and the relevant operations. Such fields contain the topics that will be used as cases of study for the multiresolution framework proposed in this thesis.

This chapter is organized as follows:

- In Section 2.2, we review k-manifolds in $\mathbb{E}^d$ (with or without boundary), that formalize the intuitive notions of surfaces and solids, as well as k-dimensional scalar and vector fields defined on them; these entities will be the representation domain of the multiresolution model proposed in this thesis.
- In Section 2.3, we review cell complexes. After introducing general notions, we focus our attention on simplicial complexes, grids, and self-similar complexes, for their impact in geometric modeling.
- In Section 2.4, we consider the use of cell complexes to model spatial objects. We introduce reconstruction and discretization problems as the two main contexts for application of cell complexes, and the concepts of accuracy and approximation error of a cell complex in representing a geometric entity.
- In Section 2.5, we present a number of *applications of cell complexes*. In particular, we discuss modeling of terrains in Geographic Information Systems (GISs), of three-dimensional objects and surfaces in CAD and Virtual Reality (VR), and of volume data in scientific visualization.
- In Section 2.6, we introduce the problem of *compression of geometric models*, which is common to many applications using cell complexes.
- Finally, in Section 2.7, we select some topics within applications presented before, that will be used as cases of study for the work developed in this thesis.

2.2 Basic Geometric Entities

In this section, we define the geometric entities in d-dimensional space that we will model as cell complexes. Some definitions involve non-trivial concepts from geometry and topology. We give just some intuitive definitions here and refer to text books (e.g., [Ago76, Mun84, Hof89]) for precise, more formal definitions.

We are only interested in objects that are closed subsets of $\mathbb{E}^d$; thus, this condition will appear in most definitions, even when such definitions apply to open sets as well.

2.2.1 Surfaces and Solids

We are interested in modeling subsets of $\mathbb{E}^d$ having a uniform dimension. In order to formalize this idea, we give some auxiliary definitions:

Definition 2.1 *Given a closed set $S \subseteq \mathbb{E}^d$, we say that S is* at least k-dimensional *at point p if any neighborhood u_p of p in S* contains *a subset which is homeomorphic either to an open k-dimensional ball, or to half an open k-dimensional ball.*

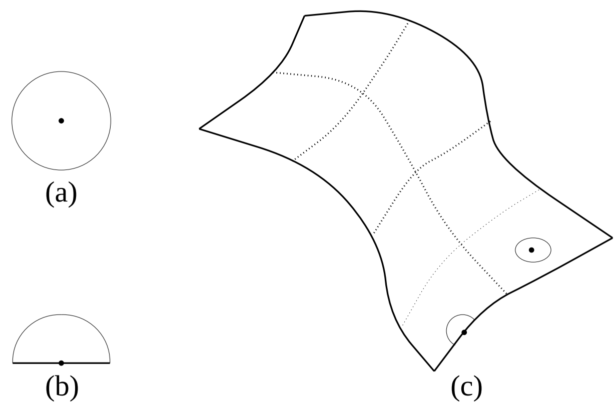


Figure 2.1: (a) A two-dimensional ball, (b) half a two-dimensional ball; thin and thick line denote open and closed boundaries, respectively. (c) A set in $\mathbb{E}^3$ wich is uniformly two-dimensional.

Definition 2.2 *A closed subset S of $\mathbb{E}^d$ is* uniformly k-dimensional *if, for every point $p \in S$:*

1. *S is at least k-dimensional at p, and*
2. *S is not at least $(k+1)$-dimensional at p.*

For instance, a discrete set of points is uniformly zero-dimensional, a set of lines is uniformly one-dimensional, a set of surface patches is uniformly two-dimensional, etc. (see Figure 2.1).

We always consider subsets of $\mathbb{E}^d$ which are uniformly k-dimensional, for some $0 < k \leq d$. For short, usually we omit the term "uniformly" and simply talk about k-dimensional sets.

The *interior* of a uniformly k-dimensional set S is the set of points $p \in S$ whose neighborhood u_p contains a subset homeomorphic to an open k-dimensional ball. The *boundary* of S is the set difference between S and its interior. The boundary of a k-dimensional set is $(k-1)$-dimensional.

In this thesis, we will focus on the representation of a subclass of k-dimensional sets, i.e., that of manifolds (with or without boundary), defined below:

Definition 2.3 *A closed subset S of $\mathbb{E}^d$ is a k-*manifold without boundary *if, for every point $p \in S$, there exists a neighborhood u_p of p in S, that* is homeomorphic to *an open k-dimensional ball.*

Definition 2.4 *A closed subset S of $\mathbb{E}^d$ is a k-*manifold with boundary *if, for every point $p \in S$, there exists a neighborhood u_p of p such that u_p* is homeomorphic to *an open k-dimensional ball, or to half an open k-dimensional ball.*

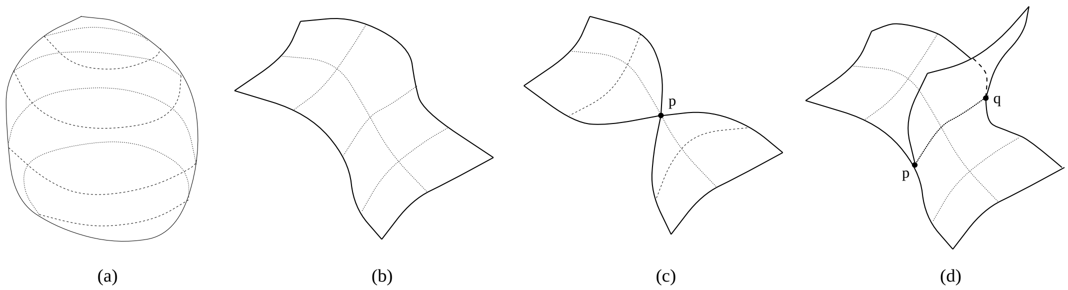


Figure 2.2: Examples of manifold and non-manifold two-dimensional sets in $\mathbb{E}^3$: (a) is a manifold without boundary, (b) is a manifold with boundary, (c) is non-manifold at point p, (d) is non-manifold at segment pq.

Intuitively, a k-manifold is a k-dimensional set whose boundary has no self-intersections. Figure 2.2 shows a 2-manifold without boundary (a), a 2-manifold with boundary (b), and some two-dimensional sets in $\mathbb{E}^3$, which are not manifolds (c,d).

The boundary of a k-manifold with boundary is a $(k-1)$-manifold without boundary.

We call a *surface* a manifold whose dimension is less than that of the embedding space, and a *solid* a manifold having the same dimension as the space in which it is embedded [1]:

Definition 2.5 *A* k-dimensional surface *is a k-dimensional manifold in $\mathbb{E}^d$, with $k < d$.*

Definition 2.6 *A* d-dimensional solid*, or a* solid object *in $\mathbb{E}^d$, is a d-dimensional manifold in $\mathbb{E}^d$.*

Note that, except $\mathbb{E}^d$ itself, all d-dimensional solids have a non-empty boundary. On the contrary, a surface may or may not have a boundary.

2.2.2 Scalar and Vector Fields

A *d-dimensional field* is a function defined over a d-dimensional subset of $\mathbb{E}^d$. In this thesis, we will always consider fields defined on manifold domains, i.e., on d-dimensional solids.

Definition 2.7 *A* d-dimensional scalar field *is a function $\phi : \Omega \subseteq \mathbb{E}^d \longrightarrow \mathbb{E}$, where Ω is a d-manifold in $\mathbb{E}^d$.*

Definition 2.8 *A* d-dimensional vector field *is a function $\phi : \Omega \subseteq \mathbb{E}^d \longrightarrow \mathbb{E}^m$, where Ω is a d-manifold in $\mathbb{E}^d$, and $m > 1$.*

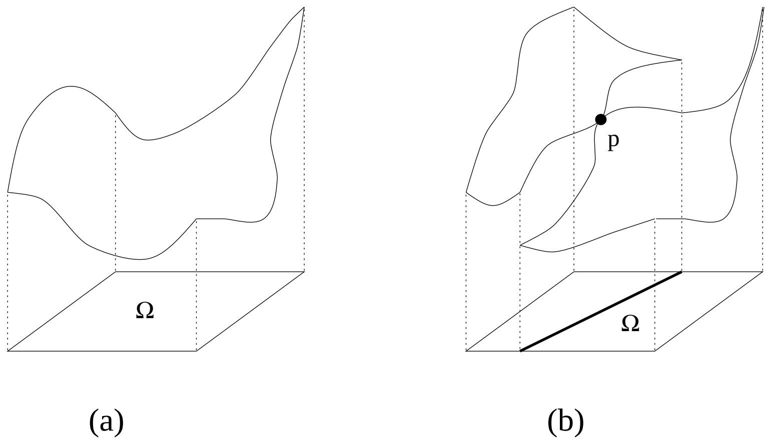


Figure 2.3: Examples of graphs of two-dimensional fields: (a) function ϕ is continuous on Ω, the graph is a surface; (b) function ϕ is discontinuous on a one-dimensional subset of Ω (the thick line), the graph is two-dimensional, but not manifold at point p.

Terrains and *grey-level images* are examples of two-dimensional scalar fields; *volume data*, used in scientific visualization, are examples of three-dimensional fields. A color image described in terms or red, green and blue components is a two-dimensional vector field with values in $\mathbb{E}^3$.

If function ϕ is continuous, then the graph of a scalar or of a vector field is a d-dimensional surface in $\mathbb{E}^{d+1}$ or in $\mathbb{E}^{d+m}$, respectively. If ϕ is discontinuous at most in a subset Ω' of Ω, where Ω' is not at least d-dimensional at any of its points, then the graph of the field is a d-dimensional set, but not necessarily a manifold (see Figure 2.3).

Terrains are usually described by continuous functions, while images usually have jump discontinuities at lines in the domain corresponding to sharp transitions between areas filled with different colors.

2.2.3 Explicit and Parametric Surfaces

Definition 2.9 *An* explicit surface *is the graph of a continuous function* $\phi : \Omega \subseteq \mathbb{E}^k \longrightarrow \mathbb{E}$*, where* Ω *is a* k*-manifold in* $\mathbb{E}^k$*.*

It is easy to see that function ϕ, describing an explicit surface, is a scalar field; thus, an explicit surface is the graph of a scalar field where function ϕ is continuous. An example of explicit surface in $\mathbb{E}^3$ is given by a terrain.

Definition 2.10 *A* parametric patch *is the image of a continuous function* $\phi : \Omega \subseteq \mathbb{E}^k \longrightarrow \mathbb{E}^d$*, where* Ω *is a* k*-manifold in* $\mathbb{E}^k$*.*

[1] Here, we use the term "surface" in a broad sense to denote what elsewhere is called a k-dimensional variety in $\mathbb{E}^d$ [Hof89].

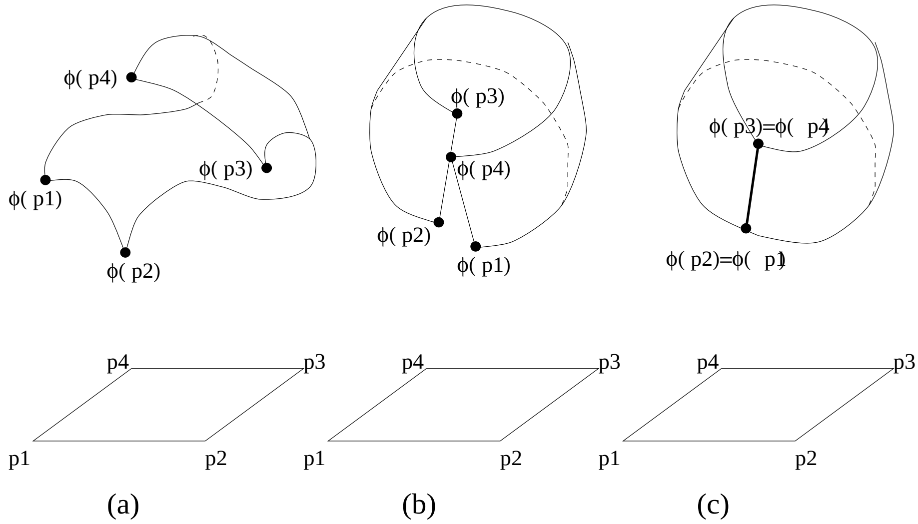


Figure 2.4: Examples of two-dimensional parametric patches: (a) function ϕ is injective, the image is a surface; (b) function ϕ is not injective, the image is not a manifold because of point $\phi(p_4)$; (c) function ϕ is not injective, the image is a manifold.

It is easy to see that function ϕ, describing a parametric patch, is a vector field; thus, a parametric patch is the image of a vector field where function ϕ is continuous.

The continuity of function ϕ is not sufficient to guarantee that the resulting image is a manifold. A sufficient condition is that ϕ is continuous and injective (see Figure 2.4 (a)); if ϕ is not injective, then the image may or may not be a manifold (see Figure 2.4 (b) and (c)).

The space $\mathbb{E}^d$ in which a parametric patch is embedded is called the *physical space*, while the space $\mathbb{E}^k$ containing the domain is called the *parameter space*. In the reminder, we will consider two-dimensional parametric patches (i.e., $k = 2$).

The boundary of the parametric domain Ω of a patch consists of one or more closed lines, called *trimming curves*. Trimming curves may be straight-line segments in parameter space, but their image in physical space is usually a curve segment (as in Figure 2.4). Sometimes, trimming curves are curve segments in parameter space as well, and they are described as parametric lines.

A *parametric surface* is a collection of patches that do not intersect in their interiors, and may touch along their boundaries:

Definition 2.11 *A* parametric surface *is a collection $\mathcal{P}$ of parametric patches such that the interiors of the images of any pair of patches in $\mathcal{P}$ are disjoint.*

Figure 2.5 shows a parametric surface. We assume that a parametric surface is a k-manifold, with or without boundary.

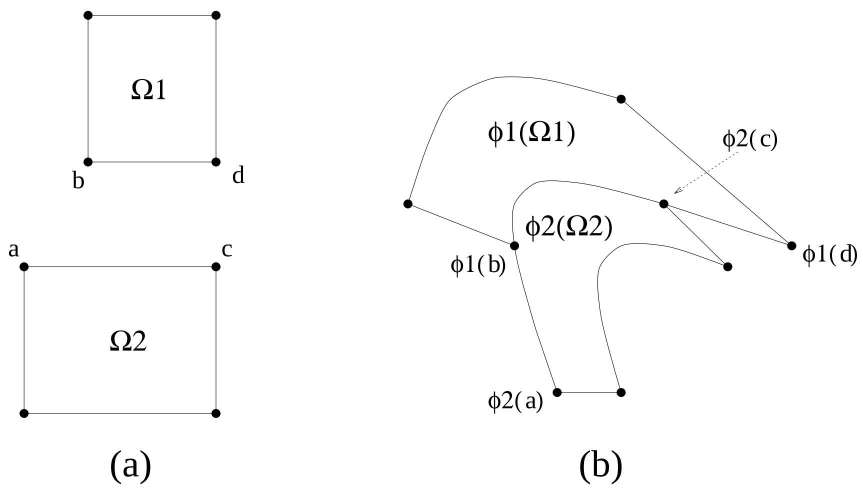


Figure 2.5: A parametric surface consisting of two patches: (a) the domains of the patches in parameter space; (b) their images in physical space.

Note that explicit surfaces are a proper subset of the surfaces of $\mathbb{E}^d$. The term *free-form surfaces* is commonly used to denote generic surfaces which do not necessarily admit an explicit representation.

2.2.4 Surface-on-Surface

The domain of a d-dimensional field is a solid in $\mathbb{E}^d$, thus, it has the same dimension as the space in which it is embedded. The term "surface-on-surface" is used to indicate fields defined on surfaces, thus, on lower-dimensional entities of $\mathbb{E}^d$. This kind of objects occur, for instance, in physics, fluid dynamics, engineering. Examples are modeling temperature on the surface of an engine, rain falls on the Earth, pressure on the wings of an airplane, etc.

Definition 2.12 *A d-dimensional* scalar field on a k-dimensional surface *is a function $\phi : \Omega \subseteq \mathbb{E}^d \longrightarrow \mathbb{E}$, where Ω is a k-dimensional surface in $\mathbb{E}^d$.*

Definition 2.13 *A d-dimensional* vector field on a k-dimensional surface *is a function $\phi : \Omega \subseteq \mathbb{E}^d \longrightarrow \mathbb{E}^m$, where Ω is a k-dimensional surface in $\mathbb{E}^d$, and $m > 1$.*

Under suitable conditions, the graph of function ϕ over Ω is a k-dimensional surface in $\mathbb{E}^{d+1}$ or $\mathbb{E}^{d+m}$, respectively, for a scalar or a vector field.

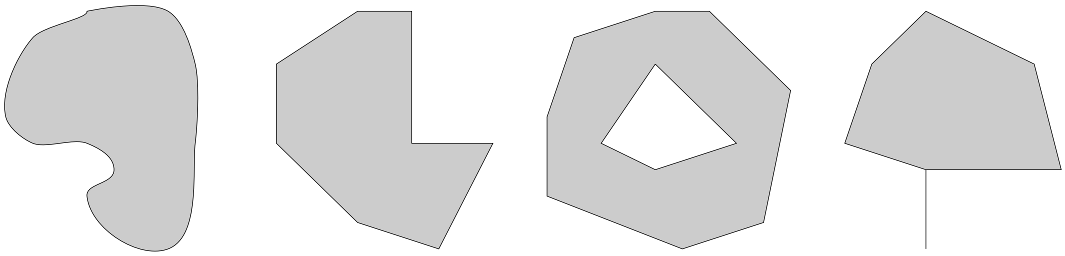


Figure 2.6: Examples of sets that are two-dimensional cells (a,b), and others that are not cells (c,d).

2.3 Cell Complexes and their Properties

We define generic cell complexes first, and then some special cases of complexes which are relevant for the applications: simplicial complexes, fixed-pattern complexes, self-similar complexes, and grids.

2.3.1 Cell Complexes

The basic constituent entity of a cell complex is the cell:

Definition 2.14 *A* k-dimensional cell *(or* k-cell*, for short) in* $\mathbb{E}^d$ *is a subset of* $\mathbb{E}^d$ *homeomorphic to a closed* k*-dimensional ball.*

Figure 2.6 shows some examples of two-dimensional cells. We define cells as closed sets for convenience; an alternative approach defines cells as open sets. Note that a point is a zero-dimensional cell.

A k-cell is a uniformly k-dimensional set; the interior of a zero-dimensional cell (point) p is p itself, and its boundary is empty.

We denote with $\mathbb{\Gamma}_d$ the set of all cells of $\mathbb{E}^d$, of any dimension, and with $\mathbb{\Gamma}_d^{(k)}$ the set of k-cells of $\mathbb{E}^d$. Given a set Γ of cells in $\mathbb{E}^d$, their union $\cup\{\gamma \mid \gamma \in \Gamma\}$ is called the *carrier* of Γ, and denoted by $\Delta(\Gamma)$.

A *cell complex* is a collection of cells of heterogeneous dimensions such that their interiors are disjoint, their union covers the carrier, and the boundary of each cell is made of cells of lower dimensions belonging to the complex itself:

Definition 2.15 *A* cell complex *(or* complex*, for short) in* $\mathbb{E}^d$ *is a finite set* Γ *of cells in* $\mathbb{E}^d$ *such that:*

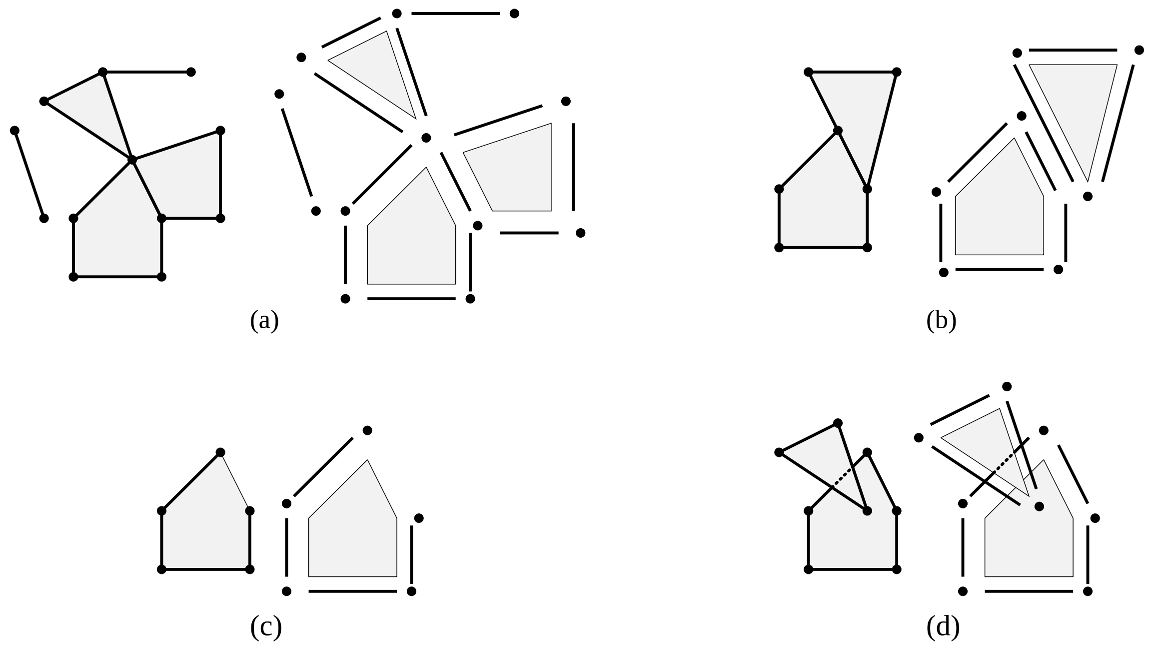


Figure 2.7: Examples of sets of cells which are and are not cell complexes, and related exploded views showing their cells of dimension 0, 1 and 2: (a) is a cell complex; (b) is not a complex because two edges intersect in their interiors (overlap), and one vertex is inside an edge; (c) is not a complex because one region edge is missing; (d) is not a complex because two regions intersect in their interiors (overlap), two pairs of edges intersect in their interiors (cross), and one vertex is inside a region.

1. *the collection of the interiors of all the cells in* Γ *forms a partition of the carrier of* Γ*;*
2. *for every cell* $\gamma \in \Gamma$*, the boundary of* γ *is the same as the union of the interiors of a subset of cells of* Γ*;*

If k is the maximum dimension of a cell in Γ, then Γ is a k-*dimensional* cell complex (or k-*complex*, for short).

Figure 2.7 shows an example of a two-dimensional complex and some examples of sets of cells that are not cell complexes; in particular, all sets (b,c,d) violate Condition 1, set (c) violates Condition 2 as well.

An alternative characterization of a cell complex is the one based on the following property:

Property 2.1 *A set* Γ *of cells in* $\mathbb{E}^d$ *is a* k*-dimensional cell complex if and only if, for every pair of distinct cells* γ *and* $\gamma' \in \Gamma$*:*

- *the interiors of* γ *and* γ' *are disjoint, and*

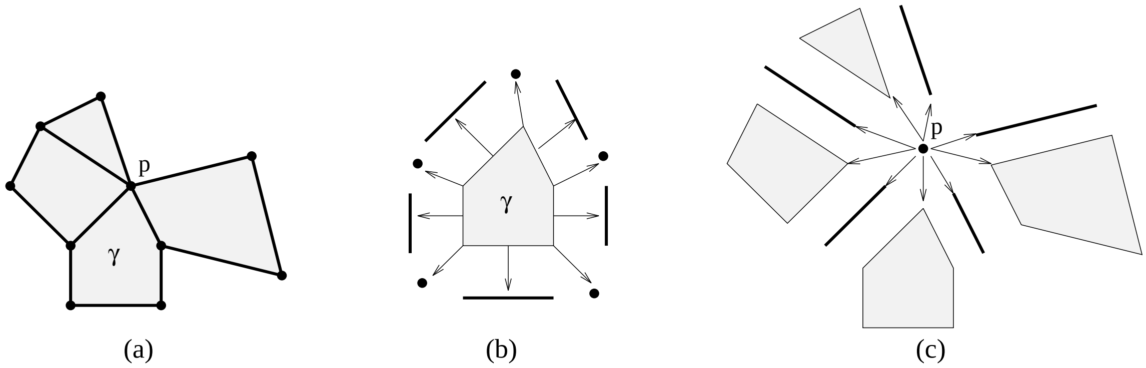


Figure 2.8: (a) A two-dimensional cell complex; (b) combinatorial boundary of γ; (c) star of vertex p.

- *the intersection of the boundaries of γ and γ' is the union of the interiors of a subset of cells of Γ.*

A proof that the conditions in Definition 2.15 and those in Property 2.1 are equivalent can be found, for instance, in [Mun84]. Intuitively, Property 2.1 means that the boundaries of any two neighboring cells in a complex are compatible.

For every $0 \leq s \leq k$, we denote with $\Gamma^{(s)}$ the set of s-dimensional cells in Γ. The zero-dimensional cells of a complex are called its *vertices*.

In a complex Γ, a cell γ' lying on the boundary of a cell γ is called a *face* of γ:

Definition 2.16 *Given a cell complex Γ and a cell $\gamma \in \Gamma$, the* faces *of γ are the cells $\gamma' \in \Gamma$ such that $\gamma' \subset \gamma$.*

The set of the faces of a cell γ is called the *combinatorial boundary* of γ in Γ. The set of cells γ', such that γ is a face of γ', is called the *combinatorial co-boundary* (or, *star*) of γ in Γ (see Figure 2.8).

2.3.1.1 Regular Cell Complexes

A cell γ in a complex Γ is called a *top cell* if there is no other cell $\gamma' \in \Gamma$ such that γ is a face of γ', i.e., the star of γ is empty. All the k-cells of a k-complex are top cells, but some lower-dimensional cells may be top cells as well.

A k-dimensional complex is *regular* if all its top cells are k-cells, i.e., any s-dimensional cell, with $s < k$ is a face of some k-cell:

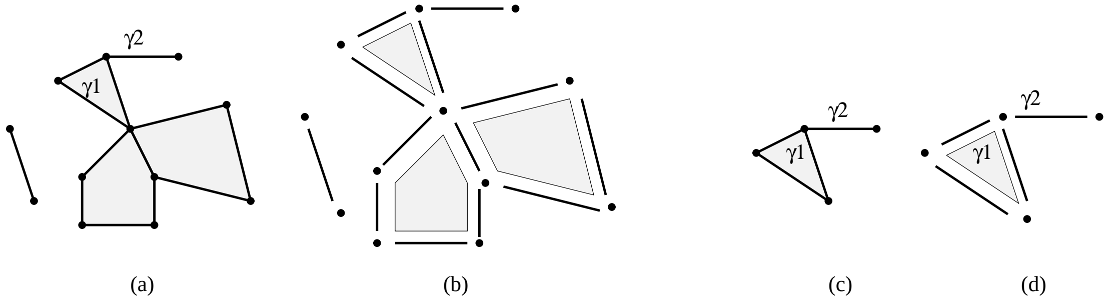


Figure 2.9: (a) A two-dimensional cell complex Γ, (b) its exploded view, (c) cell complex $\mathrm{sub}(\{\gamma_1, \gamma_2\}, \Gamma)$, and (d) related exploded view.

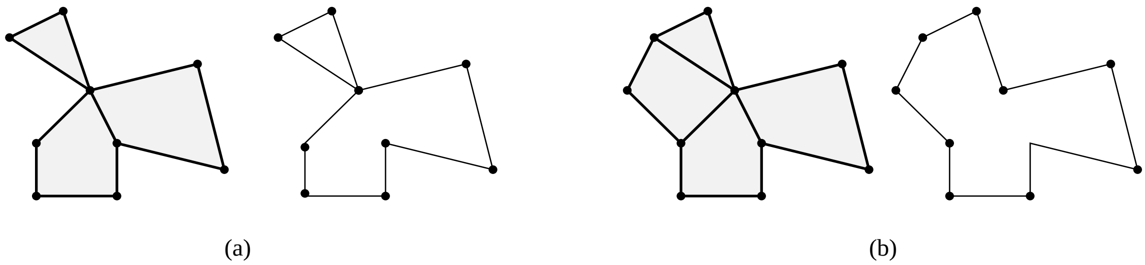


Figure 2.10: Two-dimensional regular cell complexes and their boundary complexes.

Definition 2.17 *A k-dimensional cell complex Γ is* regular *if, for every cell $\gamma \in \Gamma^{(s)}$, with $s < k$, the star of γ contains at least a k-cell.*

It can be easily seen that the carrier of a regular k-complex in $\mathbb{E}^d$ is a uniformly k-dimensional subset of $\mathbb{E}^d$. The two-dimensional cell complex of Figure 2.7 (a) is not regular; the one shown in Figure 2.8 is regular.

We define the *size* of a regular cell complex as the number of its top cells:

Definition 2.18 *Given a regular k-dimensional cell complex Γ, the* size *of Γ is the same as the cardinality of $\Gamma^{(k)}$. The size of Γ is denoted with* $\mathrm{size}(\Gamma)$.

Given a cell complex Γ, and a subset Γ' of its cells, we denote with $\mathrm{sub}(\Gamma', \Gamma)$ the cell complex formed by the cells of Γ' together with all their lower-dimensional faces in Γ (see Figure 2.9). It can be easily seen that, when Γ' is a set of k-cells, $\mathrm{sub}(\Gamma, \Gamma')$ is a regular k-dimensional cell complex.

The *boundary complex* of a regular k-dimensional complex Γ is the subset of cells of Γ that are contained (as point sets) in the boundary of the carrier of Γ. We denote the boundary complex of Γ with $\partial(\Gamma)$. Note that $\partial(\Gamma)$ is a cell complex on its own, of lower dimension than Γ. The boundary complex of a regular k-complex is a regular $(k-1)$-complex. Figure 2.10 shows some two-dimensional regular cell complexes and their boundary complexes.

2.3.1.2 Manifold Cell Complexes

The carrier of a regular k-complex is a k-manifold without / with boundary if the following conditions hold:

Property 2.2 *The carrier of a regular k-dimensional cell complex Γ is a k-dimensional* manifold without boundary *if an only if:*

- *every $(k-1)$-cell belongs to the boundary of* exactly two *k-cells;*
- *for every vertex p, the carrier of the star of p is homeomorphic to a closed k-dimensional ball.*

Property 2.3 *The carrier of a regular k-dimensional cell complex Γ is a k-dimensional* manifold with boundary *if an only if:*

- *every $(k-1)$-cell belongs to the boundary of* at most two *k-cells;*
- *for every vertex v, the carrier of the star of v is homeomorphic to either a closed k-dimensional ball, or to half a closed k-dimensional ball.*

Figure 2.11 shows some examples of manifold and non-manifold situations for $(k-1)$-cells and vertices of a k-dimensional complex, for $k = 2$.

We call a *manifold cell complex* any regular cell complex whose carrier is a manifold (with or without boundary).

The boundary complex of a manifold cell complex is a manifold cell complex. In Figure 2.10, the complex in (a) is not manifold, and its boundary complex is not manifold; the one in (b) is manifold, and its boundary complex is manifold too.

2.3.1.3 Topological Relations in a Cell Complexe

A cell complex is characterized by its *geometry*, describing the spatial location and extent of its cells, and by its *combinatorial properties*, which describe how cells are connected together. Combinatorial properties abstract the mutual spatial relations between the cells, disregarding the actual geometric shape and location in space.

For example, the combinatorial structure of a two-dimensional complex describes the way in which neighboring regions share their common edges and vertices, and is invariant with respect to small perturbations of the geometry (see Figure 2.12).

The combinatorial properties in a cell complex are described by *topological relations*:

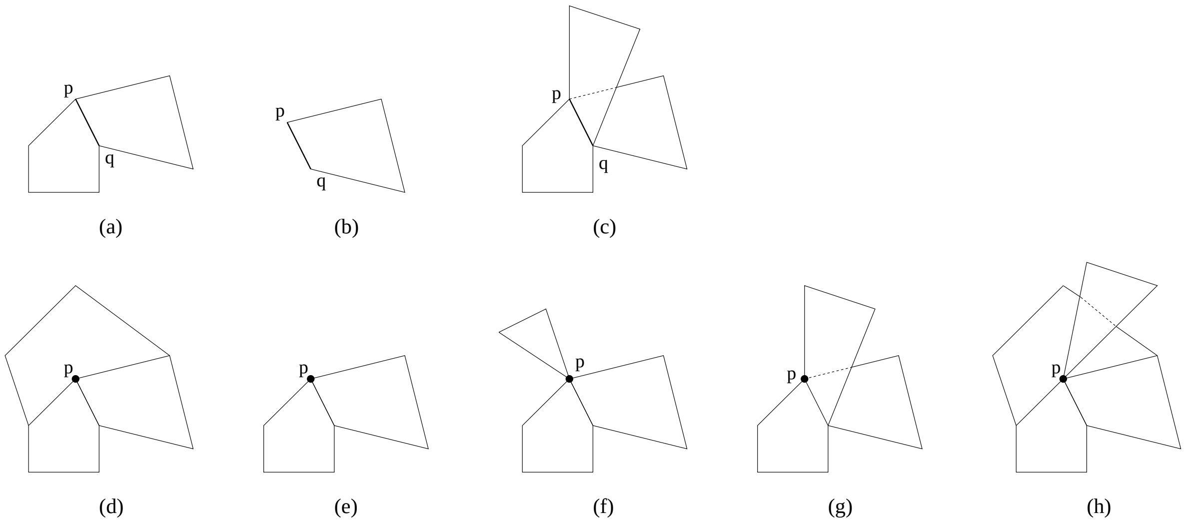


Figure 2.11: Manifold conditions for edges and vertices of a two-dimensional complex. Upper row refers to an edge pq: situation (a) is accepted; (b) is is accepted only in a manifold with boundary; (c) is not allowed since pq ha more than two incident regions. Lower row refers to a vertex p: situation (d) is accepted; (e) is is accepted only in a manifold with boundary; (f) is not allowed since the star of p is made of two half two-dimensional balls; (g) is not allowed since the star of p has a 'T' joint; (h) is not allowed since the star of p is made of a two-dimensional ball plus half another two-dimensional ball.

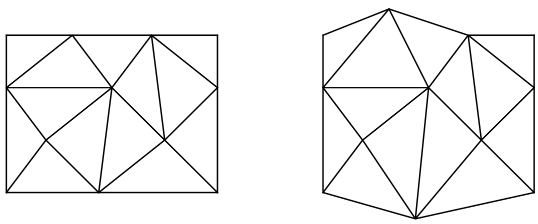


Figure 2.12: Two two-dimensional simplicial complexes in $\mathbb{E}^2$ with the same combinatorial structure and different geometry.

- *adjacency relations* are defined between cells of the same dimension that share a face: two k-cells are s-adjacent if they share a face of dimension s;
- *incidence relations* are defined between cells of different dimensions: *boundary* incidence relations link a cell to its faces; *co-boundary* incidence relations link a cell to the cells it is a face of.

In the reminder of this thesis, we will say that two k-cells are simply *adjacent* when they are $(k-1)$-adjacent.

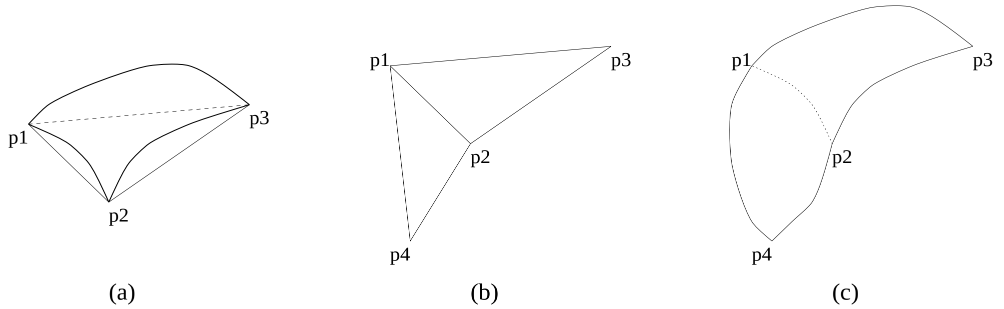


Figure 2.13: (a) A simplex and the graph of the function defined over it; (b) the C^0-continuous surface representation given by a pair of adjacent simplexes; p_1p_2 is a sharp edge; (c) the smooth surface representation given by the graphs of the functions defined over the two simplices.

2.3.2 Simplicial Complexes

A *k-dimensional simplex* in $\mathbb{E}^d$ $(0 \leq k \leq d)$ is the locus of points of $\mathbb{E}^d$ that can be expressed as the convex combination of $k+1$ affinely independent points:

Definition 2.19 *Given a set S of $k+1$ affinely independent points $p_0, p_1, \ldots, p_k$ in $\mathbb{E}^d$, the* k-dimensional simplex *(or* k-simplex, *for short), generated by S, is the set of points* $\sigma = \{a_0p_0 + a_1p_1 + \cdots + a_kp_k \mid \forall i = 0 \ldots k, 0 \leq a_i, \sum_{i=0}^{k} a_i = 1\}$.

For instance, 0-, 1-, 2- and 3-dimensional simplexes are points, segments, triangles, and tetrahedra, respectively.

A k-dimensional simplex is completely characterized by the set of points that generate it, called the *vertices* of the simplex. Every proper subset of the vertices of a simplex σ defines another, lower-dimensional simplex σ, contained in the boundary of σ. For example, any two vertices of a triangle generate one of its edges.

A k-simplex is also a k-cell. A *k-dimensional simplicial complex* is a cell complex where all top cells are simplexes, and every non-top cell is a simplex generated by a subset of vertices of some top cell.

Note that we have defined a simplex in a purely geometric fashion; according to Definition 2.19, a k-simplex is "flat", i.e., it is contained in a k-dimensional subspace of $\mathbb{E}^d$ (the one generated by its vertices). In algebraic topology, "curve" simplices are also considered, which are not captured by Definition 2.19. Here, we can realize a "curve k-simplex" from a k-simplex σ_i by defining a continuous function f_i over σ_i that maps σ_i into a k-dimensional surface patch, with the constraint that f_i is the identity function over the vertices of σ_i (see Figure 2.13 (a)). Within a simplicial complex Σ, we define a continuous (and, possibly, smooth) function f piecewise on the simplexes of Σ, with the property that f is the identity function on the 0-dimensional simplices (vertices), and, for every p-dimensional simplex $\sigma \in \Sigma$, with $p > 0$, the values of f on the boundary of σ are the same as the values of f over the lower-dimensional simplexes forming the combinatorial boundary of σ (see Figure 2.13 (b)). These conditions imply that the images of all simplexes of Σ through f define a surface homeomorphic to the carrier of Σ [2].

A *regular* k-dimensional simplicial complex is completely characterized by its k-cells since simplexes of lower dimensions are those generated by subsets of vertices of some k-simplex.

A two-dimensional regular simplicial complex is called a *triangle mesh*, or a *triangulation*. A three-dimensional regular simplicial complex is called a *tetrahedral mesh*, or a *tetrahedralization*.

The boundary complex of a k-dimensional simplicial complex is a $(k-1)$-dimensional simplicial complex. In particular, the boundary complex of a tetrahedralization is a triangulation.

For a regular k-dimensional simplicial complex in $\mathbb{E}^k$, the number of k-simplices is in $O(n^{\lceil\frac{k+1}{2}\rceil})$ in the worst case, where n is the number of vertices [Ede87]. For a plane triangulation, the number of triangles is at most $2n-2$, thus $O(n)$. For a tetrahedral mesh in three-dimensional space, the number of tetrahedra is $O(n^2)$ in the worst case.

In the applications, usually simplicial complexes, and triangle and tetrahedral meshes in particular, are used for defining a connecting structure on a set of scattered data, which may be points or lines.

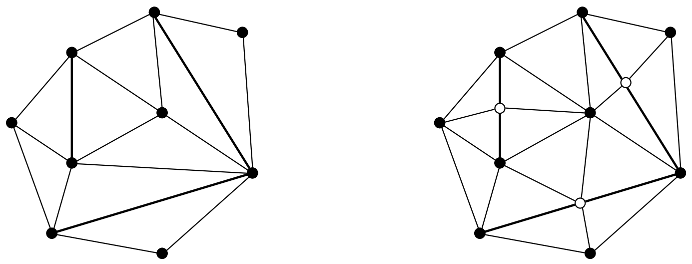
A *triangulation of a set S of points* in $\mathbb{E}^2$ is a triangulation having S as its set of vertices, and covering the convex hull of S. A tetrahedralization of a set of points in $\mathbb{E}^3$ is defined in a similar way.

Constrained and *conforming* triangulations allow enforcing the inclusion of a set of straight-line segments as edges of a triangulation [BE92]. In particular, they allow obtaining a triangulation covering an arbitrary, non-convex polygon instead of the convex hull of S. A constrained triangulation includes the given segments among its edges [LL86]; a conforming one inserts extra points on the segments, so that they get partitioned into chains of edges [Saa91, Ede93] (this operation is aimed at avoiding slivery triangles).

Definition 2.20 *Given a set V of points in $\mathbb{E}^2$, and a set L of straight-line segments, having their endpoints in V, and such that they do not intersect each other except at their endpoints, a* constrained triangulation *of V and L is a triangulation of V containing the segments of L among its edges.*

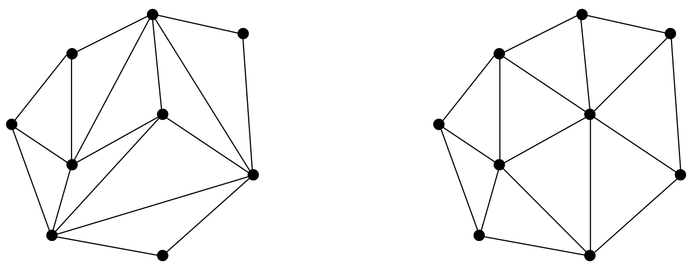
Definition 2.21 *Given a set V of points in $\mathbb{E}^2$, and a set L of straight-line segments, having their endpoints in V, and such that they do not intersect each other except at their endpoints, a* triangulation Σ *of a point set W* conforms *to V and L if $W \supseteq V$, and every segment in L is the union of a set of edges of Σ.*

[2] A decomposition of such surface into cells is also induced by Σ, and in fact such decomposition is a "simplicial" complex with "curve" simplexes.



(a) **(b)**

Figure 2.14: (a) A constrained triangulation, and (b) a conforming triangulation of the same set of vertices and segments. The given points are shown as black dots, constraint edges are shown as thick lines, extra vertices in (b) are shown as white dots.



(a) **(b)**

Figure 2.15: (a) A non-Delaunay triangulation, and (b) a Delaunay triangulation of the same set of vertices.

Figure 2.14 shows a constrained triangulation and a conforming one.

Delaunay simplicial complexes are a subclass of d-dimensional simplicial complexes in $\mathbb{E}^d$, that are widely used in the applications. Intuitively, a Delaunay simplicial complex is, among all simplicial complexes with the same vertex set, the one in which slivery simplices are avoided [For92, BE92]. Figure 2.15 compares an arbitrary triangulation and a Delaunay one.

Definition 2.22 *A* Delaunay simplicial complex Σ *is a regular d-dimensional simplicial complex in $\mathbb{E}^d$ satisfying the following property, called the* empty circle *property:*

> *for each d-simplex $\sigma \in \Sigma$, the d-sphere circumscribed to σ does not contain any vertex of Σ in its interior.*

Constrained and conforming Delaunay triangulations are used in two dimensions when it is necessary to include certain segments in a triangulation.

Definition 2.23 *A* Constrained Delaunay Triangulation *(CDT) of a point set V and a set L of straight-line segments is a constrained triangulation Σ of V and L, satisfying the following constrained version of the* empty circle property*:*

> *the circumcircle of each triangle $\sigma \in \Sigma$ does not contain in its interior any vertex which can be joined to all the three vertices of σ without intersecting the interior of an edge in L.*

Definition 2.24 *A* Conforming Delaunay triangulation *of a point set V and a set L of straight-line segments is a Delaunay triangulation of a set $W \supseteq V$ of points, which conforms to V and L.*

Bern and Eppstein [BE92] noticed that there does not seem to be a reasonable definition of constrained Delaunay simplicial complex in three or more dimensions; conforming Delaunay complexes are used in three dimensions in [BFLBM90].

2.3.3 Grids

Together with simplicial complexes, grids are another special class of complexes of great relevance in applications.

A k-dimensional cuboid is the locus of points that have $d-k$ of their coordinates equal to predefined values, and the other k belong to predefined closed intervals of $\mathbb{E}$:

Definition 2.25 *Given a sequence of d closed intervals $I_1, I_2, \ldots, I_d$ of $\mathbb{E}$, such that exactly $d-k$ of them are degenerate (i.e., contain just one point), the* k-dimensional axis-parallel cuboid *(or* axis-parallel k-cuboid*, for short) generated by $I_1, I_2, \ldots, I_d$ is the set of points $\{p = (x_1, x_2, \ldots, x_d) \in \mathbb{E}^d \mid \forall i = 1 \ldots d, x_i \in I_i\}$.*

For instance, 0-, 1-, 2-dimensional cuboids are points, segments, rectangles, respectively.

Let κ be a cuboid generated by a sequence $I_1, I_2, \ldots, I_d$ of intervals; every sequence $J_1, J_2, \ldots, J_d$, where, for all i, either $J_i = I_i$, or J_i is a degenerate interval containing just one of the two endpoints of I_i, defines a lower-dimensional cuboid, that is contained in the boundary of κ. We say that such lower-dimensional cuboids are *induced by* κ.

A *rotated k-dimensional cuboid* is the result of rotating a k-dimensional axis-parallel cuboid. In the reminder, we will use the term *cuboid* to denote axis-parallel as well as rotated cuboids.

A k-dimensional cuboid is also a k-cell. A *grid* is a cell complex where the carrier is a cuboid, all top-cells are cuboids, and every non-top cell is a cuboid induced by some top-cell (see Figure 2.16).

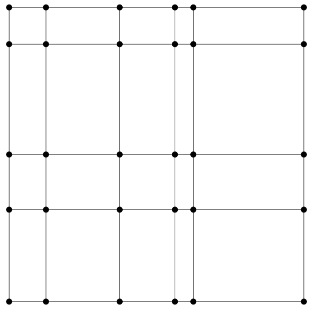
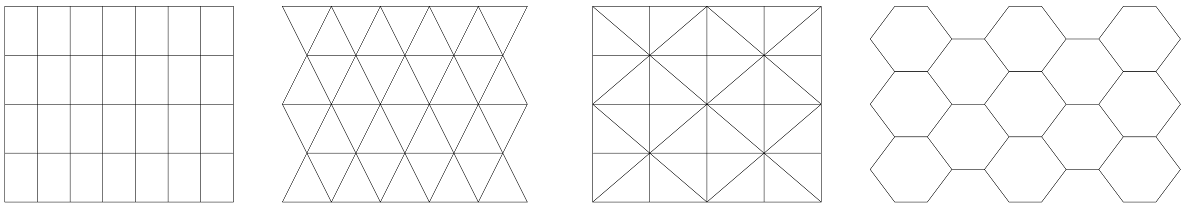


Figure 2.16: A two-dimensional grid.



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

Figure 2.17: Fixed-pattern two-dimensional cell complexes, based on the repetition of rectangular (a), triangular (b,c), and hexagonal cells (d).

A *square grid* and a *cubic grid* are a special case of a two-dimensional grid and of a three-dimensional grid, respectively, where all cells of the same dimension are congruent.

2.3.4 Fixed-Pattern Complexes

Fixed-pattern cell complexes are complexes where the same pattern of cells is repeated several times:

Definition 2.26 *A regular cell complex* Γ *in* $\mathbb{E}^d$ *is a* fixed-pattern complex *if all top-cells of* Γ *are congruent (i.e., they are a possibly rotated, translated and/or reversed copy of each other).*

Not all geometric shapes can be used for building fixed-pattern complexes [Sam90b]. In the two-dimensional case, fixed-pattern complexes can be made of squares, triangles, isosceles right triangles, and hexagons (see Figure 2.17). Note that the vertices of a fixed-pattern complexes must be regularly spaced in $\mathbb{E}^d$.

Fixed-pattern complexes based on simplexes can be defined in any dimension, and are special cases of simplicial complexes (triangle and tetrahedral meshes in two and three-dimensions, respectively). Fixed-pattern complexes based on cuboids can be defined in any dimension as well, and are special cases of grids.

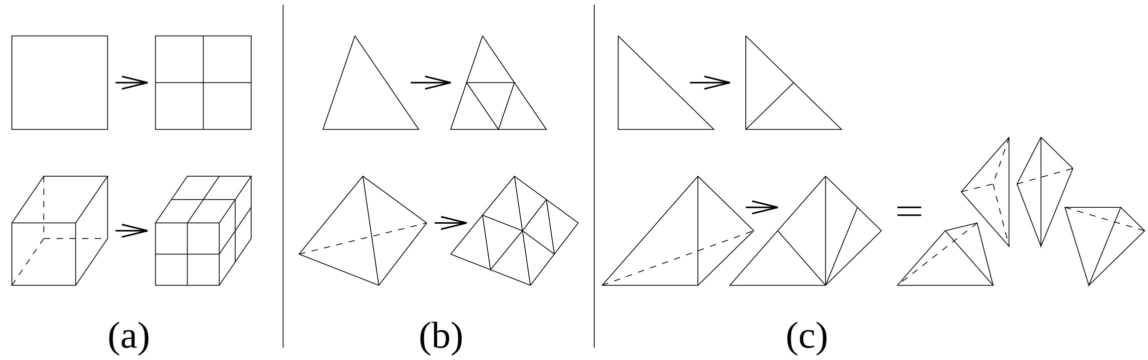


Figure 2.18: Some examples of self-similarity patterns in two and three dimensions: (a) based on cubic cells; (b) based on simplices; (c) based on right isosceles simplices (k-simplices where all the $(k-1)$-faces incident at the right angle are congruent).

2.3.5 Self-Similar Complexes

In some fixed-pattern complexes, it is possible to identify subsets of neighboring cells whose union has the same shape as a cell of the complex, scaled of a certain factor. In two-dimensions, this happens for complexes based on squares, triangles, and right triangles.

Definition 2.27 *A* pattern of self-similarity *consists of a cell* γ *and a complex* Γ*, such that* Γ *has* γ *as its carrier, and the top-cells of* γ *are all congruent to a scaled copy of* γ*.*

Some examples of self-similarity patterns are shown in Figure 2.18.

A self-similarity pattern (γ, Γ) is *admissible* for any cell γ' congruent with a scaled copy of γ. The application of a self-similarity pattern to γ' consists of replacing γ' with a copy Γ' of Γ, possibly translated, rotated, reversed and/or scaled, in such a way that the carrier of Γ' coincides with γ'. The same self-similarity pattern can be reapplied *simultaneously* to each cell of the complex obtained in this way, thus generating a possibly infinite sequence of fixed-pattern complexes at increasing resolutions [Sam90a, Sam90b].

A *self-similar cell complex* is a cell complex built by applying the same self-similarity pattern k times ($k > 1$), starting from a cell γ' for which the pattern is admissible.

2.3.6 Data Structures for Grids and for Simplicial Complexes

Data structures for d-dimensional grids are extremely simple and efficient. They reduce to storing the vertex having minimum coordinates on each axis, called the origin $o = (o_1, \ldots, o_d)$, plus the d positive reals $s_1, s_2, \ldots, s_d$ defining the width of the cells along each axis and the d positive integers $n_1, n_2, \ldots, n_d$ defining the number of cells along each axis. If the grid is *rotated*, the coefficients of the rotation matrix must be stored as well (see Figure 2.19).

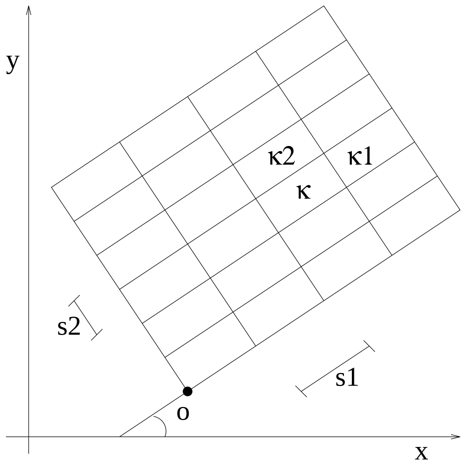


Figure 2.19: Encoding a two-dimensional grid. cell κ is identified by index pair $(2,2)$, its adjacent cells κ_1 and κ_2 are identified by pairs $(3,2)$ and $(2,3)$, respectively.

It is trivial to retrieve the geometry of each cell from such information. A d-cell is simply identified by d integer indices $r_1, \ldots, r_d$, where each r_i is in the range $0 \ldots n_i - 1$: the corresponding intervals defining the cuboid are $J_i = [o_i + r_i s_i, o_i + (r_i + 1)s_i]$, for $i = 1 \ldots d$.

Adjacency relations are also easily implemented by symbolic manipulation of indices: the cells $(d-1)$-adjacent to a given d-cell κ are those defined by a d-uple of indices obtained from that defining κ by adding or subtracting 1 from exactly one of the indices: let κ and κ' be two d-cuboids, identified by indices $r_1, \ldots, r_d$, and $r'_1, \ldots, r'_d$, respectively; κ' is $(d-1)$-adjacent to κ if and only if there exists exactly one index $1 \le j \le d$, such that $r'_j = r_j + 1$, or $r'_j = r_j - 1$, and for all other $1 \le i \le d$, $i \ne j$, $r'_j = r_j$.

Data structures for k-dimensional regular simplicial complexes in $\mathbb{E}^d$ encode the cells of dimension 0 (vertices) and k (top cells), and sometimes the relations of $(k-1)$-adjacency between k-cells. In k-dimensional regular simplicial complexes, the geometry of cells must be represented explicitly. Each vertex is described by d coordinates; each k-simplex is identified by its $k+1$ vertices.

A straight-forward structure, here called the *raw* data structure, represents each k-simplex as the list of the coordinates of its vertices. This requires $d(k+1)$ real numbers for each k-simplex.

The *indexed data structure* [Law77] avoids replicating the coordinates of vertices common to several k-simplices by storing vertex coordinates in an array, and, for each k-simplex σ, the $k+1$ indices corresponding to the positions of the vertices of σ in the array. The cost of the data structure is d reals for each vertex, and $k+1$ integers for each k-simplex, thus resulting much more efficient than the raw structure, especially for $d > 2$.

The indexed data structure can be enriched with adjacency information to represent, for each k-simplex σ, references to the $(k+1)$ k-simplices which are $(k-1)$-adjacent to σ [PBCF93].

For this purpose, k-simplexes are stored in an array, and, for each k-simplex σ, the indices of its adjacent simplexes in the array are stored. Some suitable convention rules the order in which adjacent simplexes are listed to the order in which the vertices of a simplex are stored, in order to provide also information about the $(k-1)$-face shared by a k-simplex with each adjacent one. For instance, the adjacent simplex at position i can be the one sharing all the vertices except the one of position i. This data structure requires $k+1$ more integers for each k-simplex than the indexed structure.

For instance, let us consider a triangulation of n vertices and $t \leq 2n$ triangles in $\mathbb{E}^2$, and assume that both integers and reals take the same space, here called "one unit". The raw data structure requires $18t \leq 36n$ units, the indexed structure $2n + 3t \leq 8n$ units, the enriched indexed structure $2n + 6t \leq 14n$ units.

2.3.7 Resolution of a Cell Complex

We introduce now an original notion of the *resolution* of a cell complex that we will use through this thesis. Intuitively, the resolution of a regular k-dimensional complex provides a measure of the *size and density* of its k-cells: the resolution is higher if cells are smaller, and thus more cells fit in the same unit of space. Sometimes the term *granularity* is used for resolution.

We assume to have a conventional criterion for measuring the "size" of a k-dimensional cell k: for instance, the size of γ may be defined as the k-dimensional volume of γ, or the radius of the smallest k-dimensional sphere containing γ, or the maximum distance between two points of γ. Whatever the convention adopted, the *resolution of a k-cell* γ can be defined as the inverse of the size of γ.

The *resolution of a regular k-dimensional cell complex* Γ is a function having the carrier of Γ as its domain, and defined piecewise on the cells of Γ:

Definition 2.28 *The* resolution *of a regular k-dimensional cell complex Γ is a function $\rho : \Gamma^{(k)} \longrightarrow \mathbb{R}$, where, for every k-cell $\gamma \in \Gamma$, $\rho(\gamma)$ is the inverse of the size of γ.*

For complexes made of congruent cells (for instance, fixed-patter complexes) resolution is *constant*, since all the cells have the same size. Generally speaking, a cell complex is said to have a *uniform resolution* if the difference in resolution between its cells is "sufficiently small", otherwise it is said to have a *variable resolution*. Note that uniform resolution is a qualitative concept, since the "tolerance", beyond which the resolutions of two cells is considered to be the same, depends on the specific context.

2.4 Approximation of Geometric Entities through Cell Complexes

A regular k-dimensional cell complex in $\mathbb{E}^d$ can be seen as a piecewise description of its carrier (which is a uniformly k-dimensional subset of $\mathbb{E}^d$): each cell describes a portion of the set, and their union describes the whole set.

2.4.1 Geometric Models based on Cell Complexes

In this thesis, we are interested in modeling k-manifolds in $\mathbb{E}^d$ and scalar or vector fields defined on k-manifolds in $\mathbb{E}^d$. *Discrete models* for manifolds in $\mathbb{E}^d$ (surfaces, solids, domains of scalar and vector fields) are provided by manifold cell complexes. In the reminder, we will consider cell complexes with the implicit assumption that they are *regular and manifold.*

2.4.1.1 Models for Solids and Surfaces

A k-dimensional manifold $\mathcal{O}$ in $\mathbb{E}^d$ (i.e., a surface or a solid, for $k < d$ and $k = d$, respectively) is represented throuth a manifold k-complex Γ in $\mathbb{E}^d$.

For instance, a manifold triangulation Σ in $\mathbb{E}^3$ represents a surface in space; a manifold tetrahedral mesh provides a volumetric representation of a solid object in space.

Complex Γ provides an approximate description of the *geometry* of $\mathcal{O}$; usually (but not always), a requirement is that the carrier of Γ is homeomorphic to $\mathcal{O}$, and, thus the topology of $\mathcal{O}$ is preserved. Sometimes, the topology of $\mathcal{O}$ may be approximated as well in Γ (e.g., by filling cavities, removing holes, breaking handles, etc.).

Simplicial complexes and grids are the types of complexes most commonly used in modeling. Since, for a surface, a simplicial complex Σ provides a piecewise approximation that is only C^0 continuous, sometimes a function f is defined piecewise over the simplexes of Σ: intuitively, the purpose of function f is "deforming" each cell of Γ in such a way to provide a better (usually, smoother) local representation of the surface. More precisely, function f maps each simplex of Σ into a "curve simplex", as explained in Section 2.3.2.

2.4.1.2 Models for Fields

A d-dimensional scalar or vector field $\phi : \Omega \subseteq \mathbb{E}^d \longrightarrow \mathbb{E}^{d+m}$ (where $m = 1$ for scalar fields, and $m > 1$ for vector fields) is approximated by providing:

- a d-dimensional cell complex Γ approximating the domain Ω of the field (which is a solid in $\mathbb{E}^d$), and

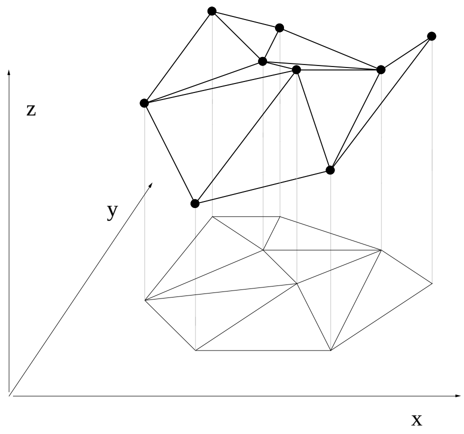


Figure 2.20: Approximation of a two-dimensional scalar field based on a triangulation of the domain and a piecewise linear function defined on it.

- a function f, piecewise defined over the cells of Γ, which determines an embedding of Γ in $\mathbb{E}^{d+m}$.

Function f is used to obtain an approximation of the field value at each point of the carrier of Γ. Figure 2.20 shows a cell complex approximating a two-dimensional scalar field.

Function f is continuous *at least* inside each cell. In general, f is not required to be continuous at the common boundary of adjacent cells. This permits to capture models of fields with jump discontinuities in their graph (e.g., images, which typically contain sharp color transitions). However, depending on the kind of objects to be represented, the continuity of function f across the whole carrier of Γ may be imposed as an additional constraint (e.g., for modeling a terrain).

If the domain Ω is discretized as a d-dimensional *simplicial* complex Γ, then function f can be defined over each simplex $\gamma \in \Gamma$ as a linear function interpolating the field values at the vertices of γ. On complexes that are not simplicial (like square grids), higher order interpolants are commonly used. Higher order interpolants are also used on simplicial complexes, when it is necessary to guarantee more than C^0 continuity in the resulting function. Such non-linear embeddings are usually adopted only in low dimensions (e.g., $d = 2$) due to their high computational cost. For an extensive treatment of this topic, see [NHM97].

2.4.1.3 Models for Explicit and Parametric Surfaces

An explicit k-dimensional surface in $\mathbb{E}^d$ is the graph of a continuous scalar field $\phi : \Omega \subseteq \mathbb{E}^k \longrightarrow \mathbb{E}$, and is modeled through a k-dimensional complex Γ approximating Ω plus a real-valued function f piecewise defined on the cells of Γ. For instance, a terrain is often modeled

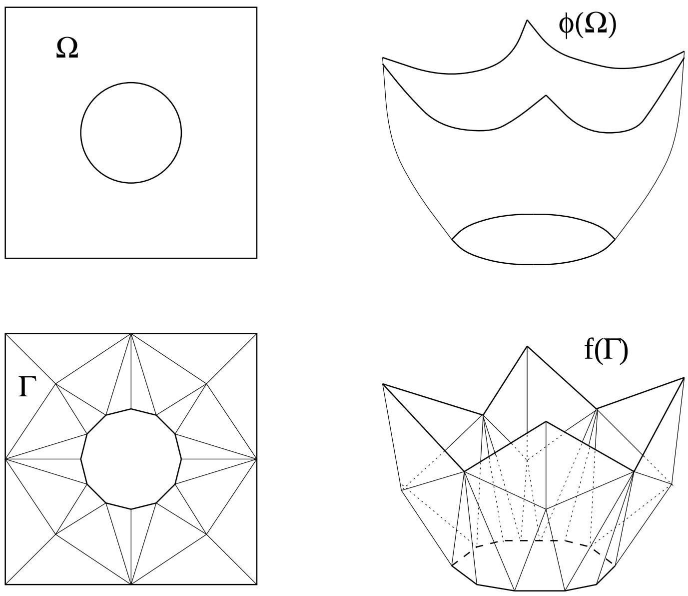


Figure 2.21: Approximation of a two-dimensional parametric patch based on a triangulation of the domain and a piecewise linear function defined on it.

through a triangulation in $\mathbb{E}^2$, which is lifted in $\mathbb{E}^3$ through a piecewise linear function adding a third coordinate to each vertex (see Figure 2.20).

As a special case of a vector field, a k-dimensional parametric patch in $\mathbb{E}^d$, image of a function $\phi : \Omega \subseteq \mathbb{E}^k \longrightarrow \mathbb{E}^d$ is modeled through a k-dimensional complex representing Ω plus a function f defined piecewise on the cells of Γ, and mapping each cell into d-dimensional space (function f provides a local approximation of function ϕ on each cell $\gamma \in \Gamma$). In this way, every k-cell $\gamma \in \Gamma$ (which is a cell in $\mathbb{E}^k$), maps through f onto a k-cell γ' in $\mathbb{E}^d$. The collection of those cells in $\mathbb{E}^d$ gives a representation of the parametric patch (see Figure 2.21).

2.4.1.4 Models for Surfaces over Surfaces

Similarly to d-dimensional fields, a field ϕ defined on a k-dimensional surface $\mathcal{O}$ of $\mathbb{E}^d$, and with values in $\mathbb{E}^m$, is modeled through a k-dimensional complex Γ approximating the surface $\mathcal{O}$, plus a function piecewise defined over the cells of Γ embedding Γ into $\mathbb{E}^{d+m}$. If the cells are simplices, a linear embedding is the one that maps every vertex of Γ into $\mathbb{E}^{d+m}$ through the value of function ϕ.

2.4.1.5 Remarks

A generic geometric model based on cell complexes consist of:

- a *regular and manifold k-dimensional cell complex* Γ in $\mathbb{E}^d$;
- possibly, a function f having $\Delta(\Gamma)$ as its domain and assuming values in $\mathbb{E}^m$, defined piecewise over the cells of Γ.

In applications, usually simplicial cell complexes or grids are used. The function f is present when describing fields, explicit or parametric surfaces (which reduce to special cases of fields), or when describing a surface with smooth, higher degree interpolants, as explained in the previous subsections.

In the reminder of this thesis, the geometric entity modeled through a cell complex Γ is called the *target object* of Γ.

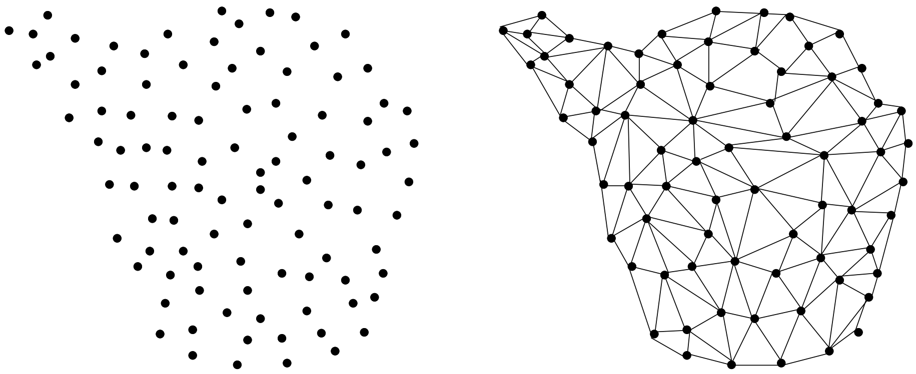
When approximating an object, usually the vertices of the complex (or their image/graph through function f, when f is present, and the target object is embedded in $\mathbb{E}^m/\mathbb{E}^{d+m}$) are placed at points belonging to the object itself, i.e., the cell complex (or its image/graph) *interpolates* a certain set of points of the object. Interpolating approximations are widely used, especially in the case of fields [GH95].

Typically, each top cell of the complex approximating a target object $\mathcal{O}$ has an associated approximation error related to $\mathcal{O}$. The way in which such errors are defined will be addressed in Section 2.4.3.

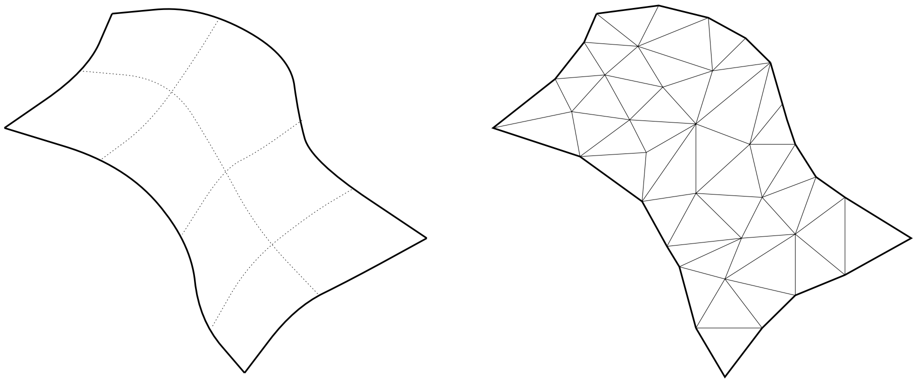
On the basis of the specific application, the cells of Γ may have additional properties associated, not necessarily of geometric nature, called *attributes*. For instance, in complexes representing surfaces in computer graphics, attributes may be local surface normals, material properties, texture coordinates, etc.; in complexes representing terrains, attributes may be soil type, vegetation, etc. In this thesis, we do not consider attributes associated with the cells of a complex. Moreover, we assume that functions are defined on the cells of a complex Γ only when Γ approximates the domain of a field.

2.4.2 Reconstruction and Discretization of Objects

Cell complexes are typically generated as discrete models of real objects reconstructed starting from experimental measurements, or obtained from the discretization of analytical descriptions of a synthetic object. The two situations are illustrated in Figure 2.22. In the first case, the aim is to provide some full description of an entity that is known only through samples; in the second case, the object is completely known in some global, continuous, analytical description, but some operation must be performed on the object, which is not tractable on the available representation. Thus, such a global representation is decomposed into simpler parts (i.e., the cells of a complex).



(a)



(b)

Figure 2.22: (a) Reconstruction of a shape from scattered data; (b) discretization of an analytical description.

2.4.2.1 Reconstruction Problems

In *reconstruction problems*, a finite set of data in the d-dimensional Euclidean space $\mathbb{E}^d$ is given, which provides *incomplete information* about an unknown spatial object. For instance, data may be points or lines belonging to an object or to the graph of a field. The aim is building a complete model for the unknown object, implementing the best estimate of the shape of the object, that can be recovered from the available data. An example is the reconstruction of a terrain from points measured on its surface.

In the literature, there exist two basic approaches to reconstruction [FN94]:

- *Discrete approach*: the reconstructed object is subdivided into pieces, an independent description is provided for each piece, and the overall shape of the object is obtained by glueing together the local descriptions.

Cell complexes fit in this approach, where each cell represents a piece of the original object. Referring to the example above, a terrain is approximated by means of a complex made of triangular cells, which provides a piecewise-linear surface, or by a complex made of square cells, which provide quadratic surface patches.

The shape description provided at each cell is *locally defined*: it depends on a small, local, *subset* of the whole data, and is typically simple (e.g., a polynomial of low degree). Referring to the example, locality means that changing an input value will change the description of the terrain only for a restricted number of cells near the modified point.

- *Continuous approach*: based on all data, a global description of the whole object is determined.

  In the example, the unknown terrain will be approximated by a unique analytical function, which minimizes the distance from the sampled data. See [FN94] for a survey on reconstruction methods for various kinds of data, which give continuous representations.

  The construction of a continuous representation reduces to solving systems of equations based on all data. Continuous representations usually involve analytical functions of high degree, and they provide a *global description*: the shape of the surface at a generic point depends on all the input data (thus, for instance, adding a new sample point may have effects on the whole reconstructed surface).

For their properties of *locality of description* and *low degree of polynomials involved*, discrete representations based on cell complexes are more suitable for reconstruction problems which involve huge amounts of experimental data. Many sampling devices also provide data which are already structured a cell complex. For instance, satellite images are square grids, medical data acquired through non-invasive inspection techniques, such as CT, MRI, PET, are cubic grids, laser range scanners produce dense meshes of triangles from the surface of the scanned object, etc. Moreover, the high order of polynomials involved in continuous representations, and their globality, make them less suitable to be manipulated algorithmically. Indeed, most operations on continuous representations require *discretization* as a preprocessing step. Finally, when the data set is too large, the systems of equations involved in the construction of continuous representations may become too costly to solve, and/or heavily ill-conditioned, thus reducing the applicability of continuous methods.

2.4.2.2 Discretization Problems

Discretization of analytical objects concerns the decomposition of a single, unstructured geometric object (e.g., a continuous representation) into an aggregate of several simple objects. Examples are found in computer graphics (where analytical surfaces such as spheres, cylinders or cones are decomposed into meshes of triangles for being rendered by graphics hardware), solid modeling (e.g., algorithms for the intersection of free form surfaces), and finite element analysis (where approximate solutions of an equation are searched by discretizing the domain into increasingly smaller elements) [HS95, Fie95].

A cell complex approximating an analytical object is usually built by sampling sufficiently many points on the given object, and then applying a reconstruction process. Unlike in reconstruction problems, here we can use available information about the shape of the object (such as gradients, normals, etc.) as constraints for driving the discretization. Also, the density of sampling can be adapted on-line: more points can be incrementally sampled from the object during the construction process itself, until the desired accuracy is achieved.

2.4.3 Approximation Error and Accuracy of a Cell Complex

A cell complex is used to model an object whose real shape is either unknown or too complicated to be described exactly. Thus, it is an approximated model.

In a cell complex that represents a geometric object, every cell γ has an associated *approximation error*, which provides an estimate of the distance of γ from the portion of the original entity it approximates. The approximation error is the result of a measurement made with respect to the initial data from which the complex has been built, and is usually recorded as a real number associated with each cell. In the applications, different criteria are used to evaluate the approximation error.

The *Hausdorff distance* can be defined between any two sets of points in $\mathbb{E}^d$; thus, we can apply it to a cell and the object portion it represents. Specialized distances are used for *scalar and vector fields*, which take into account the distance between the image of the same point through the original function and through its piecewise approximation provided by the cell complex.

When we have only incomplete knowledge about the real shape of the object approximated through a cell complex (e.g., in reconstruction problems), *discrete distances* are used, as we will explain later.

2.4.3.1 Error Evaluation Based on Hausdorff Distance

We denote with $d(p,q)$ the Euclidean distance between two points p and q in $\mathbb{E}^d$. The Hausdorff distance generalizes Euclidean distance to two generic sets P and Q in $\mathbb{E}^d$.

A first idea is defining the distance of a point p from a set Q as the smallest distance of p from a point of Q, and then define the distance from a set P to a set Q as the maximum distance of a point of p from Q, as follows:

Definition 2.29 *The* one-sided Hausdorff distance *of a set $P \subseteq \mathbb{E}^d$ from a set $Q \subseteq \mathbb{E}^d$ is*

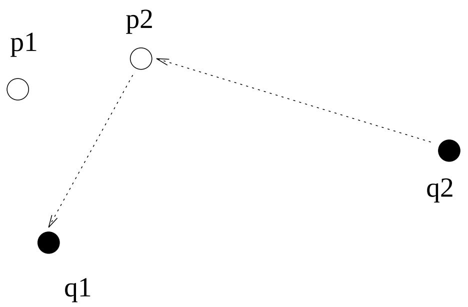


Figure 2.23: Asymmetry in the one-sided Hausdorff distances between two point sets $P = \{p_1, p_2\}$ (black) and $Q = \{q_1, q_2\}$ (white): $d_{H/2}(P, Q)$ occurs at points p_2 and q_1, $d_{H/2}(Q, P)$ occurs at points p_2 and q_2.

defined as:

$$d_{H/2}(P, Q) \equiv \sup_{p \in P} \{\inf \{d(p, q) \mid q \in Q\}\}.$$

Unfortunately, this approach leads to a distance function which is not symmetric: in general, $d_{H/2}(P, Q) \neq d_{H/2}(Q, P)$. A simple example illustrating this is shown in Figure 2.23.

The Hausdorff distance between two sets P and Q is defined as the maximum between the two one-sided distances:

Definition 2.30 *Let* $P, Q \subseteq \mathbb{E}^d$*; the* Hausdorff distance *between* P *and* Q *is*

$$d_H(P, Q) = \max\{d_{H/2}(P, Q), d_{H/2}(Q, P)\}.$$

It follows from Definition 2.30 that $d_H(P, Q) = d_H(Q, P)$, and $d_H(P, Q) = 0$ if and only if $P = Q$.

Let Γ be a regular cell complex approximating a manifold object $\mathcal{O}$. Intuitively, we want to define the approximation error of a top-cell $\gamma \in \Gamma$ as the Hausdorff distance between γ and the portion of $\mathcal{O}$ approximated by γ. In order to do that, we need some criterion for associating a subset $\mathcal{O}_\gamma \subseteq \mathcal{O}$ with each top-cell γ of Γ. In some cases, application-dependent knowledge may provide such set $\mathcal{O}_\gamma$ (we will see some examples in Chapter 8). In the absence of a-priori knowledge, we define set $\mathcal{O}_\gamma$ as the set containing:

- all the points $p \in \mathcal{O}$, such that the point $q \in \Delta(\Gamma)$ having minimum distance from p lies in γ;
- all the points $p \in \mathcal{O}$, such that p is the point of minimum distance from a point $q \in \gamma$.

In this way, set $\mathcal{O}_\gamma$ captures all points needed to evaluate both sides of the Hausdorff distance.

Definition 2.31 *Let $\mathcal{O}$ be a k-dimensional manifold, Γ be a k-dimensional cell complex approximating $\mathcal{O}$, and γ a k-cell of Γ. The* approximation error *of γ is $d_H(\gamma, \mathcal{O}_\gamma)$.*

In applications, an object may be known only at a finite set of points. In this case, a *discrete version* of the Hausdorff distance is used, in which we replace set $\mathcal{O}$ (which is unknown) with the set S of known points of $\mathcal{O}$. A subset S_γ can be associated with each top-cell γ of the approximating cell complex, defined in a totally similar way as subset $\mathcal{O}_\gamma$ above.

Definition 2.32 *Let $S = \{s_1, \ldots, s_n\}$ be a set of sampled points for a k-dimensional geometric object $\mathcal{O}$, where $s_i \in \mathcal{O}$ for all $i = 1, \ldots, n$. Let Γ be a k-dimensional cell complex approximating $\mathcal{O}$, and γ a k-cell of Γ. The* discrete approximation error *of γ based on S is $d_H(\gamma, S_\gamma)$.*

When a k-dimensional *surface* $\mathcal{O}$ in $\mathbb{E}^d$ is approximated through a k-dimensional cell complex Γ, then evaluating the approximation error of the cells of Γ with respect to $\mathcal{O}$ corresponds to the intuitive idea of measuring how close Γ is to $\mathcal{O}$.

When a *solid* $\mathcal{O}$ in $\mathbb{E}^d$ approximated through a d-dimensional cell complex Γ, we are usually interested in measuring how closely the carrier of Γ approximates the object; this is better formalized by measuring the error of the boundary complex of Γ in approximating the boundary surface of $\mathcal{O}$.

2.4.3.2 Error Evaluation for Fields

When modeling a scalar or vector field $\phi : \Omega \subseteq \mathbb{E}^d \longrightarrow \mathbb{E}^{d+m}$ through a cell complex Γ and a function f piecewise defined over Γ, an approximation error occurs at two levels:

- the distance between the domain Ω of the field and its approximation provided by the carrier of Γ (called *warping error*), and
- the distance between the field value and the value of f at each point (called *field error*).

In the general case, the Hausdorff distance is used to evaluate warping error. In many real cases, the domain Ω of the field and that of the complex Γ approximating Ω coincide, and thus there is no warping error. For instance, fields are often modeled inside a hypercubic domain, which is maintained as the domain of an approximation.

The following convention is used to evaluate the field error $\varepsilon(p)$ at each point $p \in \Omega$:

1. if p is inside the carrier of Γ, consider the distance between $\phi(p)$ and $f(p)$;

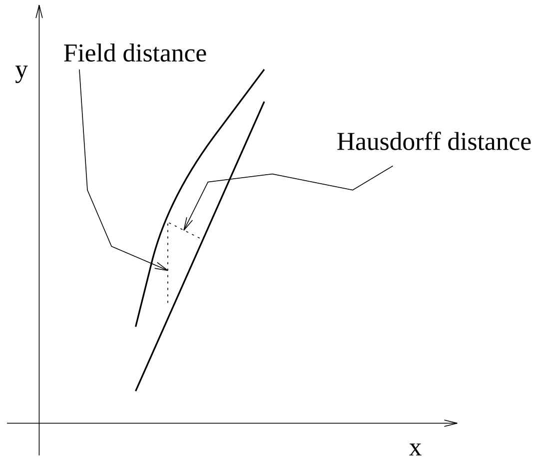


Figure 2.24: Field distance and Hausdorff distance for a one-dimensional scalar field: the field distance is constrained to be measured along a vertical line, and thus is larger than the Hausdorff distance.

2. otherwise, consider the point p' lying on the boundary of the carrier of Γ at minimum distance from p, and the distance between $\phi(p)$ and $f(p')$.

Note that Condition 2 only applies in case of a non-null warping error.

Given a field ϕ defined on a domain Ω and a cell complex Γ and related function f approximating it, intuitively we want to define the approximation error of each top-cell $\gamma \in \Gamma$ as the maximum over all points of Ω which are represented by γ. Thus, it is necessary to associate a subset $\Omega_\gamma \subseteq \Omega$ with each cell γ. If the warping error is null, then $\Omega \equiv \Delta(\Gamma)$, and we define Ω_γ as γ itself. Otherwise, in absence of application-dependent knowledge we define Ω_γ as done for $\mathcal{O}_\gamma$ in Section 2.4.3.1: here Ω plays the role of $\mathcal{O}$.

Definition 2.33 *Let $\phi : \Omega \longrightarrow \mathbb{E}^m$ be a d-dimensional field. Let Γ be a d-dimensional cell complex whose carrier approximates Ω, and f be the function defined piecewise on the cells of Γ to approximate ϕ. The* approximation error *of a cell $\gamma \in \Gamma^{(k)}$ is*

$$\varepsilon(\gamma) = \max\{\varepsilon(p) \mid p \in \Omega_\gamma\}$$

Note that, in general, measuring the approximation error according to the field distance (i.e., by applying Definition 2.33) and measuring the approximation error according to the Hausdorff distance (i.e., by applying Definition 2.31 to the graphs Γ and ϕ) do not give the same result. This happens because, for fields, we consider the distance between pairs of points corresponding to the same point in the domain of the field; the Hausdorff distance, on the contrary, considers the distance between a point and the nearest point to it. In practice, the field error over-estimates the Hausdorff distance (see Figure 2.24).

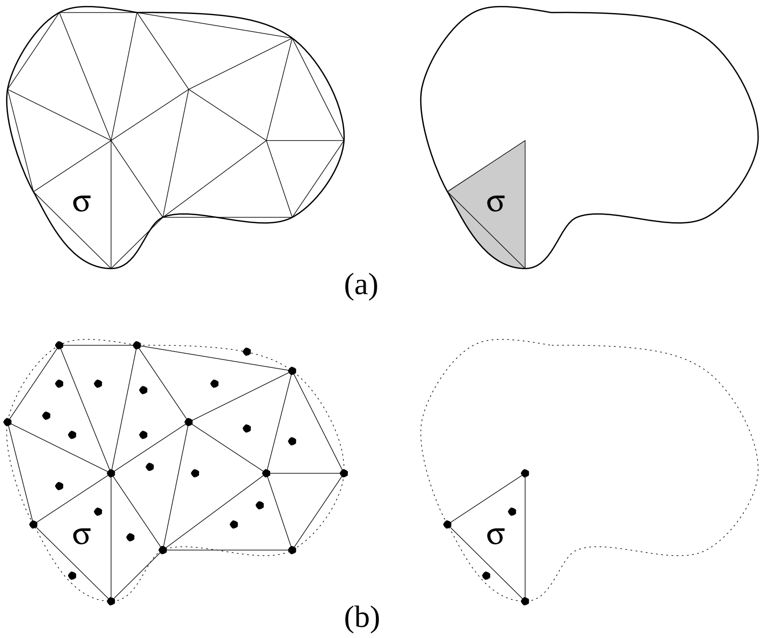


Figure 2.25: A triangulation approximating a two-dimensional object. (a) Estimate of the approximation error of a triangle σ with respect to the original object (the part of the object represented by σ is shaded on the right), and (b) with respect to a finite set of points belonging to the object (the points represented by σ are shown on the right).

In the applications, a field is typically known only at a finite set of points acquired through experiments. Thus, a *discrete version* of the distance is used, which measures both the warping and the field error just at the known points. Here, we give the expression for the field error:

Definition 2.34 *Let* $S = \{s_1, \ldots, s_n\}$ *be a set of sampled points for a* d*-dimensional field* $\phi : \Omega \longrightarrow \mathbb{E}^m$*, where* $s_i = (x_1, \ldots, x_d, y_1, \ldots, y_m)$*, with* $(y_1, \ldots, y_m) = \phi(x_1, \ldots, x_d)$*, for every* $i = 1, \ldots, n$*. Let* Γ *be a* d*-dimensional cell complex whose carrier approximates* Ω*,* f *be the function defined piecewise on the cells of* Γ *to approximate* ϕ*, and* γ *a* k*-cell of* Γ*. The* discrete approximation error *of* γ *based on* S *is:*

$$\varepsilon(\gamma) = \max\{\varepsilon(s_i) \mid s_i \in S_\gamma\}$$

where $S_\gamma \subseteq S$ *is defined as in the case of error evaluation based on Hausdorff distance.*

Figure 2.25 compares the discrete error evaluation and the continuous one.

2.4.3.3 Approximation Error, Accuracy, and Error Thresholds

Assuming that a criterion for determining the approximation error of a cell has been fixed, we define the *approximation error* of a regular k-dimensional cell complex Γ as a function

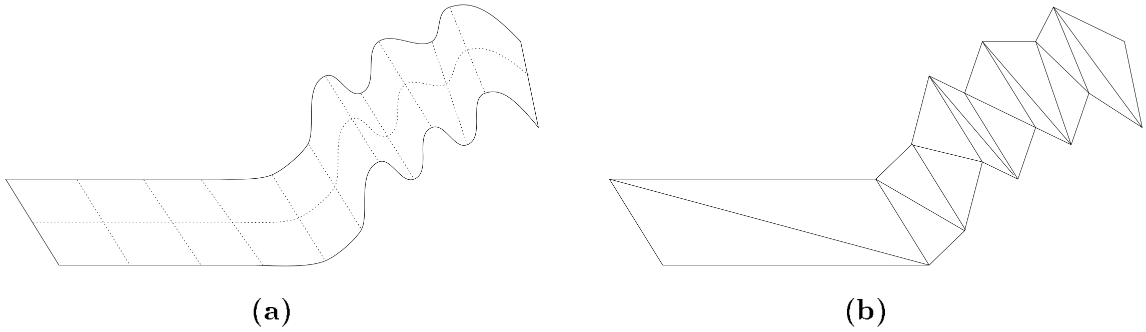


Figure 2.26: (a) A surface; (b) a triangle mesh approximating it at a uniform accuracy and a non-uniform resolution: in the flat area a lower resolution is sufficient.

$\varepsilon : \Gamma^{(k)} \longrightarrow [0, +\infty)$, where, for every $\gamma \in \Gamma^{(k)}$, $\varepsilon(\gamma)$ is the approximation error of γ.

Intuitively, the *accuracy* of a representation decreases as the approximation error increases, and vice versa. We define the *accuracy* of a regular k-dimensional cell complex Γ as a function $\alpha : \Gamma^{(k)} \longrightarrow (0, +\infty]$, where, for every $\gamma \in \Gamma^{(k)}$, $\alpha(\gamma) = \frac{1}{\varepsilon(\gamma)}$. Note that an infinite accuracy corresponds to a null error.

Generally speaking, a cell complex is said to represents a spatial entity at a *uniform accuracy* if the difference in accuracy between its cells is "sufficiently small" (depending on the context), otherwise it said to have a *variable accuracy.* Like uniform resolution, uniform accuracy is a qualitative concept, and does not admit a precise formalization.

Accuracy and resolution are two strictly related, but different concepts. Usually, accuracy increases with resolution: the description is more accurate when a larger number of cells is used to represent the same portion of an object. However, the resolution (i.e., the cell size) needed for achieving a given accuracy depends on the characteristics of the modeled shape. Note that a uniform accuracy does not necessarily imply a uniform resolution, since in rough areas of a shape a higher density is necessary to achieve the same accuracy for which, in flat areas, a low density is sufficient (see Figure 2.26). Conversely, a uniform resolution does not necessarily imply a uniform accuracy.

In applications, often it is needed that the cells of a complex satisfy some given accuracy requirements, which may be variable depending on the spatial location of cells.

We define an *error threshold function* in $\mathbb{E}^d$ as a function $\tau : \mathbb{E}^d \longrightarrow [0, +\infty)$ that defines, for each point p, the maximum approximation error tolerated for an object representation at p. We say that a cell complex Γ in $\mathbb{E}^d$ satisfies an error threshold function τ if, for each top-cell $\gamma \in \Gamma$, the approximation error of γ is not greater than the minimum of τ over γ:

Definition 2.35 *A k-dimensional cell complex Γ in $\mathbb{E}^d$* satisfies *an error threshold function τ if, for all $\gamma \in \Gamma^{(k)}$, $\varepsilon(\gamma) \leq \min\{\tau(p) \mid p \in \gamma\}$.*

2.4.4 Operations on Cell Complexes

A variety of operations are performed on cell complexes used to represent different kinds of data in different applications. Here, we provide a broad classification of such operations into operations for:

- *building* a complex;
- *editing* a complex;
- *querying* a complex;
- *analyzing* a complex.

The *construction of a cell complex* is a fundamental operation in every application. The formulation of the problem, and the algorithms used to solve it, are different in the various applications. They depend on whether we are reconstructing or discretizing an object, and, in each of the two cases, on the specific type of input (i.e., on the types of samples and on the type of analytical description available, respectively), on the type of complex to be built, on the conventions adopted for measuring the *approximation error*, the *desired properties* for the reconstructed model, etc. We will discuss such issues for a number of applications in Section 2.5.

Editing operations manipulate a cell complex after its construction. Examples of editing operations are: refining or coarsening a complex locally (e.g., through the introduction of a new data point in the case of a terrain), changing the shape of the cells while keeping their connecting structure consistent (e.g., moving a vertex for updating a wrong sample in a terrain, interactively deforming the shape of an object in a CAD system). Modalities and purposes are different in the various applications. A few editing operations will be presented in Section 2.5.

Basic query operations on a complex are concerned with retrieving information about the cells of a complex that satisfy certain properties. In particular, geometric queries are concerned with geometric properties of cells. Such operations get different semantics depending on the specific application, but their behavior with respect to the cell complex is the same. For instance, the basic query asking for the cell that contains a given point (called a *point location* query) is used in geographic applications for evaluating the height of a terrain at a certain location; in solid modeling it is used for testing whether a given point is internal or external to a given solid object.

In the case of complexes representing fields, a function f is piecewise defined over the cells of a complex Γ, thus embedding Γ into a higher-dimensional space. Therefore, it is possible to perform geometric queries in three different spaces:

- the space containing Γ;

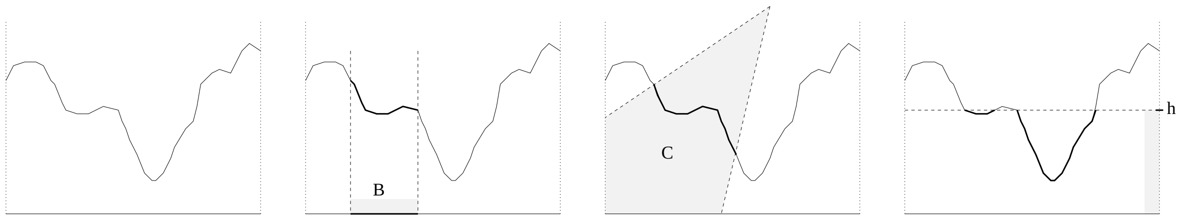


Figure 2.27: (a) A terrain (a one-dimensional section of the domain and of the graph are shown, obtained with a vertical plane); (b) query in the domain (retrieve all cells which project onto B); (c) query in the graph (retrieve all cells which lie in C); (d) query in the image (retrieve all cells with a field value below h).

- the space containing the graph of f over Γ;
- the space containing the image of Γ through f.

For instance, in the case of a terrain model, Γ is a two-dimensional complex in $\mathbb{E}^2$, and f is a real-valued function mapping Γ into $\mathbb{E}^3$. We can ask for the cells intersecting a given entity in $\mathbb{E}^2$ (e.g., to retrieve the terrain configuration along a river, or in a certain region, see Figure 2.27 (b)), in $\mathbb{E}^3$ (e.g., to retrieve the portion of the terrain surface inside a view frustum, see Figure 2.27 (c)), or in $\mathbb{E}$ (e.g., to retrieve the portion of the terrain surface lying above or below a certain height, see Figure 2.27 (d)).

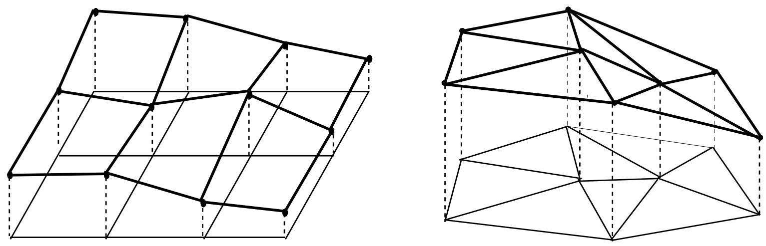
By *analysis operations*, we mean elaborated tasks that construct new information and/or new geometric entities from a given object represented as a cell complex. Such operations are specific to individual applications since they depend on the semantics of data and on the purpose of the application. Examples are the computation of the drainage network on a terrain, the collision test between two solid objects in motion planning, simulation of the evolution of a field on a spatial domain given its current state. The implementation of analysis operation needs query operations to extract information from a cell complex. Typical operations in a number of application fields will be illustrated in Section 2.5.

2.5 Applications of Cell Complexes

In this section, we describe a number of applications in which cell complexes are used as discrete models of geometric entities. These application fields contain the topics used as cases of study for the work developed in this thesis (these topics are listed in Section 2.7).

The applications we describe are:

- Modeling *terrains* in Geographic Information Systems (GISs);



(a) (b)

Figure 2.28: (a) A Regular Square Grid, and (b) a Triangulated Irregular Network.

- Modeling *solids* and *free-form surfaces* (e.g., the boundary of a solid) in CAD and virtual reality;
- Modeling *three-dimensional scalar fields* for analysis and visualization of scientific data.

For each application, we present the *kind of object* treated and of the *complexes* used to represent them, the available *input data* for building a complex, the *error metric* adopted, and the relevant *application-dependent operations*.

2.5.1 Terrains in GISs

A *terrain* is a two-dimensional scalar field with a special geographic semantics. The domain of the field is a part of the Earth surface (considered as plane), and the field value means elevation (for land) or depth (for sea).

A terrain model is based on a *two-dimensional complex* Γ in $\mathbb{E}^2$, covering the domain, plus a *scalar function* f defined piecewise on the cells of Γ, and embedding it into $\mathbb{E}^3$. Two basic types of complexes are used in terrain modeling: triangulations and square grids [DFPM99]. Terrain models based on triangulations are known as *Triangulated Irregular Networks* (TINs) for their capability of adapting to irregularly distributed data points; models based on grids are called *Regular Square Grids* (RSGs). The two approaches are shown in Figure 2.28. On TINs, function f is typically piecewise linear; on RSGs, it is either constant on each cell (giving a stepped terrain model) or a biquadratic patch. Higher-order functions may be used on triangular cells to obtain a smooth surface [Nie93, Nie97].

Data sources for terrains are field surveying, satellite and aerial images, digitization or scanning of existing maps. Data for a terrain are a set of points, and/or lines (e.g., isolines, ridges, valleys, rivers, etc.) at which the elevation of the terrain is known.

Some data sets (e.g., remote sensing images) are intrinsically structured as regular square grids, and thus trivially translate into a cell complex with square cells, or into a fixed-pattern triangulation obtained by cutting each square into two right triangles. However, their resolution is usually too high to allow an efficient processing of the model at full resolution. Typical grids have a resolution of 256 points per side, or more, thus yielding grids with tens of thousands of cells.

The resolution of a square grid can be trivially coarsened by subsampling the grid (e.g., deleting every second row and column and, possibly, adjusting heigth values at the remaining points). Approaches of this kind are inherited from image processing techniques, and are more suitable for smoothing a surface by removing high frequency details [Mal89, Dau92]. However, in a terrain it is recommended to keep vertices corresponding to significative terrain features, which are typically irregularly distributed on the domain. In other cases (e.g., data from on-site surveying and map digitizing), the known elevations correspond to *significative points and lines* on a terrain (e.g., peaks, pits, passes, ridges, valleys, rivers, coastlines, isolines, etc.), which must be preserved in the final model. Triangulations are used for their capability to include arbitrary points and lines among their vertices and edges, respectively.

The most commonly used *error metric* for terrains measures the approximation error of a cell as the maximum vertical distance between the cell and those data points whose vertical projection falls in the cell. Usually, terrains models are built for rectangular areas (like a map sheet), thus the domain of the terrain coincides with the carrier of the complex, and there is no warping error.

Examples of *simple queries* on a terrain are point location (used to evaluate the height at a certain point), line or region intersection (used, for instance, to evaluate the configuration of the terrain along a road, or in a certain area).

Terrain analysis in GISs includes a wide range of operations. A non-exhaustive list includes visibility analysis, computation of optimal paths, extraction of terrain features, computation of drainage networks, overlay with other geographic maps, simulation of complex processes such as erosion or pollution. All such operations require elaborated and time-consuming algorithms, and may benefit from the use of a variable-resolution model of the terrain [DFPM99].

Visibility problems are concerned with the computation of visible and invisible parts of objects placed on a terrain, or of the computation of visible parts of a terrain from a viewpoint, and with the use of such knowledge for solving optimization problems. Optimization problems related to visibility are concerned with placing observation points on a terrain according to certain requirements (e.g., placing a set of viewpoints that collectively can see the whole surface), line-of-sight communication (e.g., placing a sequence of mutually visible relay towers between two points), computation of paths with specified visibility characteristics (e.g., hidden paths, scenic paths). Applications include the location of fire towers, radar sites, radio, TV or telephone transmitters, path planning, navigation and orientation. See [Nag94, DFM99] for surveys on terrain visibility problems. Since visibility computations are sensitive to errors in elevations near the viewpoint, the solution of visibility problems can take advantage from representing a terrain at variable accuracy, higher near the viewpoint and lower in more distant parts [DFM97].

Paths can be defined on a terrain according to various optimality criteria, e.g., paths of minimum length, paths with minimum variation in height, with minimum slope along the path, paths avoiding or preferring certain types of soil (see [vK94, dBvK97, LMS97] for various characterizations of optimal paths). Multiresolution can help optimization tasks since it gives the possibility of reasoning on a terrain model at increasingly high accuracy, while progressively refining a rough candidate solution.

Topographic features are special points, lines and regions that have a specific meaning for describing the shape of the terrain: point features are *peaks*, *pits* and *passes*, lineal features are *ridges* and *valleys*, areal features are maximal regions on the terrain where the surface is *flat*, *convex* or *concave* (see [WLH85] for a classification of topographic features). Terrain drainage characteristics provide information on water resources, flood areas, erosion, and natural resource management. On a terrain water flows downstream according to gravity and collects into streams, which join and form rivers. The collection of all streams and rivers forms the *drainage network* of the terrain. The computation of a drainage network is strongly related to point and lineal features of the underlying terrain (see [YVS96, dBBD$^+$96]). Drainage network computation can benefit from representing a terrain with higher accuracy in proximity of certain types of features.

The overlay problem is a fundamental problem in GISs, and is not restricted to terrain analysis. It is common practice in GISs to organize spatial information into layers, each layer describing a certain feature. Terrain models correspond to the layer representing elevation; other layers may represent, for instance, land use, administrative subdivision, roads, etc., and are *maps* consisting of collections of points, lines and regions in $\mathbb{E}^2$ [3]. The overlay problem consists of superimposing two or more layers (with possibly a terrain model among them), in order to combine, at each spatial location, the information contained in all the layers. As a geometrical problem, overlaying a terrain with a map reduces to computing the intersections between the cells of the complex describing the domain of the terrain, and the collection of geometric entities describing the map. See [DFPM99] for a survey on overlay algorithms in GISs.

2.5.2 Objects in CAD and Virtual Reality

Solid objects considered in Virtual Reality (VR) applications are connected three-dimensional manifolds with boundary. There are two common approaches for their representation:

- *Boundary* representation schemes, which describe a three-dimensional solid through the two-dimensional surface bounding it.
- *Volumetric schemes*, which represent the *interior* of a solid.

[3] In GISs, the basic geometric elements of a map are more general than cells, and a map is not a cell complex (see [Ber97]). For instance, regions with holes (e.g., a parcel with a lake), points lying inside segments (e.g., a town on a road), points and segments lying inside a region (e.g., town and roads in a state) are common.

In the approach based on cell complexes, the boundary of a solid object (a two-dimensional surface) is usually described as a *triangle mesh embedded in* $\mathbb{E}^3$ [FP91]. The class of surfaces under consideration is sometimes enlarged to include surfaces with boundary, and non-manifold surfaces; this is done in order to encompass complexes that contain singularities due to errors in the data or in the acquisition process, and complexes resulting from boolean operations (which are not closed with respect to the class of manifolds). Volumetric representations consist of *tetrahedral complexes in* $\mathbb{E}^3$ [Boi84, Boi88, HDD+92, BBX95, BBCS96] or of hexahedral grids (voxel-based models).

In both approaches, available data for a given object may be the result of an acquisition process, or consist of a CAD model. In the first case, we deal with a *reconstruction* problem, and/or a *simplification* problem (since a model using the whole data is usually too large). In the second case, we deal with a *discretization* problem.

Operations for solid objects are visualization, typically with real-time requirements in an interactive environment, collision detection, finite element analysis, interactive editing of the object shape.

For visualization, a surface-based representation is used, since what needs to be displayed is the external surface of the object. In other applications, such as collision detection, or testing whether a point is internal or external to an object, a volumetric approach is more appropriate.

Generating meshes at variable resolution is crucial for real-time visualization and animation of large CAD models (e.g., aircrafts, ships, buildings) used in virtual reality. Heckbert and Garland [HG94] illustrate the motivations and benefits of multiresolution modeling for rendering applications. The possibility of reasoning at different levels of abstraction on the shape of an object helps operations such as collision detection (to quickly solve trivial cases on a coarse representation) and editing (to edit an intermediate resolution and then propagate the modifications to the representation at full resolution).

2.5.2.1 Acquired Objects

Data for reconstructing a solid object may come from several sources, and be in different formats. Scanning devices provide a set of points belonging to the object boundary. Such points are either scattered, or are connected through a very dense triangle mesh (e.g., data from laser range scanners [HDD+92, SZL92, BBCS96, CVM+96]). Stereo processes provide a set of segments lying on the object surface. Medical equipment (e.g., CT, MRI, PET, etc.) provides contours corresponding to cross sections of the object with a sequence of cutting planes. Except when a triangle mesh is given, the input is a set of unrelated entities in $\mathbb{E}^3$. However, when the data consists of contours, partial connectivity information is represented by the polylines connecting the points of each contour and by the sequential order of contours, and must be preserved in the reconstructed model.

Volumetric models of solid objects are usually reconstructed from scattered data: points [Boi84, HDD$^+$92, Vel93, BBX95, BBCS96], segments [BFLBM90], or contours [Boi88]. *Boundary models* can be derived either as a byproduct of volumetric ones [Boi88], or directly from the data. A very dense triangle mesh is already provided in the case of data from laser range scanners; due to its large size, such meshes are usually *simplified* (see Section 3.2.5 for a survey on mesh simplification algorithms).

2.5.2.2 CAD Surfaces

For CAD surfaces, the input is a boundary representation of a solid object which must be discretized into a mesh of triangles. Such boundary representations are usually given as parametric surfaces. Typically, each patch of the surface is analytically described as a Non-Uniform Rational Bi-Spline (NURBS) [PT96]; the trimming curves in parameter space may in turn be described by one-dimensional NURBS.

Note that a parametric surface, although defined as a collection of patches, has not the topological structure of a cell complex: the common parts of the boundaries of two touching patches may be decomposed in an incompatible way by the trimming curves of the two patches, thus forming T-joints (an example is shown in Figure 2.5). In fact, the trimming curves bounding each patch are defined independently of the others patches, without taking care of matching with the corresponding boundary portions of neighboring patches. Due to numerical inaccuracies, cracks, partially overlapping boundaries, and other inconsistencies are also quite common.

Converting a parametric surface into a triangulation is typically done for rendering or finite element purposes.

2.5.3 Volumetric Data for Scientific Visualization

Many application fields produce *volumetric data sets*, i.e., three-dimensional scalar or vector fields, that need to be understood and analyzed by researchers. Examples are medical data output by modern non-invasive inspection techniques such as CT, MRI, PET, data from fluid dynamics, geosciences, mechanics, and others.

The aim of scientific visualization is to provide tools for interactively displaying three-dimensional data sets with suitable techniques which allow the user to detect, enhance, recognize, and classify relevant properties of the data. Note that this is very different from realistic visualization of objects in virtual reality and CAD: here we do not reproduce the visual appearance of an object, but instead we produce an artificial image from data that are of non-visual nature.

Models of scalar fields used in medical applications often consist of dense, regular hypercubic grids, since they are obtained from data sampled at regular positions in space. Datasets from fluid dynamics, mechanics and earth sciences may not be regularly distributed. Most of them

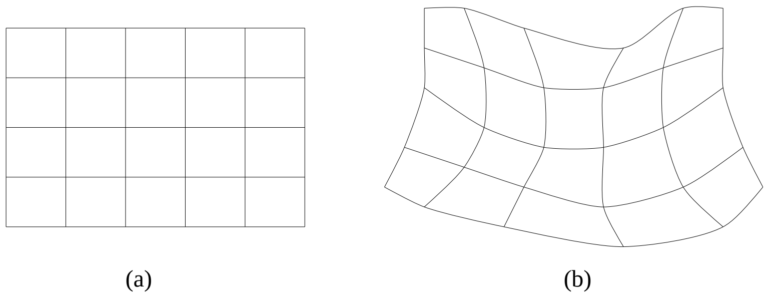


Figure 2.29: A curvilinear dataset: (a) the grid; (b) the deformed grid.

are distorted grids, i.e., cell complexes that have the same topology (i.e., connecting structure) of a grid, but deformed geometry (see Figure 2.29); these are called *curvilinear datasets*. In the most general case, data are scattered in an irregular, and possibly non-convex domain. In the first two cases, a grid seems the simplest and most suitable model; in the third case, a natural choice is a tetrahedralization.

Typical data sets are huge, in the range of 10^5 points or more. The need of reducing the size of such models by simplifying them has been pointed out by several authors [Kau90, Kau93, KLY92, Yag93]. As in the case of terrains, again we can either apply subsampling to a cubic grid, or we define a tetrahedral mesh on a reduced set of vertices, suitably selected.

Operations for volume data include visualization, Boolean operations, filtering operations, finite element analysis.

Three-dimensional scalar fields can be visualized with several techniques. *Indirect methods* visualize a part of the data at a time, e.g., a cross section (producing a two dimensional image where different colors correspond to different values of the field), or an isosurface (i.e., a two dimensional surface in $\mathbb{E}^3$ which is the locus of points where the field is equal to a certain value). *Direct methods* map the whole dataset onto the screen and make use of transparency to let the human viewer look through the three-dimensional volume. In both cases, since datasets are typically extremely dense, it is convenient to adapt their resolution (e.g., coarsening distant parts, or less important field values) before rendering. An alternative for direct methods is applying simplification not to the dataset, but to the part of it that is extracted for rendering (i.e., the cross section, or the isosurface).

In finite element analysis, for an efficient solution of an equation the density of the mesh must reflect the expected behavior of the solution, being denser where the solution changes more rapidly (see, e.g.,[Bri87]). A multiresolution model of the domain allows the extraction of different resolutions without computing them from scratch.

2.6 Mesh Compression

In this section, we describe a problem which is common to all applications, namely the *compression* of cell complexes representing geometric objects.

In order to accurately describe an object, cell complexes must be made of a large number of cells. When a very large complex must be rendered, stored (on secondary memory) or transmitted through a network, an explicit description of all the cells may waste storage space and cause delay in the communication. For instance, triangle meshes used in virtual reality may have about 10-100 thousands of triangles: an explicit encoding of the complex (e.g., through the indexed data structure introduced in Section 2.3.6) may have very large space requirements.

The problem of finding efficient encoding of a cell complex as a bitstream with small storage requirements is known as *geometry compression* problem. This research topic is connected with multiresolution modeling since it can be solved with similar techniques, as we will show in Section 3.4. In particular, compression techniques based on multiresolution are especially suitable for *progressive transmission* of cell complexes. For instance, let us consider a client/server framework where a server is in charge of storing and managing a large surface mesh, and clients are application tasks which can obtain (compressed) pieces of such mesh from the server. Multiresolution techniques allow adapting the resolution of the mesh that is sent from the server to a client according to the accuracy requirements of the client.

For transmitting a cell complex Γ over a network, the sender generates a bitstream from Γ by applying an encoding algorithm; the encoded version of Γ will reach its destination in short time since its size is small; then, the receiver recovers Γ by applying a decoding algorithm. Solutions to the mesh compression problem have impact not only in mesh transmission, but they also permit to save disk space, by storing the encoded version of a complex instead of the complex itself (in this case, the encoding / decoding algorithms are applied when writing to / reading from disk, respectively). Usually, encoded complexes do not support operations that are possible on "explicit" complexes; however, some encoding schemes have a format that can be directly processed by *rendering algorithms* [ESV96, Cho97, Dee95], and thus allow an application concerned just with visualization to save memory space, since decoding is no longer necessary.

The problem of *progressive transmission* is connected to that of geometric compression, but has an additional goal. Beside being compact, the bitstream generated for a cell complex must allow incremental decoding while it is being received. Each decoding step produces a complete representation of the object at a lower resolution, and, after the bistream has been received and decoded completely, the full resolution (i.e., the initial cell complex sent on the line) will have been reconstructed. In this way, the receiving application (e.g., a visualization tool) is able to start processing the lowest resolution while it is waiting for more accurate descriptions. Also, it can stop the communication as soon as a sufficient accuracy has been reached.

Multiresolution models can help progressive transmission since they are able to generate different levels of resolution for a cell complex. A number of existing methods for progressive transmission are based on the same ideas as multiresolution models [Hop96, DFMP98a]. Methods for mesh compression (and progressive transmission) will be reviewed in details in Section 3.4.

2.7 Case-Study Topics

The aim of this thesis is proposing a general framework for multiresolution cell complexes, which will be the object of the following chapters. In order to demonstrate the applicability of the work, we have selected a subset of topics from the applications just described, that we will use as cases of study: each topic correspond to one specific problem among the ones involved in the application containing it.

The selected application topics are:

1. Modeling *terrains* from scattered points;
2. Modeling *free-form surfaces* from discretization of CAD data;
3. Modeling *solid objects* reconstructed from scattered data;
4. Modeling *free-form surfaces* (e.g., the boundary of a solid object) from dense triangle meshes acquired through scanning devices;
5. Modeling *three-dimensional scalar fields* for scientific visualization;
6. Compression and *progressive transmission* of triangle meshes.

Topic 1 (terrain modeling) will be the main reference topic. Indeed, the field of terrain modeling presents most of the typical problems motivating the development and use of both multiresolution and mesh compression techniques. Another motivation for this choice is the previous experience of the author in the field of terrain modeling and analysis algorithms [DFM94b, BDFM95a, DFM94a, DFM95a, DFM95b, DFMPB96, DFM97, DFPM99, DFM99, DFMP98a].

2.8 Summary

In this chapter, we have introduced background notions that will be used throughout the thesis.

We have defined the classes of geometric entities for which we will develop our multiresolution model in Chapter 4: such entities are k-manifolds in $\mathbb{E}^d$, with $k < d$ (that we call surfaces), or $k = d$ (that we call solids), and scalar or vector fields defined on k-manifolds in $\mathbb{E}^d$ (surfaces or solids).

We have defined the basic components of a geometric model based on cell complexes: such a model is composed of a cell complex Γ, where each top-cell has an associated approximation error, and possibly a function piecewise defined on the top-cells of Γ. Since we represent manifold objects, we will consider regular and manifold cell complexes. We have defined the resolution of a cell complex and its accuracy in representing a given entity.

We have described a set of application fields of cell complexes and a number of interesting topics within them, which will be used as case-studies for applying the multiresolution framework proposed in this thesis.

Chapter 3

Previous and Related Work

3.1 Introduction

In order to accurately describe the shape of geometric objects, cell complexes with a high number of cells are needed. As a consequence, costs for processing a cell complex may become unaffordable for applications which handle very large objects, or have strong time constraints. On the other hand, we remarked that not all application tasks accessing a geometric entity require a high accuracy (and, thus, a high resolution) in *every part* of the complex representing it. The global size of the complex can be reduced by *locally adapting* the accuracy to the specific needs of each task: the various parts of the object can be represented at different resolutions based on their importance for the current task.

The ideal aim is to work, at each instant in time, with a cell complex whose approximation error satisfies some given bound, while its size is maintained as low as possible. In Chapter 1, we outlined two basic approaches to approximate this ideal situation, raising to the following two problems:

1. on-the-fly construction of a complex according to the current application requirements, and

2. off-line construction of a *multiresolution model*, from which several cell complexes at different resolutions can be extracted on-line with a reduced computational cost.

This thesis is focused on multiresolution models (i.e., on Problem 2). However, algorithms for building a multiresolution model usually rely on the same or similar techniques as the ones used to build approximated cell complexes; thus, Problem 1 is also of interest.

In particular, we will see that *iterative approaches*, i.e., methods that build an approximation at the required accuracy by iteratively refining / coarsening an initial complex at low / high resolution, are the basis for the construction of multiresolution models. We will consider the impact on a multiresolution model of the iterative technique adopted to build it in Section 7.5, showing experimental results in the case of terrains.

In Section 2.6, we have also introduced the problem of the *compression* of a cell complex, which is connected with multiresolution in two ways: firstly, both problems occur when dealing with large data sets, and, secondly, compression methods can be designed based on techniques and models developed for multiresolution.

This chapter reviews existing literature about the three problems mentioned above:

- In Section 3.2, we review methods for *building a cell complex* approximating a geometric object. A special emphasis is given to *iterative approaches*, for their connection with multiresolution modeling.
- In Section 3.3, we provide a survey of existing *multiresolution models for cell complexes.* Nested models which based on recursive space subdivisions are reviewed first, then the the attention is focused on models which trace the evolution of a complex through local modifications.
- In Section 3.4, we review existing methods for *compression* of cell complexes. We also present a new method based on shelling that we have developed.

3.2 Building an Approximated Cell Complex

Methods for the construction of an approximated cell complex heavily depend on the kind of geometric object to be modeled, and on the data available for that object. In Section 2.4.2, we mentioned the two major application problems that motivate the use of cell complexes: the definition of discrete approximations for continuous analytical objects, and the reconstruction of unknown objects from sampled data.

In the *discretization* case, the problem reduces to finding a suitable set of samplings on the given object for building a cell complex on them. This problem can be conceptually decomposed in two stages: first, the selection of the data, then the reconstruction. However, in practice the two stages are merged in iterative approaches that start from a complex built on an initial set of samples, and then improve it to fit the desired accuracy or size by iteratively generating more samples, or discarding some of the existing ones.

Discretization problems can thus be seen as special cases of reconstruction problems, where the available data set can be augmented at run time. This survey is mainly focused on reconstruction problems; some specific issues arising in the discretization of CAD data are presented in Section 3.2.5.

In the *reconstruction* case, we distinguish between two problems:

1. *Construction of a full-resolution cell complex*, using *all data* of a given set (for instance, a triangulation interpolating a set of points). In this case, the data-to-complex conversion does not introduce any approximation error with respect to the data. The resulting model is *approximated* with respect the object on which data have been sampled, but it is *exact* with respect to the input data.

2. *Construction of an approximated cell complex*, either by selecting a *subset* of the available data, or by transforming the given data set into a smaller one through operations such as, for instance, merging several data items into one. In this case, the resulting model is *approximated* with respect to the data as well.

Note that Problem 2 includes 1 as a special case, when no selection or transformation are performed on the input set. Also, Problem 1 becomes a subproblem of 2, if Problem 2 is addressed by decomposing it into two stages: the selection / transformation to create the reduced data set, and the construction of a complex using all the data of the reduced set.

The emphasis of this survey is on algorithms for Problem 2; algorithms for Problem 1 will be briefly considered since they may be subcomponents of algorithms for Problem 2.

When constructing an approximated cell complex, either a bound on the approximation error of the final representation or a bound on the size of the resulting complex is given. The ideal aim is either to produce the smallest possible number of cells within the given accuracy constraints, or the maximum possible accuracy within the given bound on the number of cells.

Both problems are likely to be NP-complete. A result by Agarwal and Suri [AS94] shows that the problem of deciding whether there exists a triangulation of size k that approximates a terrain within a predefined error tolerance, with respect to n sampled points, is NP-hard. It is a conjecture that similar negative results may hold for the other cases as well.

Provably good suboptimal algorithms exist only for the case of terrains [AD97] and convex polyhedra [BG95], but they are mainly of theoretical interest. In practice, heuristics are used.

Some heuristics are *iterative*, i.e., they start from an initial complex and incrementally modify it until it meets the given accuracy or size requirements. In particular, most iterative methods generate, in the intermediate stages of the process, a sequence of representations whose accuracy is monotonically increasing or decreasing: these methods are the most interesting for the purpose of this thesis, because the sequence of intermediate representations can be captured and encoded in a multiresolution model.

Interesting iterative methods can be classified into:

- *Bottom-up*, or *coarsening* methods, which start from the complex at full resolution and progressively decrease resolution by reducing the number of cells of the complex.

- *Top-down*, or *refinement* methods, which start from a coarse approximation and progressively improve resolution by increasing the number of cells.

In evaluating algorithms which build approximated cell complexes, the following issues are relevant: the *configurations of data* that they can accept (some methods are only suitable for data distributed according to a fixed pattern), the possibility of *controlling the approximation error* of the complex they produce (some methods simply build a complex of a certain size with no bound on its error).

The following review of construction methods follows a classification based on the type of geometric objects for which a complex is built, and gives a special attention to approaches based on iterative refinement or coarsening.

3.2.1 Two-Dimensional Fields

Input data for a two-dimensional field consist of a set of entities belonging to the graph of the field. Data may be points equally spaced along two orthogonal axes, implicitly defining a grid, or they may be points and lines in arbitrary positions (see Section 2.5.1). Models of two-dimensional fields are based on either grids or triangulations.

Approximated grids are obtained by subsampling gridded data sets (which are typically extremely dense); wavelet techniques, inherited from the theory of signal processing, fit here. Approximated triangulations are built by using a proper subset of either a gridded or of an arbitrarily distributed data set. Triangulations have the capability of fitting any given vertex set, and of including a given set of non-intersecting straight-lines segments among their edges.

These two alternatives give approximated models with different characteristics: wavelets tend to smooth a field by eliminating high frequency details, and are better suited for images, but not for terrains. Triangle meshes, on the contrary, allow maintaining characteristic details of a field, such as point and lineal features of a terrain.

3.2.1.1 The Wavelet Approach

Wavelets are a technique developed in the field of signal processing which allow separating the high-level shape of a signal from its details. Thus, they provide a mean to simplify a signal. The approach applies to signals in any dimension, in particular to two-dimensional signals, i.e., two-dimensional fields [Mal89, Dau92]. Wavelets are based on domain decomposition, and thus make use of cell complexes: they use *self-similar* cell complexes.

The application of wavelets works in two stages: signal *analysis* and signal *reconstruction* (sometimes called *synthesis*). The output of signal analysis is a decomposition of the signal into a coarse part and a sequence of correction terms (details) at increasing accuracy. In signal reconstruction, the details can be incrementally added to the coarse part to obtain an approximated representation of the signal at a certain (uniform) resolution.

Here, we provide just an intuitive description of the basic idea of wavelets. The wavelet approach involves non-trivial mathematical theory, which is out of the scope of this thesis. The interested reader is referred to [SDS96].

Signal analysis decomposes the domain into a sequence of *nested square grids*, which can be visualized as a sequence of levels. Starting from the finest level, the finest details are separated from the signal and encoded as differences (called wavelet coefficients) with respect to the rest of the signal; the signal without the details is then passed to the next coarser level and the process is repeated for each level up to the root. In the final situation, the square grid at the lowest resolution provides a coarse description of the signal by means of functions defined on its cells. Functions are associated with the cells of each finer grid, and represent details that must be combined with the representation of the signal at the next coarser level in order to refine it.

Signal reconstruction starts with the representation of the signal provided in the root and, for each subsequent level, combines the details found in the new level with the current signal approximation in order to build a new, more accurate one. The process is repeated for the next finer level, until the current approximation error becomes smaller than a given bound.

The wavelet framework has been extended to *fixed-pattern triangular subdivisions* in [Lou94, LDW97], for representing fields defined over spherical surfaces. In this case, the nested domain decomposition is based on the recursive subdivision of equilateral triangles into four parts.

Wavelet techniques automatically support multiresolution modeling (see also Section 3.2.3.7).

3.2.1.2 Building an Approximated Triangulation

Building an approximated model for a two-dimensional (scalar or vector) field reduces to building a triangulation Σ of its plane domain. Such a triangulation usually has its vertices at a subset of the data points. Then, a function approximating the field is defined locally on each triangle of Σ. In the case of terrains, a linear interpolant of the elevations measured at the three triangle vertices is commonly used; in the case of digital images, a constant or a polynomial fit to data spanned by the triangle can be adopted [Pup95].

In general, a set of points in the plane admits several triangulations. Optimality criteria for triangulations can be defined, which either take into account just the shape of triangles in the two-dimensional domain, or take into account the field values as well.

The most popular criterion is the *Delaunay* one (see Section 2.3), which belongs to the first type. Delaunay triangulations avoid slivery triangles (which cause numerical problems in processing) and minimize the "roughness" of the function defined over them [1]. Moreover, several efficient algorithms are available to compute a Delaunay triangulation. Since it works on the domain of the field, the Delaunay criterion applies to both scalar and vector fields. The second class

[1] Rippa [Rip90] formally defines the roughness of a triangle mesh as an energy function, and shows that the Delaunay triangulation minimizes it.

of optimality criteria gives so-called *data-dependent* triangulations. Several data-dependent criteria have been discussed and compared in [DLS90, Rip88].

Constrained and *conforming* (Delaunay or data-dependent) triangulations allow including a set of straight-line segments as edges of a triangulation. In particular, they are used to preserve the shape of the domain of the field, if this is not convex.

Algorithms that build a triangulation of a given set of points (and segments) solve Problem 1. A variety of algorithms exist for the construction of a Delaunay triangulation (surveys can be found in [For92, BE92]). Proposed approaches include optimization of an arbitrary triangulation [Law77], divide-and-conquer [LS80], sweep-line [For87], and incremental insertion of points while maintaining the Delaunay property at each step [GS85, Wat81, GKS92]. Dynamic algorithms that maintain a Delaunay triangulation after insertions and deletions of vertices also exist [KMS91]. Many algorithms are also available to build CDTs [LL86, BFLBM90, Che89, SW87, DFP92a].

Data dependent triangulations can be built through optimization of a generic triangulation (as in [Law77]), or through incremental insertion of points and re-optimization at each step (as in [GS85, Wat81, GKS92]); however, unlike in the Delaunay case, here the optimization process is not guaranteed to converge to a globally optimal triangulation.

For Problem 2 in the case of a two-dimensional field, usually *iterative* algorithms are used. Such algorithms process the data one at a time, and maintain a (locally) optimal triangulation at any stage. Iterative methods can be top-down or bottom-up, and adopt heuristics to decide which point to insert or remove at each step.

An initial triangulation for refinement methods can be obtained simply by considering a (possibly constrained or conforming) triangulation of the polygonal boundary of the domain of the field. Many authors include in the initial triangulation also selected feature points [FL79, Sou91].

Most heuristics select the next point for insertion in a greedy way, as the data point causing the maximum error over the current triangulation. The refinement methods by Fowler and Little [FL79], Park and Kim [PK95], and De Floriani et al. [DFFP85] use a Delaunay triangulation, Southard [Sou91] a data-dependent one. Garland and Heckbert [GH95] present and compare a number of variants of refinement methods for terrains.

An initial triangulation for coarsening methods is built on all data. Different heuristics are used to select vertices which must be deleted from the current triangulation.

A common heuristic is the symmetrical of the one used for point insertion: it deletes, at each iteration, the point that is likely to cause the least error increase; then, the triangulation is optimized according to the specific criterion. Lee [Lee89], and De Floriani et al. [DFMP97a] apply such an idea to Delaunay triangulations. Simplification techniques are more expensive than the corresponding refinement techniques, because update after deletion must be simulated for all current vertices in order to evaluate the corresponding cost in terms of error increase: such cost must be recomputed for a vertex each time any of its incident triangles is modified (as explained in Section 7.5).

De Floriani et al. [DFMP97a] describe, analyze and compare various heuristics for selecting vertices to be removed from a Delaunay triangulation, including error-driven and random criteria, selection of single vertices and of sets of independent vertices (through an approach inspired to that proposed by Kirkpatrick for the construction of a search structure over a planar triangulation [Kir83]). This work will be expanded in Section 7.5 in connection with the use of such methods for building multiresolution models.

3.2.2 Three-Dimensional Fields

The input data for the reconstruction of a three dimensional field typically consist of points sampled at the vertices of a cubic grid, or at the vertices of a warped grid (for curvilinear datasets), or scattered (see Section 2.5.3).

In the first two cases, data implicitly define a three-dimensional grid, but in practice approximated models of smaller sizes are used, which are obtained from the original grid either by subsampling it, or by constructing a tetrahedral mesh having its vertices at a subset of the data. The wavelet approach, described in Section 3.2.1, extends to three- and d-dimensional fields by using a subdivision of the domain into nested cuboids or nested regular simplexes [Mal89, Dau92, GLE97, NORS97].

In the case of scattered data sets, a tetrahedralization must be defined in order to impose a connecting structure to the data. The common choice is a Delaunay tetrahedralization, for reasons similar to those exposed in the two-dimensional case (see Section 3.2.1).

Algorithms for building a Delaunay tetrahedralization of a set of points in space exist which solve Problem 1 (surveys can be found in [BE92, Nie97]). Such algorithms include algorithms which incrementally insert points [Bow81, Wat81, Joe91, Raj91].

As far as Problem 2 is concerned, a non-iterative approach, proposed by Williams [Wil93], chooses a random subset of the vertices and computes a new Delaunay tetrahedralization. Such an approach can control the size of the resulting model, but not its accuracy.

Other authors propose *iterative* techniques, based on incremental refinement or coarsening, which try to select the smallest set for a certain error bound, and are an extension of those developed for two-dimensional fields. They are based either on incremental insertion of points in a current tetrahedralization, or on incremental deletion of vertices.

Hamman and Chen [HC94] propose a refinement method for convex domains. They iteratively select important points and insert them in an initial tetrahedralization covering the convex hull of the data set. Significant data are identified by large absolute curvatures obtained by a local least square approximation method. They adopt a data dependent triangulation that tries to minimize the local approximation error. Cignoni at al. [Cig97, CDFM+94] insert one point at a time in a Delaunay tetrahedralization. The next point to be inserted is selected as the one causing the maximum field or warping error in the current approximation.

Deletion of a vertex from a tetrahedralization is a difficult task, since, unlike the two-dimensional case, the polyhedral hole left from the removal of a vertex does not always admit a triangulation filling its interior [BE92]. Ruppert and Seidel [RS89] have shown that the problem of deciding whether a polyhedron can be tetrahedralized is NP-hard.

Proposed methods based on vertex deletion use heuristics: they choose a vertex, try to remove it, and, only if they succeed in building a tetrahedralization of the resulting hole, they actually remove the vertex.

Renze and Oliver [RO96] compute the Delaunay tetrahedralization of the vertices of the hole, and check if it contains a subset of tetrahedra fitting the hole: only if the answer is positive, they actually remove the vertex and update the mesh. This approach is very simple, but the test is likely to fail quite often, thus increasing the computational time. Moreover, vertices are dropped in a random order in [RO96], and the approximation error is not bounded.

A coarsening method for Delaunay tetrahedralizations is proposed in [Cig97]. The method iteratively discharges vertices as long as the current error is acceptable. The next vertex to be removed is selected as the one causing the smallest increase in field/warping error. Since an exact estimation of the error is expensive, the algorithm uses an approximated estimate based on field gradient. Since warping error changes only if a vertex on the boundary is removed, the algorithm precomputes warping errors for each boundary vertex p by considering the distance of p from the average plane that best fits the neighbors of p on the boundary triangulation. Once a candidate vertex v has been selected, the algorithm tries to remove it by collapsing one of the edges incident at v onto the other endpoint. The collapse must pass topological and geometrical tests: it must not change the genus or violate the 3-manifold condition, and it must not change the orientation of tetrahedra (i.e., flip them over). If the collapse is successful, then the vertex is removed and the tetrahedralization of the hole optimized to a Delaunay one.

Methods that simplify tetrahedralizations through edge collapses have also been proposed [PH97, GS98, THJW98], which extend similar methods for triangulations (see Section 3.2.3). Here, the size of a tetrahedral mesh is iteratively reduced by collapsing an edge into a point, and its adjacent tetrahedra accordingly. More details on these methods can be found in Section 3.2.3, where we describe the two dimensional approach that they extend.

3.2.3 Simplification of Triangulated Free-Form Surfaces

Given a triangulation at high resolution, representing a free-form surface, the problem of producing a triangulation that represents the same surface with a lower accuracy and a smaller number of triangles is known as *mesh simplification*.

A broad classification of mesh simplification algorithms can be made into:

1. methods aimed at simplifying the *geometry* of the surface, i.e., reducing the number of triangles in the mesh, while preserving its topological type [2];

2. methods aimed at simplifying the *topology* of the mesh, (e.g., closing holes, breaking handles, filling interior cavities, merging connected components, etc.);

3. *hybrid* methods, which may simplify both geometry and topology.

It is important to preserve the topological type of the input mesh in applications where meshes have to be processed to compute some properties of the objects that they represent; examples are biochemistry, finite element analysis, etc. In other cases, e.g., for realistic visualization in virtual reality environments, non-manifold conditions are tolerated as long as they cannot be perceived in the image.

Geometry simplification alone cannot simplify a mesh under a certain size without loosing the appearance of the object. Assuming for simplicity that every connected component of a scene is homeomorphic to a sphere, in a limit case of geometric simplification, every such component is coarsened into a mesh of four triangles (the surface of a tetrahedron), but this makes the scene unrecognizable. In this perspective, it is preferable to allow topology simplification by merging those connected components that are more distant from the observer. On the other hand, it does not make sense to perform topology simplification alone. Usually, either hybrid approaches simplifying both geometry and topology are used, or a topology simplification method is interleaved with a method for simplifying geometry. For instance, El-Sana and Varshney [ESV98] apply first topological simplification, followed by geometric simplification (through the algorithm of [CVM$^+$96]), and show that the resulting mesh is smaller in size than that obtained for the same approximation error by algorithm [CVM$^+$96] alone.

This thesis is concerned with *geometry simplification*, since we focus on cell complexes as models for representing the geometry of an object. Thus, in this thesis multiresolution of cell complexes is mainly multiresolution of geometry. Moreover, we have a special interest in iterative approaches.

The literature on mesh simplification is quite wide; existing mesh simplification algorithms have different characteristics, which sometimes depend on the application for which they have been designed:

- Some methods preserve the *topological type* of the input mesh [HH93, KT96, Tur92, SZL92, BBCS96, SL96, CCMS97, Hop96, AS96, Gué96, Ham94, CVM$^+$96] (in particular,

[2] For instance, its genus, the number of its connected components, or the number of closed curves forming its boundary (see [Ago76]).

manifold meshes are simplified into manifold meshes); other approaches do not preserve the topological type, and may produce non-manifold output for a manifold input [RB93, Sch97, GH97, GH98, PH97], or vice versa [HHK+95, HHVW96].

- Some methods can produce simplified meshes with a *bounded approximation error* [KT96, BBCS96, SL96, CCMS97, Sch97, CVM+96, Gué96, RB93, HHK+95, HHVW96]; other methods can bound just the size of the output mesh [Tur92, SZL92, Hop96, AS96, Ham94].

- Some methods pay a special attention in preserving features of the surface such as *sharp lines or corners* (e.g., so that a human viewer can easily recognize it) [HH93, KT96, SZL92, BBCS96, SL96, CCMS97, Hop96, AS96, Gué96, Ham94, CVM+96]; other methods tend to smooth the surface [HHK+95, HHVW96, Tur92].

In the following, we review mesh simplification methods. First, we provide a brief overview of non-iterative approaches, and then we describe iterative ones.

3.2.3.1 Non-Iterative Methods for Mesh Simplification

Some non-iterative methods rely on the construction of an auxiliary three-dimensional structure. The method by He and Hong [HHK+95, HHVW96] simplifies a mesh that describes the boundary of a solid. They first define a three-dimensional grid and assign a value equal to 0 or 1 to each cell, depending whether the center of the cell is outside or inside the object. Then, they compute a convolution of the values in the grid cells with a low-pass filter; the result is a grid where cell values are real numbers in the range $[0, 1]$. Finally, they get a simplified surface as the isosurface of value 0.5, which is extracted through a standard marching cube algorithm [LC87]. This method can control the error (by choosing the resolution of the grid), but not the size of the resulting mesh; it simplifies topology, and always produces a manifold surface. In practice, the effect is smoothing the surface by removing high-frequency details. Andujar et al. [AAB+96] use a similar approach, based on an octree instead of a grid.

Other direct methods for mesh simplification are based on merging groups of nearby triangles or vertices. The *coplanar face merging* by Hinker and Hansen [HH93], and similar methods [KT96, HZ91], detect adjacent triangles that are coplanar or nearly coplanar (found by comparing their normals), merge them, and retriangulate the resulting polygons. Such approach preserves surface topology, and features such as sharp lines, but it cannot guarantee a bounded approximation error. The *superfaces* method by Kalvin and Taylor [KT96] chooses a seed triangle and merges it with adjacent triangles whose vertices are within a given distance from the average plane of the vertices of the initial face, and have a similar orientation; this approach guarantees a bounded error.

The *vertex clustering* method by Rossignac and Borrel [RB93] identifies clusters of nearby vertices, replaces each cluster with a single vertex, and collapses their incident triangles accordingly. Vertices to be unified are found by using a grid superimposed on the triangle mesh:

vertices lying in the same grid cell form a cluster. Later work [LT97, Red96] elaborates on the same basic idea. The approximation error introduced is bounded by the resolution of the grid. In fact, the distance between old and new vertices cannot exceed the cell diagonal. On the contrary, it is not possible to bound the number of vertices in the output mesh, if not a-posteriori. The method is not concerned with maintaining topology, or surface features. This method is born as a direct method, but it can be turned into an iterative one by clustering with increasingly finer grids (such variant is sketched in [LE97]).

The *Re-Tiling* method by Turk [Tur92] is another non-iterative method, based on a different idea. A new set of vertices is distributed randomly on the original triangle mesh, and then moved to optimal locations based on surface curvature. Finally, a new mesh is built based on such vertices only. The approach preserves the topology and the overall shape of the mesh, but not its sharp features; it is especially suitable for smooth surfaces. The algorithm builds a triangulation with a specified number of vertices, and provides no control over the approximation error.

3.2.3.2 Classification of Iterative Methods

Iterative methods start from the given mesh and iteratively transform it through local modifications until some ending condition occurs. Based on the type of local modifications applied, the proposed methods can be classified into:

- methods which *iteratively remove a vertex* and retriangulate the resulting hole [SZL92, CVM+96, CCMS97, BBCS96];
- methods which *iteratively collapse a higher-dimensional entity* into a vertex; such methods are further classified on the basis of the type of entities allowed to collapse: edges [Hop96, Gué96] or triangles [Ham94];
- methods which *iteratively collapse a group of* (not necessarily adjacent) *vertices* into a single vertex [GH97].

While the above approaches monotonically reduce the size of a mesh, the *Mesh Optimization* method proposed by Hoppe et al., [HDD+93], tries to minimize an energy function which takes into account both the size and the error of a mesh, and allows size-increasing modifications as well. This approach lately evolved into an edge-collapse method [Hop96], and thus will be reviewed in the second class.

A remark needs to be made about the way in which iterative simplification methods estimate the approximation error of the current mesh. In fact, some approaches [SZL92, AS96] take into account just the approximation error introduced at each step of the simplification (i.e., the distance of the current mesh from the *previous mesh*), instead of the approximation error of the current mesh with respect to the *initial mesh* at full resolution. The first kind of

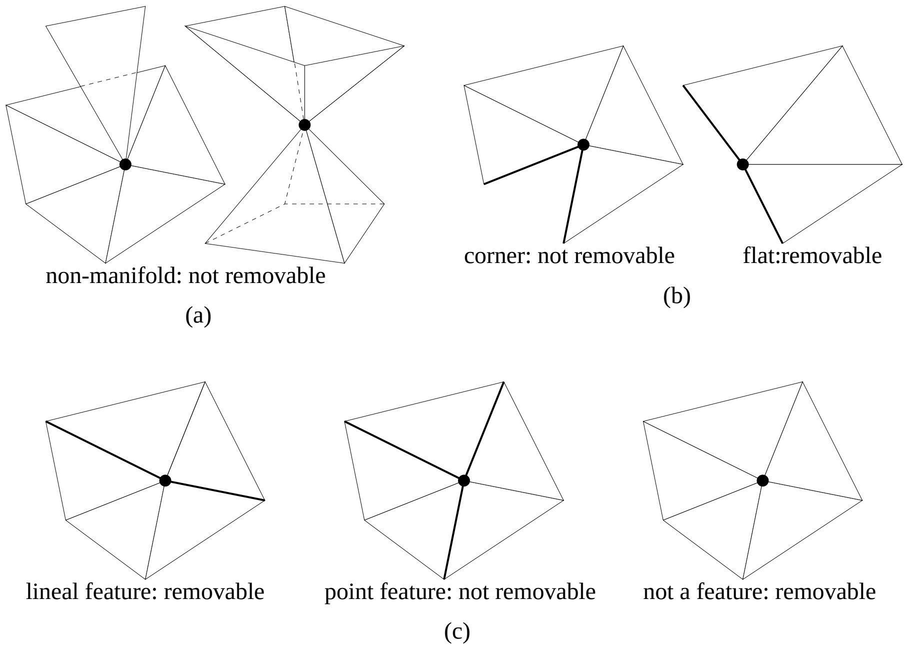


Figure 3.1: Criteria for removing vertices in [SZL92]: (a) a non-manifold vertex is not removable; (b) a boundary vertex is removable only if the angle between the two border edges is nearly flat; (c) an internal vertex is removable only if it is not a surface feature.

estimate is called *local error*, the second one *global error*. Only methods that control the global error [BBCS96, SL96, CCMS97, CVM+96, GH97, GH98] are able to guarantee a bound on the approximation error of their final triangle mesh. Some methods, finally, do not take into account the approximation error explicitly [LT98], or combine it in a weighted way with other parameters related to the "quality" of the current approximation [Hop96, RB93, Ham94].

3.2.3.3 Iterative Methods Based on Vertex Removal

Historically, the first algorithm based on iterative vertex removal is the *Mesh Decimation* developed by Schroeder et al. [SZL92]. They classify vertices as removable or not removable, based on their local topology and geometry, in order to preserve mesh topology and sharp features:

1. if a non-manifold condition occurs at a vertex v (as in Figure 3.1 (a)), then v is not removable, since removing it would change the mesh topology;
2. if v has half a fan of incident triangles (i.e., v lies on the boundary of the surface, as in Figure 3.1 (b)), then:

a. if the angle between the two boundary edges incident at v is smaller than a tolerance (i.e., v defines a corner), then v is not removable;

b. otherwise, v is removable;

3. if v has one fan of incident triangles (i.e., v lies in the interior of the surface, as in Figure 3.1 (c)), then:

   a. if two sharp edges are incident at v (i.e., v lies in the interior of a sharp line), then v is removable, but the new triangulation without v must contain the edge joining the other endpoints of the two sharp edges;

   b. if more than two sharp edges are incident at v (i.e., v is a sharp point), then v is not removable, since it is a surface feature;

   c. otherwise, v is removable.

The decision whether to remove a given removable vertex is based on an approximate estimation of its distance from the new mesh that would be obtained from its removal. An interior vertex v which is not a feature (case 3c) is removed if the distance between v and the average plane of its neighbors is smaller than a tolerance; a boundary vertex v (case 2a) is removed if the distance between v and the new boundary edge is smaller than a tolerance. A vertex v lying on a sharp line (case 3a) is removed if the distance between v and the segment joining the two other vertices of the sharp edges incident at v is smaller than a tolerance. This approach can bound just the local error introduced in each vertex removal with respect to the current mesh, not to the original one. Thus, it does not guarantee a bounded global error for the final approximation.

Several later variants of mesh decimation estimate and bound the *global approximation error* by using different techniques. Since a precise computation of the distance of the current mesh from the input one is too expensive, approximate estimations are used. A pessimistic estimate of the global error is obtained by accumulating the local errors introduced in each decimation step [CCMS97]; Soucy and Laurendeau [SL96] consider just the distance of the original vertices from the current mesh; Ciampalini et al. [CCMS97] consider the distance of the edges of the original surface from the current one, by sampling points on the edges. Bajaj et al. [BBCS96, BS96] improve the vertex decimation strategy by introducing a model for global error propagation, which allows estimating error both for the mesh, and for all its scalar attributes (e.g., normals, colors).

In [Sch97], Schroeder proposes a variant of mesh decimation that allows simplifying the topology. He introduces an additional operator, called *vertex split* [3], which replaces a vertex p with two congruent copies of p, and divides the faces incident in p between the two copies. This is typically used to "separate" the two fans of triangles of a non-manifold vertex in order to allow further simplification (see Figure 3.2). This algorithm removes a vertex by collapsing it

[3] This operator must not be confused with the *vertex split* used in Progressive Meshes (see Section 3.3)

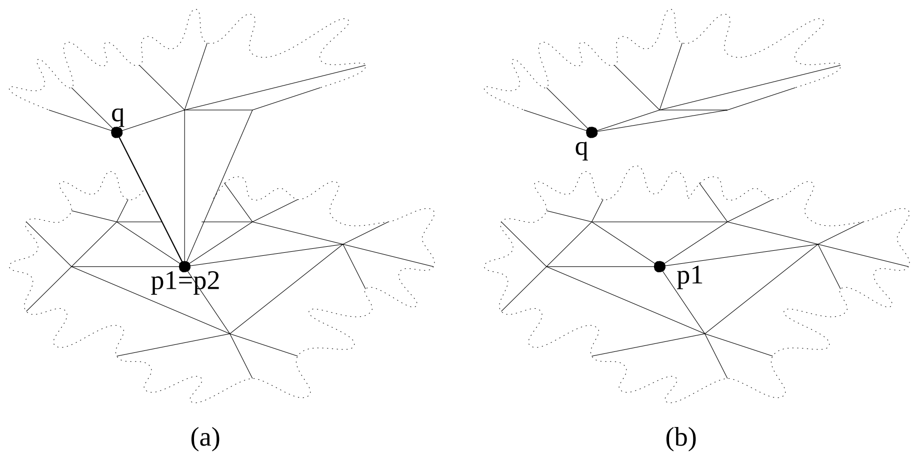


Figure 3.2: (a) A non-manifold vertex p is split into two coincident vertices p_1 and p_2; (b) later, p_2 is collapsed onto q, thus eliminating the non-manifold situation.

to one of its neighbors. In the same paper, Schroeder adopts a global error evaluation obtained by accumulating local errors.

Simplification Envelopes by Cohen et al. [CVM+96] use a different approach to guarantee that the simplified mesh is within a predefined distance h from the original one: they build two envelope meshes at distance $+h$ and $-h$ from the input mesh, and allow the removal of a vertex only if the new local mesh does not intersect the envelopes. Such envelopes are polygonal approximations of the offset surfaces[4] at distance $+h$ and $-h$ from the input mesh, but, unlike offsets, they do not self-intersect by construction. The method applies to manifold surfaces, guarantees a bounded global error, and preserves topology. In order to deal with manifold surfaces with boundary, they define *border tubes* (which are for a polygonal line the analogous of envelopes for a triangulated surface). Cohen et al. propose two different approaches for simplifying the mesh within the constraints of the two envelopes (and of the border tubes): removing either one vertex at a time, or a set of vertices at the same time.

3.2.3.4 Methods Based on Edge Collapse

Historically, the first approach based on edge collapse is the *Mesh Optimization* algorithm, proposed by Hoppe et al. [HDD+93]. This method performs a sequence of elementary modifications that try to minimize an energy function defined over the mesh as:

$$E = E_{dist} + E_{rep} + E_{spring}$$

where E_{dist} is the sum of the squared distances of the original points from the current mesh

[4] The offset surface at distance d from an entity $\mathcal{O} \subset \mathbb{E}^3$ is the locus of points of $\mathbb{E}^3$ lying at distance equal to d from $\mathcal{O}$.

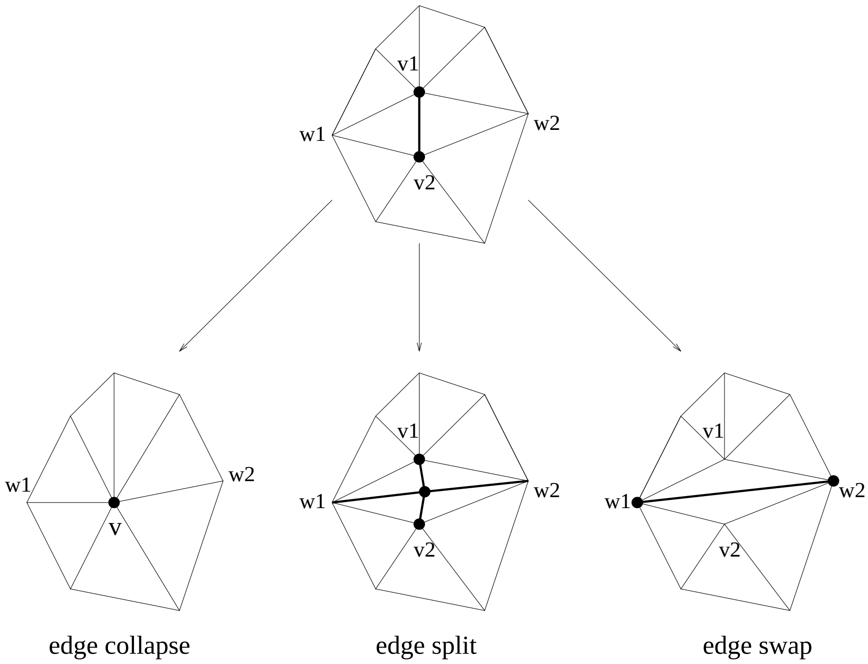


Figure 3.3: The three mesh optimization operators.

(and, thus, provides an estimate of the global error), E_{rep} is proportional to the inverse of the number of vertices, and E_{spring} is the sum of the edge lengths. In practice, they consider a weighted combination of the global approximation error, the size of the mesh, and the shape of the triangles (avoiding long edges, and thus slivery triangles). They modify the mesh by applying three elementary modifications (see Figure 3.3):

- *edge collapse*, which collapses an edge into a point belonging to its interior;
- *edge split*, which divides an edge into two by introducing a new vertex in its interior;
- *edge swap*, which replaces an edge e with the edge joining the two other vertices of the two triangles incident at e.

Of the three operators, only edge collapse reduces the size of the mesh; the other two are used to improve its accuracy. This method, and subsequent methods based on energy function optimization [HDD+93, Hop96], do not guarantee a bounded approximation error for the output mesh since the error is just one of the components of the function they try to optimize.

Later, Hoppe [Hop96] proposed a variant of mesh optimization using the edge collapse operator only. Since the size of the mesh monotonically decreases, the term E_{rep} (used in [HDD+93] to penalize the increase of size through vertex splits) is omitted from the energy function. Rather, Hoppe includes in the energy function other terms related to preservation of surface features and attributes. All candidate edge collapses are ordered by the estimated energy function improvement that they cause, and the vertex causing the highest improvement is selected at each step. The resulting approach is simpler, and allows a multiresolution model, called *Progressive Meshes*, be built based on it (such model will be described in Section 3.3).

The method by Ronfard and Rossignac [RR96] applies a variant of edge collapse in which an edge is always collapsed to one of its two endpoints. Note that, unlike in [HDD+93, Hop96], here the simplified mesh uses a subset of the vertices of the original one, without introducing new vertices. For this reason, this kind of edge collapse can be also seen as a special case of vertex removal. A peculiar characteristic of the method is the criterion adopted to decide which collapses to perform first. Each potential collapse has a cost, made of two components: LGE, which measures the approximation error of the mesh resulting from the collapse, and LTE, a penalty function that measures the likeness the mesh may self-intersect if the collapse is performed. LGE is measured for each vertex p by considering the square distance of p from the planes of its incident triangles; when two vertices collapse into one, the surviving vertex inherits the planes of both. This provides an approximation of the global error. Edge collapses are performed in increasing order of their costs.

Gross and Staadt [GS98] extend the simplification strategy based on edge collapse to three-dimensions: here, the size of a tetrahedral mesh is iteratively reduced by collapsing an edge into a new vertex, and its adjacent tetrahedra accordingly. A special attention is devoted to make sure that an edge collapse does not cause intersection or reverse the orientation of tetrahedra. Their energy function takes into account distance between the field values at the endpoints of the edge to collapse, preservation of volume, and shape of tetrahedra.

Getting back to triangle meshes, Algorri and Schmitt [AS96] first find vertices which are nearly coplanar, and then collapse edges connecting pairs of nearly-coplanar vertices into a new vertex. The method preserves the topology and the sharp features of the mesh, but does not guarantee a bounded global error.

The method by Gueziec [Gué96] is also based on edge collapse. Shorter edges are collapsed first; additional constraints prevent changes in topology, creation of slivery triangles, and variation of normals. The global error is bounded by maintaining a tolerance volume around the surface, whose width is, at any point, equal to the local approximation error. At each edge collapse, the algorithm checks that the width of the tolerance volume does not get larger than the given error bound.

Lindstrom and Turk [LT98] propose a triangle mesh simplification method based on edge collapse (in the variant that creates a new vertex) which does not take into account approximation error at all. Instead, they try to minimize the variation in the surface area and in the volume of the enclosed solid. They show experimentally that the resulting meshes have the same visual quality of those produced by methods that guarantee a bounded global error.

3.2.3.5 Methods Based on Triangle Collapse

The approach proposed by Hamman [Ham94] is based on a *triangle collapse* operator. It iteratively shrinks the triangle having minimum principal curvature at its vertices to a single vertex. The approach tends to reduce the density of triangles in flat zones, and preserves sharp lines. The error of the resulting approximation is not bounded. In a more recent paper [GHJ$^+$97], the same authors propose collapsing independent sets of triangles at the same time, bound the global error, and use the method as a basis for a multiresolution model (see Section 3.3.3).

Trotts et al. [THJW98] shrink a triangle to a point by performing two consecutive edge collapses. The same approach is used in three dimensions where three edge collapses shrink a tetrahedron to a vertex. They consider a field defined over the tetrahedral mesh, and control the coarsening process based on the field error: the tetrahedron causing the least error increase is iteratively collapsed, as long as the error remains beyond a threshold; they also consider the possibility of shrinking a set of independent tetrahedra, as in [GHJ$^+$97]. The authors pay attention to prevent changes of tetrahedra orientations and to preserve mesh boundaries.

3.2.3.6 Methods Collapsing Groups of Vertices

Garland and Heckbert [GH97, GH98] propose an iterative variant of vertex clustering [RB93] (see Section 3.2.3.1). They allow clustering just two vertices at a time. With respect to edge collapse, non-adjacent vertices can be collapsed as well, thus allowing modification of topology (see Figure 3.4). They evaluate the global error by defining a distance for each vertex, similar to LGE in [RR96] (see Section 3.2.3.4), but, instead of explicitly storing plane sets, they represent them as of quadric error matrices; in this way, combining the planes of two collapsed vertices is done efficiently by summing up two matrices. Vertex pairs to collapse are selected according to their distance.

Popovic and Hoppe [PH97] extend the principle of collapsing pairs of vertices to simplify not necessarily regular simplicial complexes of any dimension. Their final mesh may have top cells of any dimension, and topology simplification (in addition to geometry simplification) is one of the intended goals. The method is designed for fast visualization; they consider the preservation of visual attributes (e.g., colors) of the mesh elements. To improve realism, they visualize isolated vertices / dangling edges of the resulting mesh as points of size / lines of thickness proportional to the area of the portion of surface that collapsed into them.

3.2.3.7 Wavelet Methods

Wavelets are based on a recursive decomposition of a plane domain, and are thus suitable for two-dimensional fields and parametric surfaces (see Section 3.2.1). They can deal with free-

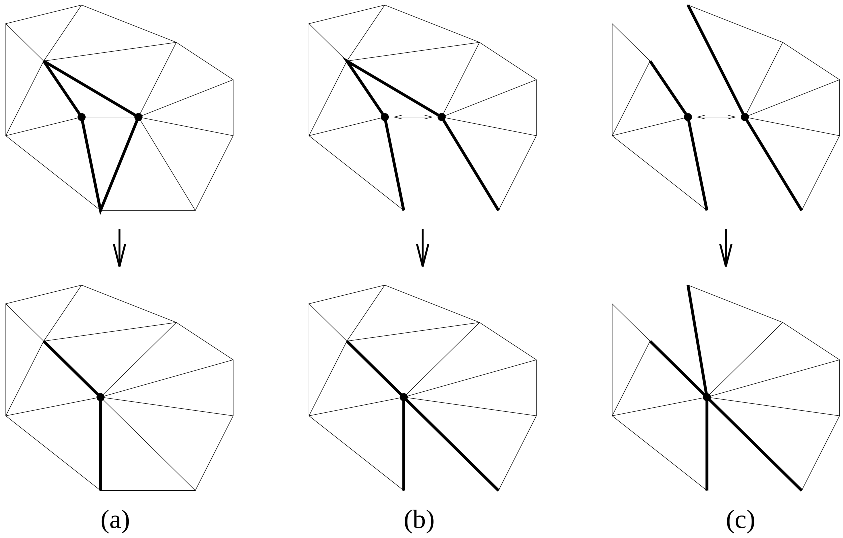


Figure 3.4: Different effects of collapsing a pair of vertices in a mesh: (a) an edge connecting the two vertices exists, an internal area is simplified; (b) an edge connecting the two vertices does not exist, a corner on the mesh boundary disappears (a hole is possibly closed); (c) an edge connecting the two vertices does not exist, two different boundary portions are joined (two connected components are possibly merged).

form surfaces provided that a surface is parametrized before applying the wavelet analysis.

In Eck et al. [EDRD+95], the given surface is partitioned into a small number of triangular patches and each patch is parametrized independently. Then, a quaternary subdivision (based on recursively refining a triangle into four reduced copies of it) is applied to each patch in order to define the wavelet transform. Note that the vertices of the quaternary subdivision in general fall inside a triangle of the original mesh, and thus their value must be estimated through interpolation. The vertices of the original surface are lost in the simplification process as well as sharp lines and corners. This method guarantees a bounded global error and provides a support to multiresolution.

3.2.4 Reconstruction of Surfaces and Solid Objects

The primary issue for reconstructing an unknown object given a set of scattered points or segments, or a sequence of contours, is defining a structure connecting the data. Reconstruction methods from segments or contours are not iterative, and thus our overview is more focused on reconstruction from scattered points. Data are located on the object surface and, thus, present an irregular distribution.

The aim is reconstructing either a representation of the interior volume, or a representation of the bounding surface of the object. In the volumetric approach, a tetrahedralization filling the interior of the object, and containing the given data as its vertices, or edges, is built. In the boundary approach, a triangle mesh passing through all the data is generated. Note that the volumetric approach provides a triangle mesh describing the surface of the object as a byproduct: such a mesh is the collection of all the triangles that are incident to just one tetrahedron.

Since the size of a triangle mesh at full resolution may be too large, an approximated complex can be built in a successive step, by using simplification methods [BBCS96], as described in Section 3.2.3.

3.2.4.1 Reconstruction from Scattered Points

A number of techniques have been proposed for reconstructing a triangle mesh from an unorganized set of points in space. Most techniques build first a three-dimensional structure, and then obtain a description of the boundary surface from it.

Hoppe et al. [HDD$^+$92] propose a non-iterative method to reconstruct a triangulated surface that does not necessarily pass through the given points, but reflects the shape of the original object. The method works in two steps: first, they define a function (a scalar field) in space that estimates, for each point of a three-dimensional cuboidal grid, the signed distance from the unknown surface; then, they extract the reconstructed surface as the isosurface corresponding to the zero value of the field, by applying a standard isosurface algorithm [WMW86]. The signed distance is defined by first estimating the tangent plane to the surface at each data point. The method works well for smooth surfaces, but it fails in reconstructing sharp lines. A post-processing phase improving the fitting to data (and allowing the reconstruction of sharp features) was introduced in [HDD$^+$94].

The *sculpturing method* by Boissonnat [Boi84] starts from a Delaunay tetrahedralization of the data points. Since the Delaunay tetrahedralization covers the convex hull of the input data, several points may remain inside, although they are known to belong to the surface of the object. The algorithm iteratively eliminates one tetrahedron at a time based on a heuristic criterion, in order to move internal points to the boundary of the resulting polyhedral shape. The heuristic criterion tends to remove tetrahedra that are not likely to be part of the object (e.g., because they have big triangular faces on the boundary of the current shape). A tetrahedron can be removed only if it leaves the resulting mesh connected and manifold. A drawback is that the sculpturing process can get locked before all the innermost vertices have been included into the boundary, because it may not be possible to remove further tetrahedra without a violation of topological consistency. In particular, it is not possible to reconstruct objects with a non-null genus (i.e., with through holes or handles) by removing one tetrahedron at a time without passing through a non-manifold condition.

The approach by Veltkamp [Vel93] applies the same principle, starting from a different geometric structure, called the γ-*neighborhood graph*. Such structure contains more triangles than a tetrahedralization, and includes the triangular faces of Delaunay tetrahedra. A triangulated approximation of the surface defined by the data points is obtained from such graph through iterative deletion of triangles lying on its boundary.

Bajaj et al. [BBX95, BBCS96, BBCS97] find the subset of the Delaunay tetrahedralization which better approximates the shape of an object through a method based on the concept of α-*shapes* [EM94]. The intuitive idea is having a sphere S_α of radius α, initially located outside the convex hull (i.e., out of the domain of the Delaunay tetrahedralization), and free of moving everywhere it can go without colliding with data points. The α-solid is the tetrahedral mesh obtained from the Delaunay tetrahedralization by removing all tetrahedra which can be intersected by S_α in its movement.

While varying the value of α, the α-solid gives a variety of tetrahedral meshes raising from the Delaunay tetrahedralization (for $\alpha \to \infty$) to the empty mesh (for $\alpha = 0$). The optimal shape description is provided by the α-solid for the smallest α such that the α-solid is a connected mesh, contains all data points inside or on its boundary, and its boundary is two-manifold. Bajaj et al. determine the optimal value of α through a binary search in the ordered sequence $l_1 \ldots l_k$ of lengths of the edges of the Delaunay tetrahedralization. An iterative variant, allowing a sequence of removal operations, has been proposed in [DFMP98d], and will be discussed in Section 8.6.

For a set of points uniformly sampled and dense enough, the α-solid found by the algorithm is a good approximation of the object. However, small cavities may remain filled with tetrahedra. Thus, a second phase is added to the algorithm in [BBCS96], which applies a sculpturing process to remove such tetrahedra. In [BBX95], the authors also address the problem of reconstructing a scalar field defined on the surface of the reconstructed object.

3.2.4.2 Reconstruction from Segments

Boissonnat et al. [BFLBM90] have proposed a method for reconstructing the boundary of an object from data acquired through a stereo process applied from several viewpoints. A set of straight-line segments is given, and, for each segment, the position of the viewpoint from which the segment has been acquired is known.

The method builds first a conforming Delaunay tetrahedralization of the given set of segments; then, it eliminates a suitable subset of tetrahedra in such a way that all given segments lie on the boundary of the resulting shape. In order to delete tetrahedra, the algorithm exploits the fact that triangles linking stereo segments to the corresponding viewpoints represent optical rays and, thus, cannot intersect the object (see Figure 3.5). For each triangle σ having an edge in one of the input segments and the opposite vertex in the viewpoint corresponding to that segment, all tetrahedra intersected by σ are deleted.

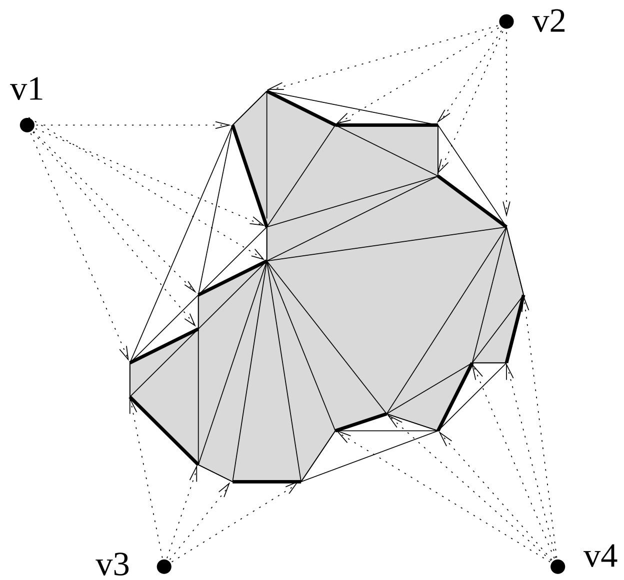


Figure 3.5: Object reconstruction from a set of segments acquired from four viewpoints v_1, v_2, v_3, v_4, according to [BFLBM90]. For simplicity, the example is shown in two dimensions. Solid lines denote the Delaunay triangulation, with data segments thick; dotted lines denote optical rays; white triangles are eliminated; the grey triangles remain in the final representation.

3.2.4.3 Reconstruction from Contours

The problem of finding a triangulated surface passing through a given sequence of cross sections is usually addressed by constructing a sequence of partial shapes, each of them connecting two cross sections lying on adjacent planes, and then glueing them.

The construction of a triangle mesh joining two adjacent cross sections involves three basic issues:

1. find a correspondence between the contours lying on the two cross sections (this problem is non-trivial if any of the two cross sections consists of more than one contour, i.e., more than one simple polygon);

2. triangulate the strip between two corresponding contours with triangles having one edge on one contour and the opposite vertex on the other contour (this problem is non-trivial if the contours have very different shapes);

3. solve branching situations, i.e., cases when one contour in a cross section corresponds to more than one contour on the adjacent section (this problem is non-trivial).

In the absence of a-priori knowledge, the solution of Problem 1 relies on heuristics, for instance

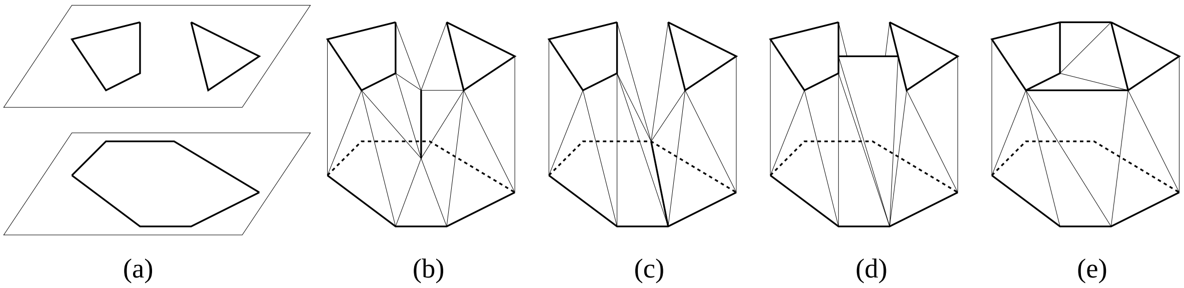


Figure 3.6: (a) A pair of cross sections with a different number of contours, and various ways to solve the branching situation: (b) introduction of lines at an intermediate level, (c) splitting the single contour, (d,e) joining the two contours.

taking into account the overlapping area [WA86].

Problems 2 and 3 are strictly inter-related and must be solved simultaneously: in practice, a contour may have one or more corresponding contours on the adjacent cross section. The basic problem is deciding which points of two contours must be joined with edges.

A number of algorithms reduces the problem to that of finding a minimum cost cycle in a toroidal graph [Kep75, FKU77]. Fast heuristics have also been proposed, which try to minimize measures such as edge length [CS78, GD82]. A survey, and experimental comparison can be found in [HT95]. These algorithms try to build the surface between two cross sections directly. Another approach [Boi88, Gei93] computes a Delaunay tetrahedralization for each pair of adjacent cross sections and then eliminates empty tetrahedra from it.

Existing algorithms solve branching situations in different ways. Suppose that we have one contour on cross section i and two corresponding contours on section $i+1$. Some methods break the single contour by tracing a chord of the polygon, and then process the resulting two pairs of corresponding contours independently [Boi88, Gei93] (see Figure 3.6 (c)). Some methods connect the two contours on cross section $i+1$ in some way (for instance, by considering their common tangent lines), and reduce to having a single pair of corresponding contours [CS78, Sha81, SK91, BS94] (see Figure 3.6 (d,e)). Finally, some methods introduce lines on an intermediate plane to model the place where the branching occurs [EPO91, BCL96] (see Figure 3.6 (b)). This last class of algorithms is perhaps the one giving the more natural results. Bajaj et al. [BCL96] use the medial axis for this purpose.

The triangle-based approach to surface reconstruction from cross-sections is not the only possible one. Sloan and Hrechanyc [SH81] address the surface reconstruction problem as that of generating a sweeping rule for a generalized cylinder representation giving rise to the contours. This approach is especially well-suited to deal with sparse data, and provides acceptable reconstructions from cross sections which are too sparse for the smoothness assumption common to the above mentioned triangulation algorithms.

3.2.5 Discretization of CAD data

CAD modelers allow the definition of smooth surfaces as collections of parametric patches, which are glued at their boundaries (see Section 2.2.3).

A triangle mesh approximating each patch is usually obtained by defining a triangulation of a polygonal approximation of its domain in parameter space, and lifting it to a triangle mesh in $\mathbb{E}^3$ by assigning the function value to each vertex [SH92, VB93, Che93]. To this aim, various points on the domain are sampled to become the vertices of the triangulation. A problem in this approach is that a domain triangulation may create a self-intersecting surface when lifted to three dimensions. This may happen because the sampling is not dense enough, or because of surface singularities. For the first problem, methods have been proposed which adapt the density to the local curvature of the surface. The second problem is solved by recognizing critical points and lines of the domain (i.e., those that map onto surface singularities), and further subdividing the domain along such curves and points before triangulating [BR94].

When several parametric patches form a surface, a major issue is ensuring matching between the approximations of any pair of neighboring patches. In fact, the parametric domains of neighboring patches are independently defined, and there is no information about which pairs of trimming curves, from two different parametric domains, contribute to the same curve in physical space. If two corresponding trimming curves in two adjacent faces are discretized into polylines independently, then their images in $\mathbb{E}^3$ may not match, and the overall discretized surface may present cracks [Del97].

Many authors do not address this problem, and leave its solution to the mesh simplification methods that will be applied to the resulting surface. Here, simplification methods that can accept a non-manifold input, and act both on the geometry and on the topology of a mesh (e.g., [RB93, Sch97, GH97, GH98]), have a special interest since they may be used to repair artifacts in meshes that are expected to represent manifold objects, but contain non-manifold conditions due to errors.

R. Klein and W. Straßer [KS95] use an error-based incremental algorithm to triangulate each parametric domain. At each step, the algorithm inserts the point causing the maximum error with respect to the current triangulation; if such point lies a trimming curve, then the corresponding point in the adjacent patch is also inserted in the triangulation of that patch.

In [DFMP98d], an approach is proposed which first builds a polygonal approximation of the trimming curves in $\mathbb{E}^3$. Later, each approximated trimming curve is back-projected in the parameter space of the patches sharing it and its segments are used as constraints for building a triangulation of the interior of each patch. This work will be expanded in Section 8.3.2, as the basis for a multiresolution model.

3.3 Multiresolution Models

In this section, we review multiresolution models proposed in the literature. All such models can be interpreted within the framework of the *Multi-Complex* that will be introduced in Chapter 4.

A multiresolution model provides a way to manage approximate representations of a geometric object at different resolutions. The construction of multiresolution models relies on algorithms for the *iterative refinement and coarsening of a cell complex*, reviewed in Section 3.2. The algorithm used to build a multiresolution model has a relevant importance in determining its properties. The following issues are important in evaluating the good quality of a multiresolution model:

- *Range of applicability*, i.e., the class of spatial objects it can represent.
- *Properties required on the input data*: an important distinction is between models that need data to be distributed according to a *predefined, regular pattern*, and those which do not. Models based on fixed-pattern subdivisions offer in general limited possibilities in fitting available data and/or adapting to the shape of an object.
- *Shape of the cells*: cells with elongated shape (slivers) should be avoided, since they may cause problems in numerical processing, as well as artifacts in visualization.
- *Expressive power*, i.e., the variety of complexes that can be obtained from it: an important distinction is between models that can provide cell complexes at *variable resolution and accuracy*, and those that cannot.
- *Compression factor*, i.e., what resolution is necessary in order to achieve a target accuracy in an extracted complex.
- *Extraction algorithms*, i.e., how easy it is to extract a complex either at uniform or variable resolution and accuracy.
- *Other operations* possibly supported by the model beside that of obtaining a complex at a certain accuracy (e.g., spatial interference queries).
- Possibility of *smooth transitions*, also called *geomorphs*, between consecutive levels of resolution, i.e., of gradually deforming the complex at a certain resolution into the complex at the next higher resolution, instead changing the representation instantaneously; this feature is important in visualization in order to avoid undesired popping of details in the mesh.

For the last item in the list, we note that multiresolution models supporting geomorphs are those based on collapse operations (edge collapse [Hop96, Hop97, Hop98b, XV96, XESV97, DWS+97, MMS97, GTLH98, Paj98], triangle collapse [GHJ+97], vertex-pair collapse [LE97], etc.). In fact, in these cases it is possible to transform the situation before the collapse into that after the collapse (or conversely) by changing the location of vertices gradually. Models based on vertex removal or insertion do not in general have this property, unless in special cases (e.g., collapsing an edge by shrinking it to one of its vertices is a special case of vertex removal).

We classify existing multiresolution models in the following three types, according to their structure:

- *Layered multiresolution models*, which encode a sequence of independent complexes, each with a predefined accuracy/resolution, and provide no support for adapting accuracy to desired (variable) error thresholds.

  This is the only type of models that can be built from a refinement or a coarsening algorithm that performs global modifications on a complex. For global modification we mean an operation that modifies the whole complex each time it is applied (for instance, resampling the vertices of the complex [Tur92, RB93]).

- *Nested multiresolution models*, which encode a family of nested cell complexes, organized into a tree.

  Such models are generated by refinement algorithms that modify a complex by expanding a cell at a coarser resolution into a local complex at a higher resolution, or by coarsening algorithms that unify several cells at a higher resolution into one cell at a coarser resolution. Note that the coarsening approach is only possible when data have a regular distribution, and and reduces to subsampling a self-similar complex; the refinement approach can be applied to any data distribution (see [DFP95b] for a survey).

  The nested structure of these models has several advantages, as far as algorithms operating on them are concerned (e.g., they can act as spatial indexes), but constraints are imposed on cell shape and structure of complexes that can be obtained from them.

- *Evolutionary multiresolution models*, which encode the evolution of a complex through a refinement/coarsening process consisting of a set of local modifications. By local modification we mean an operation that affects just a small set of cells in a complex (e.g., all those sharing a vertex [dBD95, SZL92, CVM+96, BBCS96], or those around a collapsed edge [Hop96, XV96, XESV97, Hop97, LE97, DWS+97, MMS97, GTLH98]).

  Evolutionary models encode a *sequence* of local modifications [Hop96, KS97] or a *partial order* which captures their mutual dependencies [DF89, CPS97, dBD95, Pup96, Pup98, Hop97, MMS97, XV96, XESV97, LE97, KCVS98, GTLH98, DWS+97, Cig97]. Thus, they can be further classified into *sequential* and *dependency-based* models.

3.3.1 Layered Models

A *layered model* consists of an ordered sequence of cell complexes representing an entity at increasing resolution and accuracy. The expressive power of layered models is limited to the complexes of the sequence: these are a small number and their accuracy / resolution is predefined (in general, it is uniform in space). Thus, the possibility of adapting to the needs of user applications is scarse. The extraction of a cell complex reduces to selecting the complex in the sequence, whose characteristics are the closest to application needs.

Such models are standard technology in graphics languages and packages, such as OpenInventor™ [Gro94, Wer94], and VRML [VRM96], and are used to improve efficiency of rendering: depending on the distance from the observer, one of the available models is selected. The approach works well for small or distant object, but it is not efficient for large objects spanning a range of different distances from the observer.

3.3.2 Nested Models

Nested models encode the recursive subdivision of a domain into nested cell complexes: a cell at a certain resolution is expanded into a cell complex at a higher resolution. Usually, the carrier of the complex expanding a cell γ is constrained to geometrically coincide with γ. Nested models based on this approach apply to domains having the same dimension as the space in which they are embedded, and thus they can be used for representing solid objects, fields, explicit and parametric surfaces (which can be regarded as special cases of fields). Nested models have also been proposed for free-form surfaces by first reducing them to a parametric form (for instance, in [EDRD$^+$95] a surface is split into pieces and such that each piece is parametrized independently by projecting it onto a plane).

The simplest nested models are based on fixed-pattern self-similar complexes (see Section 2.3.4), i.e., they use a subdivision into cells of predefined geometry and topology [Sam90a, SS92, WvG94, VHB87, DWS$^+$97, EKT97, HK95, LKR$^+$96]. Such models can be encoded by extremely compact and efficient data structures, because the hierarchy is implicitly defined on the basis of fixed patterns. The regularity of their structure is also their main limitation. In the case of scalar fields, data points must necessarily be regularly distributed, or resampling and interpolation must be used to infere regularly distributed data from the available ones, with a consequent loss of accuracy. In the case of parametric surfaces, the shape of the domain must coincide with the predefined cell type, therefore it is difficult to handle patches with complicated trimming curves. Wavelet approaches are also based on self-similar subdivisions with a fixed pattern, and have similar drawbacks.

Other nested models are based on irregular simplicial complexes [DFP95a, SP92]. The major drawback of these structures in inherent in the coexistence of an irregular subdivision with a nesting structure: since every cell is subdivided independently, they cannot guarantee that

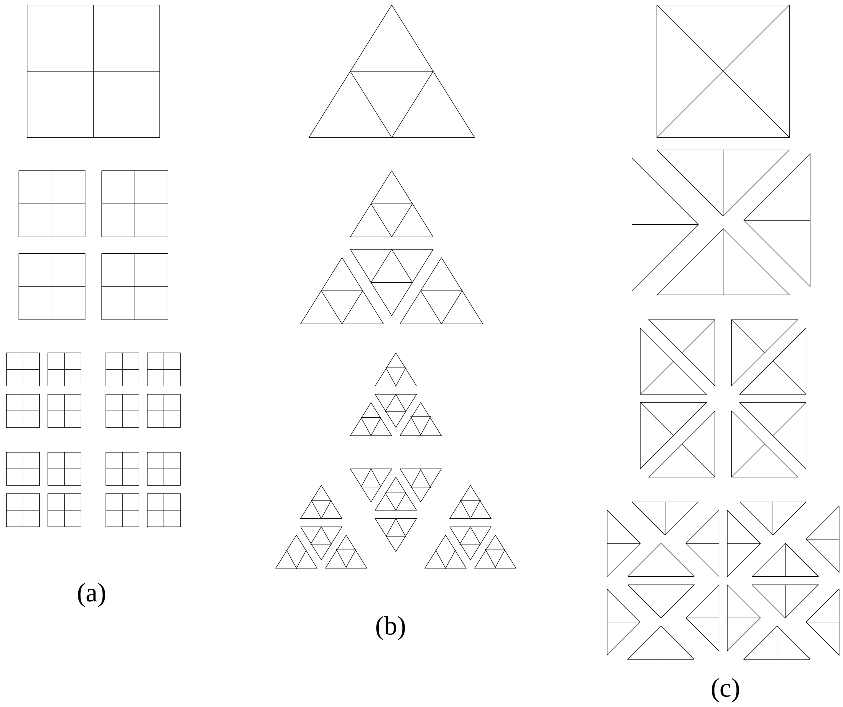


Figure 3.7: Nested two-dimensional models based on fixed-pattern complexes: (a) quadtree, (b) quaternary triangulation, (c) hierarchy of right triangles.

an extracted complex satisfies optimality criteria (e.g., the Delaunay criterion) on the whole domain.

In the following, we describe the various proposals in more details.

3.3.2.1 Quadtrees and their Variants

Quadtrees are two-dimensional nested models which recursively subdivide a square domain into four quadrants [Sam90a, Sam90b] (see Figure 3.7 (a)). The resulting hierarchy can be represented as a quaternary tree, where each node corresponds to a complex made of four square cells, while its children represent the four complexes refining each such cell. A quadtree is used to approximate two-dimensional fields by associating a bilinear function with each cell. The *quaternary triangulation* [GG79] is a straightforward generalization of the quadtree to a triangular mesh. The recursive splitting rule consists of subdividing a triangle into four equal triangles (see Figure 3.7 (b)). The quaternary triangulation is widely used in domain decomposition for the finite element methods, and especially suitable to represent fields defined on a spherical surface (e.g., on the surface of the Earth) [Lou94, LDW97, LS98]. As for quadtrees, the resulting hierarchy can be represented by a quaternary tree. The quadtree has been generalized to three and higher dimensions, see [WvG94] for the three-dimensional case.

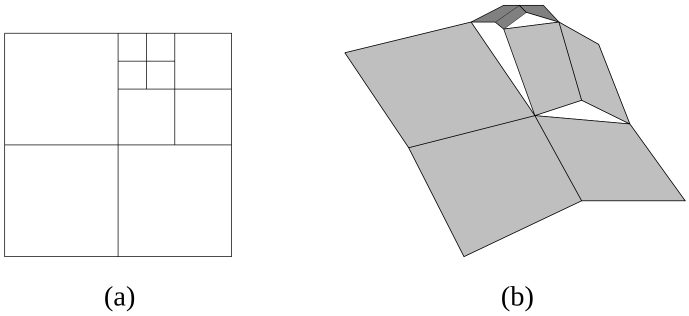


Figure 3.8: (a) A mesh combining cells at different resolutions from a quadtree, and (b) the corresponding surface in $\mathbb{E}^3$; note the cracks.

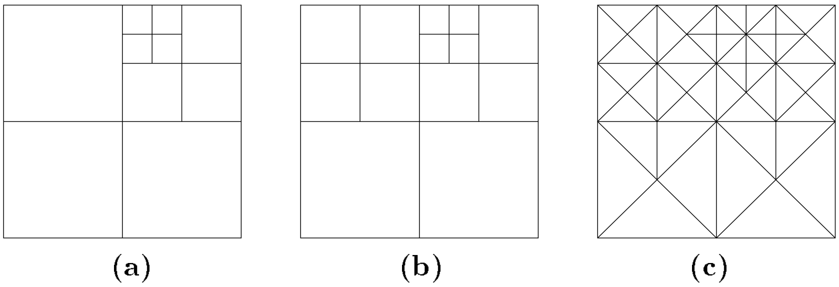


Figure 3.9: (a) The set of cells (not a complex) corresponding to the leaf level of a quadtree; (b) the set of cells at the leaf level after balancing; (c) the final triangulation.

A rich set of algorithms has been developed on quadtrees and their higher-dimensional extensions, including Boolean operations, point location, range queries, nearest neighbor search, and many others (see [Sam90a, Sam90b] and references therein, [Ore91]).

The main drawback with quadtrees and their variants is in the impossibility of combining cells from different resolutions into a cell complex: the boundary of two neighboring cells γ_1 and γ_2

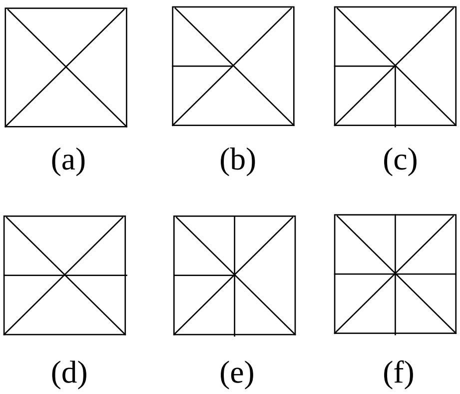


Figure 3.10: The six patterns for triangulating a cell in a restricted quadtree.

of different sizes (i.e., belonging to different levels of the tree) is not compatible, and thus any set of cells containing both γ_1 and γ_2 is not a complex. If the quadtree represents a field, the image and graph of function f have a discontinuity between γ_1 and γ_2 (see Figure 3.8).

Von Herzen and Barr [VHB87] first proposed a method to build cell complexes from a quadtree (and thus, avoid cracks). The method works in two steps:

1. First, the quadtree is locally balanced by further subdividing some of its leaf squares, in such a way that two leaves of the resulting tree differ for no more than one level (see Figure 3.9 (b)).
2. Next, a triangulation of each leaf square is built by using the four corners plus those mid-points of its edges, that are vertices of neighboring squares (see Figure 3.9 (c)). Sixteen possible configurations are possible, that are obtained through rotational symmetry from the six patterns shown in Figure 3.10.

As a result, the domain subdivision corresponding to the leaves of the tree is a triangulation, and the piecewise-linear function defined on it is continuous. The resulting structure is called a *restricted quadtree*. Note that the triangulation in Step 2 produces triangles that are not part of the original structure, and thus there is no control over their accuracy.

Sivan and Samet [SS92] propose two algorithms for building a restricted quadtree using a bottom-up and a top-down approach, respectively. Such algorithms build a locally balanced tree directly and use piecewise-linear functions inside each node during construction. In order to achieve a linear running time, the algorithms must test the accuracy at each datum only a constant number of times, but this prevents the possibility of guaranteeing the accuracy of the resulting triangulation. An algorithm able to guarantee a final triangulation within a given error bound is proposed in [DFMP94], based on an iterative top-down strategy. The complexity of the approach is in $O(n \log n)$ for n data points.

3.3.2.2 Hierarchies of Right Triangles

Hierarchies of right triangles are based on the recursive split of a right triangle into two [DWS$^+$97, EKT97, HK95, LKR$^+$96] (see Figure 3.7 (c)). The hierarchy is described as a binary tree where each node corresponds to a complex made of two triangles. Variable resolution complexes can be obtained by combining adjacent nodes from consecutive levels of the tree.

Although in [HK95] recursive splitting has been only used as a mesh simplification mechanism, full exploitation of this kind of hierarchy as a support to variable resolution has been proposed independently in [LKR$^+$96], and in [EKT97], for the purpose of terrain visualization in flight simulation.

In [EKT97], a data structure that maintains the tree of triangles explicitly is adopted, where, for each triangle, only its accuracy is stored, while geometry is provided implicitly by its position in the tree. A mesh at variable resolution can be extracted by traversing the tree top-down, and recursively refining triangles that do not satisfy the accuracy needed for the current eye position. At each refinement operation, further refinements may be enforced on adjacent triangles in order to obtain a complex.

In [LKR+96], only the raw grid of vertices is stored, while both the triangles and the tree are only implicitly encoded. A complex at variable resolution is extracted by a bottom-up visit of the tree. The basic idea is that a pair of adjacent triangles can be merged into their parent if the consequent loss of accuracy is small enough. Since accuracies of triangles are not stored, and computing them exactly is too expensive, heuristics are adopted to estimate them on-the-fly. Further heuristics are used to purge large portions of the domain (e.g., those outside the view frustum).

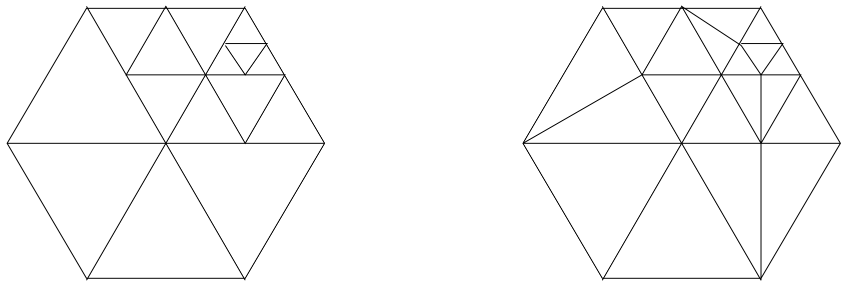
Real-Time Optimally Adapting Meshes (ROAM), by Duchaineau et al. [DWS+97], encode a hierarchy of right triangles by storing dependency relations corresponding to the cases in which a triangle can be split leading to a valid modification of the mesh only if a triangle adjacent to it is split. By propagating dependencies, they are able to extract triangulations at variable resolution (they consider view-dependent error bounds). A similar approach is used by Pajarola [Paj98], where, in addition, a scheme for organizing the model on disk is described. Since dependency relations define a directed acyclic graph, these models will be considered again when dealing with dependency-based models (see Section 3.3.4.3).

This kind of hierarchy has been extended to the higher dimensional cases, and used to model hypersurfaces produced by scalar fields over (hyper)cubic domains. Higher dimensional models maintain similar characteristics of regularity, and easy symbolic manipulation (see [Heb94] for details on the three-dimensional case).

3.3.2.3 Hierarchies Based on Wavelets

Eck et al. [EDRD+95] define wavelets on a surface by previously parametrizing it (see Section 3.2.5). They use a recursive domain subdivision that follows the pattern of a quaternary triangulation. Note that the vertices of the complex, on which wavelets are defined, are not the same as those of the original surface. Thus, even the maximum resolution mesh available from the model is only an approximation of the original one. Eck et al. use this multiresolution model as a support to *mesh editing*. Wavelets are especially well-suited for this purpose because the finer details of the mesh are encoded as differences with respect to the coarser ones and not as absolute values. Thus, as one changes the shape of the mesh at a coarse level, all the details remain attached to it in a consistent way.

Wavelets can provide meshes at uniform resolution easily, since each step of signal reconstruction involves two complete levels of the hierarchy. The level of resolution can be chosen by stopping reconstruction after a certain number of steps. Getting meshes at variable resolution is more complicated and involves problems of consistency as those occurring with quadtrees



(a) (b)

Figure 3.11: (a) A set of triangles (not a triangle mesh) obtained by combining triangles from different levels of a quaternary triangulation; (b) the triangulation built from the set in (a).

and quaternary triangulations.

Gross et al. [GSG96] use wavelets as a basis for a terrain model from which it is possible to extract on-line restricted quadtrees that satisfy a certain error bound *variable in space*. It is essentially a modification of the reconstruction stage in the standard wavelet approach. In wavelets, each iteration (from a coarser to a finer level) in the reconstruction stage adds all the details of the new level to the current representation, and refines the underlying square grid uniformly over the domain. In [GSG96], details are added only in a certain area of space, namely only where the application needs a more accurate representation than the current one. The resulting domain subdivision is a set of squares of different sizes, which are not a cell complex (exactly as in the case of quadtrees); this subdivision is then refined into a triangulation, with an approach similar to restricted quadtrees. The method applies to terrains and explicit two-dimensional surfaces; it can be extended to higher dimensions. Note that the algorithm for extracting a triangulation requires consistent floating-point computations.

Zorin et al. [ZSS97] use wavelets based on a quaternary triangulation to support variable resolution, and multiresolution mesh editing. To obtain sets of cells at variable resolution which are triangle meshes, they insert a final triangulation step similar to restricted quadtree (see Figure 3.11). They also propose a local variant of the reconstruction algorithm, which reconstructs a triangulation covering just a portion of the object. The user can perform mesh editing on the model by acting on a certain triangulation extracted from the model, which corresponds to a level of detail fine enough to recognize the shape features, and coarse enough to allow real-time rendering. If the user changes the position of a vertex p lying at a certain, intermediate, level of the structure, then this change is automatically propagated to the finer levels thanks to the fact that wavelets encode finer levels as difference from coarse ones: when reconstructing the mesh, the same details that were added around the old location of p, now will be added around its new location. Propagation of the changes to the coarser levels requires updating the locations of some points in the structure, namely of those points for which p represents a detail.

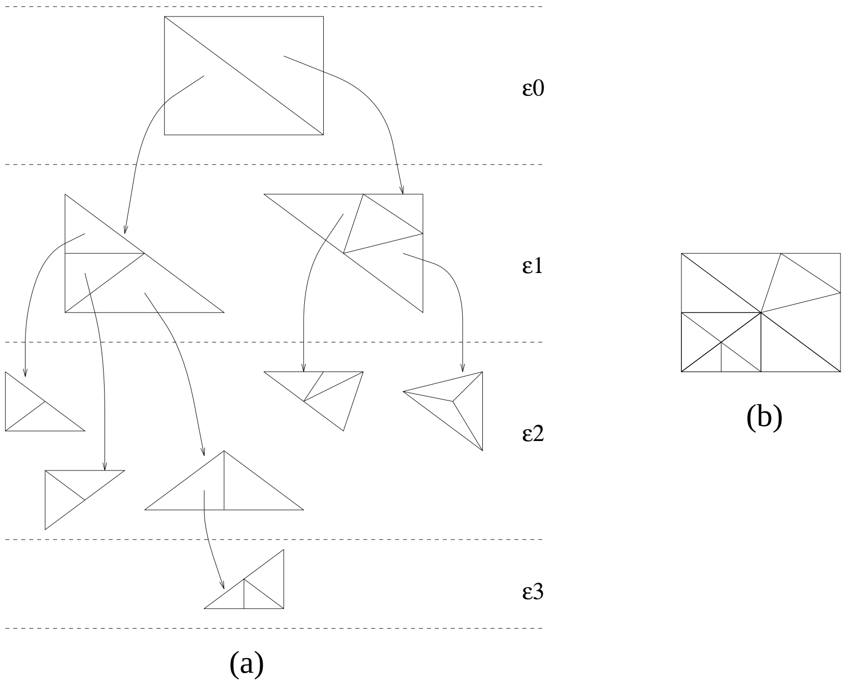


Figure 3.12: (a) A hierarchical triangulation build based on a sequence of four error thresholds $\varepsilon_0 > \varepsilon_1 > \varepsilon_2 > \varepsilon_3$; (b) an example of a triangulation obtained by combining patches from different levels of the hierarchy.

3.3.2.4 Hierarchical Triangulations

Nested models based on irregular triangulations have been proposed for representing two-dimensional fields [DFP92b, SP92, DFP95a]. These models can work on arbitrary datasets because no constraint is imposed on the vertex distribution. A model is built top-down by iteratively refining the triangles of an initial coarse mesh. A triangle is refined by inserting one or more points either inside it, or on its edges, which are chosen on the basis of an error-driven criterion.

The construction of the hierarchy is driven by a predefined, decreasing sequence of error thresholds: each level of the tree represents the field within one of the given thresholds, starting from the coarsest one; in passing from one level to the next one, a triangle is iteratively refined until the error of the resulting local triangulation drops below the next error threshold. The points inserted in order to refine a triangle σ are chosen according to an error-driven criterion, that always tries to insert the point currently causing the maximum approximation error inside or on the boundary of σ. Figure 3.12 shows an example of a hierarchical triangulation.

The most extensively studied model is the *Hierarchical Delaunay Triangulation* (also called Hierarchical Triangulated Irregular Network, HTIN) [DFP92b, DFP95a], with the proposal of algorithms for performing various operations typical of GIS applications (briefly reviewed

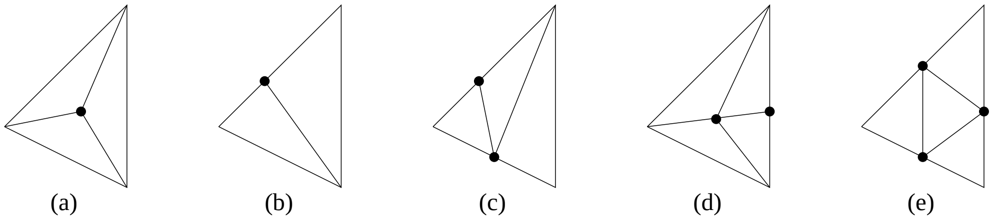


Figure 3.13: Predefined patterns for splitting a triangle in [SP92]: (a) interior point; (b) one side point; (c) two side points; (d) interior and side; (e) three side points.

next). In an HTIN, a triangle is refined by inserting first points on its edges, and then inside, and by updating a Delaunay triangulation after each insertion of a point.

The Adaptive Hierarchical Triangulation by Scarlatos and Pavlidis [SP92], is similar to an HTIN, but here the mesh refining a triangle is generated based on heuristics, rather than being a Delaunay triangulation. A triangle σ is refined by iteratively considering four key points, namely the point of maximum error lying inside and on each of the three edges of σ, and then splits σ according to one of five predefined patterns (see Figure 3.13).

In both models, the error-driven policy for vertex insertion in a triangle and the stopping condition at a predefined error threshold ensure that, for each pair of adjacent triangles at a given level, their common edge either remains unrefined, or it is refined by inserting the same set of points in both triangles; thus, the two resulting triangulations can be glued consistently. Edges that survive across different levels of the hierarchy permit to combine triangles from different levels of the tree. In general, many refined edges reduce the possibility of combining complexes from different levels of accuracy, and thus the expressive power of the model (if all edges are refined, the result is a layered model). On the other hand, a model with few refined edges may contain many slivery triangles. Being based on heuristics, the construction method used in [SP92] does not guarantee that the shape of triangles is well maintained. The construction strategy of [DFP95a] produces triangles with a better shape because it locally builds a Delaunay triangulation; moreover, they give a method to determine on-line (i.e, during construction) an optimal sequence of tolerances such that only few edges are subdivided at each level, and the resulting hierarchy virtually spans a continuous range of accuracies.

The issue of obtaining representations at variable accuracy is not addressed in [SP92]; in [DFP95a], an algorithm for extracting a mesh at variable accuracy is presented. The algorithm extracts first a set of triangles satisfying the given accuracy requirements, which is not necessarily a triangulation; then, it transforms it into a triangulation by decomposing some triangles; the approach is similar to restricted quadtrees, and has the same drawbacks.

In [DFM97] two algorithms for mesh extraction at variable accuracy are proposed, which output a triangulation of the domain formed by triangles of the HTIN.

The first algorithm applies the following idea: whenever a triangle σ is refined into the triangulation Σ that covers it, then any triangle σ_j, which is in the current solution and is adjacent to σ along an edge of σ refined in Σ, must be refined as well (even if σ_j already meets the desired accuracy requirements). The algorithm descends the tree in a top-down fashion: starting from the triangulation at the root, it iteratively refines every triangle σ, which is not accurate enough with respect to the criteria set by the user; in addition, it also refines any triangle σ_j, adjacent to σ along an edge that gets refined. Triangles to be refined are managed by means of a stack.

The second algorithm is suitable for error thresholds increasing with distance from a viewpoint, and follows a similar approach as the method proposed in [CPS97] for mesh extraction from a Hyper-Triangulation (to be described in Section 3.3.4.4). The algorithm starts from a triangulation made just of the triangle at the highest available resolution which contains the viewpoint. Then, it incrementally enlarges the current triangulation by adding adjacent triangles which satisfy the given threshold function. The fact that the threshold function increases while moving away from the viewpoint ensures a new triangles can always be added to enlarge the current triangulation.

The model proposed in [DFP95a] has been extended to three dimensions in [BBDFP94], and applied to volume data visualization.

Algorithms for various operations on HTINs have been developed by the research group at the Department of Computer and Information Science of the University of Genova (including the author of this thesis), especially in the two-dimensional case [DFP92b, DFGMP94, DFP95b], including operations specific for geographic applications: extraction of isolines [DFMP93], visibility [DFM94a, DFM95a, DFM95b, DFM97] and overlay [BDFM95a].

Testing the mutual visibility of two points p and q reduces to testing whether the segment connecting p and q lies completely above the surface. On an HTIN, this can be checked in a top-down traversal of the tree, using the tree structure as a spatial index to eliminate parts of the domain that are not relevant (because they do not contain the vertical projection of segment pq), and errors of triangles to eliminate parts that certainly lie completely above or below the segment [DFM97]. Staring from the root, in each visited node, we find the triangle(s) intersecting the projection of pq. For each intersecting triangle σ, if σ is a the maximum resolution, then we compare σ with segment pq in $\mathbb{E}^3$; otherwise, the search should continue in the subtree expanding σ. Indeed, searching such subtree can be avoided in two cases: if pq passes above / below σ at a height difference larger than $\varepsilon(\sigma)$, then, we can directly conclude that certainly pq is above / below any triangle in the subtree. This basic approach extends to visibility queries involving other types of entities than points [DFM97].

Another visibility problem addressed on HTINs is the computation of the viewshed (i.e., of the visible and invisible parts of the terrain surface) from a viewpoint [DFM94a, DFM95a, DFM97]. An HTIN permits the solution of this problem on a terrain representation at variable resolution, where accuracy is higher near the viewpoint and progressively decreases with distance. Traditional approaches to visibility computation are based on traversing a terrain in front-to-back order with respect to the viewpoint. On an HTIN, such traversal can be interleaved with a traversal of the tree. At each step, we extract from the hierarchy the next triangle, at the required accuracy, that must be traversed by the visibility algorithm. Error information stored in the model is used to prune branches of the tree corresponding to areas are clearly totally visible or totally invisible (in a similar way as in the case of visibility queries). In [DFM95b], the update of visibility information for a new terrain representation at a different accuracy is solved by using dynamic algorithms.

Classical overlay algorithms are based on a sweep-line approach, where the triangulation underlying a TIN and a plane map are traversed ("swept") from left to right by progressively moving a vertical line across the plane; while the sweep-line is moved, pairs of overlapping elements are detected and their intersections computed. In [BDFM95a] an extension is proposed of such approach to HTINs, which allows computing the overlay of a map with a terrain representation at a user-defined accuracy. The method consists of interleaving the movement of the sweeping line with a traversal of the tree describing an HTIN, in order to find triangles on the fly.

3.3.3 Sequential Evolutionary Models

Sequential evolutionary models encode the history of an iterative process either refining or coarsening a cell complex.

3.3.3.1 Models Tracing the Life of Cells

By tracking the change of accuracy of a cell complex through a process of refinement or coarsening, it is possible to assign a *life* to each cell, namely the range of accuracies through which the cell "survives" in the mesh. The model stores all the cells that appear during the process, each tagged with its life. Life of cells is used as a filter: a complex at uniform accuracy can be obtained by collecting all those cells whose life contains a given error. In [BDFM95b], cells are stored in a search structure with their life as key, and filtering from the sequence can be performed in optimal time.

The model is independent of the dimension of the complex and of the embedding space. It is also independent of the type of modifications performed: the only requirement is that the error must monotonically decrease after each modification (see [BDFM95b]). The expressive power is limited to the cell complexes generated during the construction process, which have a uniform accuracy.

3.3.3.2 Models Storing a Sequence of Modifications

Other sequential models store a coarse cell complex plus a sorted sequence of local modifications that progressively refine it. Such models allow very compact data structures, based on the fact

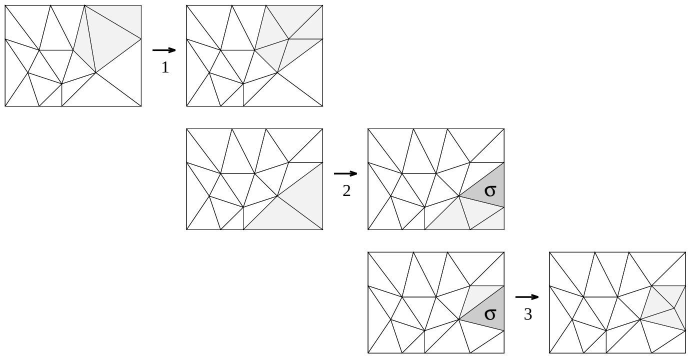


Figure 3.14: A sequence of three modifications of a triangle mesh. Modifications 1 and 2 are independent; modification 3 depends on modification 2 because it replaces triangle σ, which does not exist in the mesh if modification 2 has not been performed.

that all modifications in the sequence belong to a predefined type, and thus can be described with a few parameters. A cell complex at uniform accuracy can be extracted by starting from the initial one, scanning the list of modifications and performing modifications from the sequence until the desired accuracy is obtained. In this case, the output complex is one of those generated during the construction process. For extracting a cell complex at a variable accuracy, the idea is scanning the list of modifications, performing only those relevant to achieve the required accuracy. Unfortunately, a straight scan is not sufficient, since skipping a modification may prevent performing some later modification that is necessary to fulfill the accuracy. We say that a modification C_j *depends*[5] on a previous modification C_i in the sequence (and C_i *blocks* C_j), if C_j cannot be performed unless C_i has been performed, i.e., if C_j modifies the same cells introduced by C_i (see Figure 3.14). Therefore, dependencies between modifications must be computed on-the-fly, thus involving complex, and computationally expensive algorithms. This is done in different ways depending on the model.

The model by Klein and Strasser [KS97] applies to parametric surfaces, and makes use of a Delaunay triangulation in parameter space. It extends to general surfaces after providing a parametrization. The local modification operator consists of inserting a vertex into a Delaunay triangulation. The model stores only the simplest mesh, plus the sorted sequence of vertices inserted during refinement, each tagged with the approximation error of triangles incident at such vertex at time of its insertion. The great advantage in terms of storage requirement is paid in terms of efficiency in the traversal algorithms: an algorithm for extracting a domain-uniform mesh must scan the sequence of vertices and explicitly build the mesh through an on-line Delaunay triangulation algorithm [Wat81, GS85, GKS90] until the desired accuracy is achieved. An algorithm for extracting domain-variable meshes is also outlined in [KS97], which requires non-trivial tests based on the circumcircle criterion (hence involving heavy numerical computation) in order to trace vertex dependencies.

[5]This concept will be formally defined in Chapter 4.

Progressive Meshes (PM), proposed by Hoppe [Hop96], apply to generic surfaces in space. Here, the basic local modification is *edge collapse* (see Section 3.3.4), which is reversed into *vertex split* to obtain an equivalent refinement sequence. The model stores just the initial mesh, and the sequence of vertex splits; each vertex split requires one new vertex plus three indices to existing vertices. The extraction of a mesh at uniform accuracy is simpler than in [KS97], because no numerical computation is required, while the connecting structure of the existing mesh is modified through symbolic updates. An algorithm for extracting meshes according to variable error thresholds is just outlined in [Hop96], which requires involved traversals of the list to find vertex dependencies on-line. The approach has been also applied to mesh transmission [Hop98a] (see also Section 3.4).

Progressive Simplicial Complexes by Popovic and Hoppe [PH97] extend PMs by considering collapses of pairs of vertices not necessarily connected by an edge, and define *generalized vertex splits* as their inverse operations. They are able to deal with non-regular complexes (having top cells of any dimension) and multiresolution applies both to geometry and to topology.

Gieng et al. [GHJ+97] propose a sequential model which is built, for a triangulated free-form surface, through a bottom-up technique based on triangle collapse. Each transition between two consecutive meshes corresponds to a set of independent triangle collapses that are performed simultaneously and without interfering with each other. They store the initial mesh at low resolution and the sequence of groups of vertices which must be simultaneously expanded into triangles in order to get the next finer level of resolution. The issue of extracting meshes at variable resolution is not addressed in [GHJ+97].

3.3.4 Dependency-Based Evolutionary Models

Rather than storing a totally ordered sequence of local modifications, *dependency-based models* link each local modification to the set of modifications that block it. Thus, there is no more need of complicated procedures to find out dependency between modifications, as in sequential models.

3.3.4.1 Delaunay Pyramids

In its original formulation [DF89], the Delaunay pyramid is a layered model for terrain surfaces, made of a sequence of Delaunay triangulations, whose approximation errors obey to a predefined sequence of tolerances $\varepsilon_0 > \ldots > \varepsilon_h$. Each triangulation is obtained from the one preceding it in the sequence by applying the mesh refinement algorithm described in [FL79] until the next accuracy is met. All levels are encoded in a data structure which stores all spatial interferences between triangles of consecutive meshes in order to support interference queries (e.g., point location).

In [BDFM95b], a pyramid with many levels is considered, where some triangles "survive"

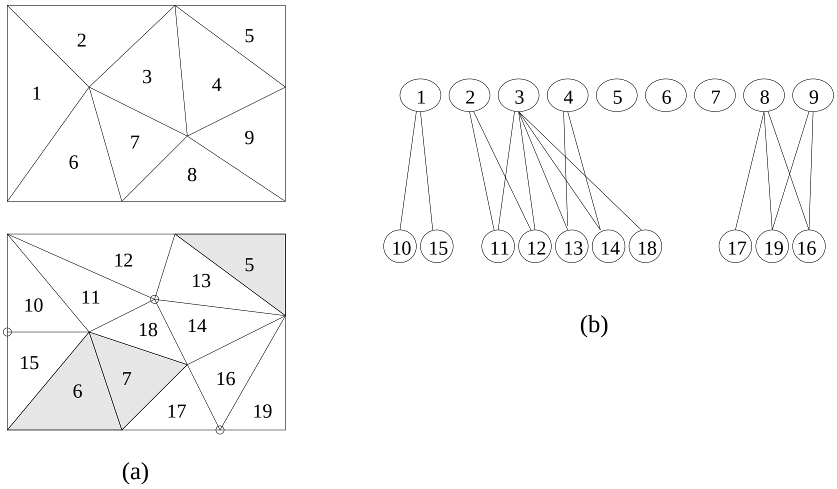


Figure 3.15: (a) A Delaunay pyramid with two levels; (b) the graph stored in [BDFM95b].

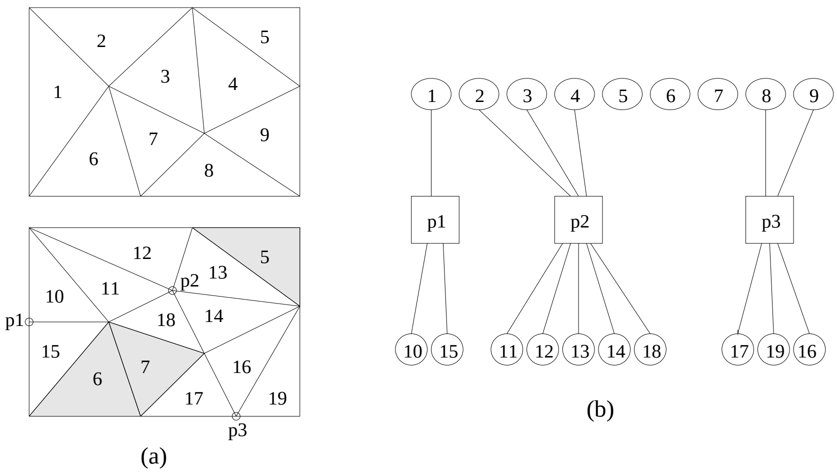


Figure 3.16: (a) A Delaunay pyramid with two levels, the upper one obtained from the lower one by removing vertices p_1, p_2, p_3; (b) the graph stored in [dBD95].

across different levels, and a more compact data structure is proposed which avoids replicating surviving triangles (see Figure 3.15). Only an algorithm for mesh extraction at uniform accuracy is proposed.

The model proposed by De Berg and Dobrindt [dBD95] is essentially a Delaunay pyramid built bottom-up by iteratively removing an independent set of vertices of bounded degree. The model is described by a directed acyclic graph (DAG) encoding the interference relations between a removed vertex and the triangles present in the mesh before and after its removal (see Figure 3.16). The height of the DAG (i.e., the maximum length of a root-to-leaf path) is logarithmic in the number of nodes. This property guarantees a good worst-case complexity of geometric

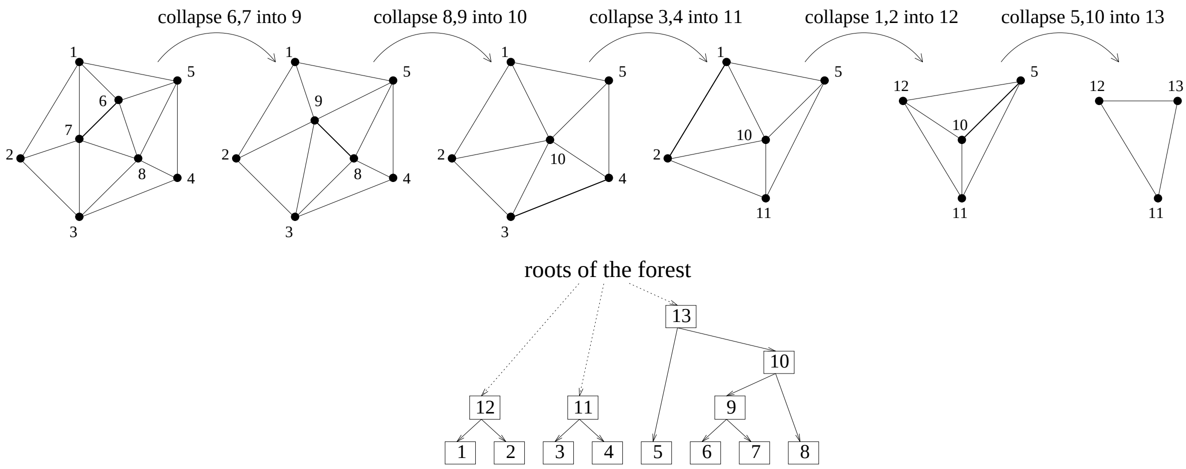


Figure 3.17: A sequence of edge collapses and the corresponding binary forest of vertices.

search algorithms such as point location.

In [dBD95], an algorithm is described, for extracting, in time linear in its output size, a representation at variable resolution based on an arbitrary error threshold function. The algorithm is based on a top-down traversal of the pyramid, during which a visited triangle is accepted in the solution if its accuracy satisfies the threshold function. Unfortunately, this greedy approach does not guarantee that the desired threshold will be fulfilled everywhere: accepting a triangle automatically prevents performing the local modification that removes it, but such a modification may be necessary in order to refine some triangles that are encountered later during the traversal.

Delaunay pyramids have been initially designed for two-dimensional scalar fields, and have been extended to the three-dimensional case for volume visualization [CDFM+94]. The framework could be extended to deal with parametric surfaces, by using a constrained Delaunay triangulation in parameter space. An extension to surfaces in space is hardly feasible due to the difficulty of reproducing the Delaunay refinement framework on free-form meshes embedded in $\mathbb{E}^3$.

3.3.4.2 Hierarchies of Vertices

In the attempt to obtain efficient algorithms for selective refinement from a Progressive Mesh, data structures encoding vertex dependencies have been proposed [Hop97, MMS97, XV96, XESV97].

Such models use a primary structure consisting of a *binary tree* (actually, a forest) *of vertices*, where the children of each vertex w are the endpoints v_1, v_2 of the edge that collapsed in w (see Figure 3.17). Of course, v_i $(i = 1, 2)$ can be split only if w has been split, but this is not

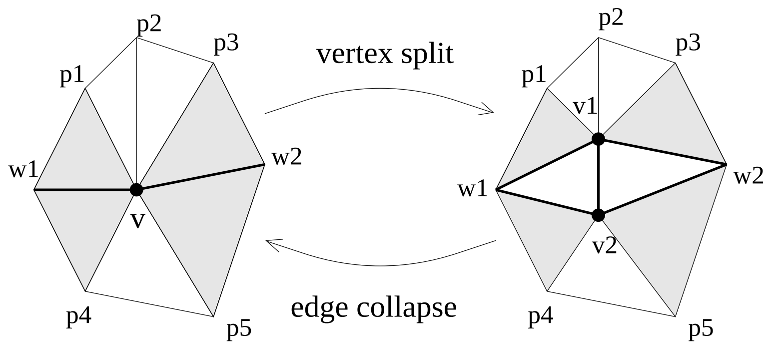


Figure 3.18: Edge collapse and its reverse vertex split. According to [XV96, XESV97], vertices v_1 and v_2 depend on the vertices of all triangles incident at v (i.e., on vertices $w_1, w_2, p_1, p_2, p_3, p_4, p_5$); according to [Hop97], v_1 and v_2 depend on the vertices of the shaded triangles (i.e., on $w_1, w_2, p_1, p_3, p_4, p_5$); according to [MMS97], v_1 and v_2 depend just on w_1 and w_2.

sufficient. Further dependency information for a vertex v is stored at the tree node representing v: these additional dependency relations are defined differently in the various proposals.

In [XV96, XESV97], a PM is built by collapsing independent sets of edges, following a technique similar to that used in [dBD95]. In this way, the binary tree is guaranteed to be balanced. Additional vertex dependencies are defined based on the observation that a vertex v can be split while extracting a mesh only if all other vertices that surround v at the moment when v is split in the original PM sequence belong to the current mesh already (see Figure 3.18): thus, a link between v and such vertices is stored in the model. Variable resolution extraction from such a structure is performed in two steps: first, the binary tree is traversed bottom-up, and vertices corresponding to too fine details are discarded; next, vertex dependencies are traced back from the remaining vertices in order to include all vertices that must be necessarily inserted. The extracted mesh is obtained by performing vertex splits only at selected nodes, in the original order of the PM. The main drawback of this algorithm is that the hierarchy is visited bottom-up, hence all vertices of the model must be analyzed to obtain even the simplest possible mesh.

In [Hop97], Hoppe stores the dependency of a vertex v from just six of the vertices that are adjacent to v when v is split in the original PM sequence: namely, the vertices of the triangles that are adjacent to the two triangles created when v is split (see Figure 3.18). It is claimed that such information is sufficient to determine all vertex dependencies. A *dynamic* algorithm is proposed, which extracts a mesh at variable resolution starting from a current one (this allows relevant time saving if the two meshes are close to each other). The algorithm traverses the tree, executing an edge collapse whenever the resolution requirements allow it and the collapse is legal for the dependency relations, and forcing vertex split whenever the user parameters require it: a split operation is performed by recursively managing vertex dependencies through a stack.

In [Hop98b], such model is applied to terrains, and enriched with a technique for the construction and management of a terrain database using the PM. A PM is built for a large terrain dataset (too large to fit in memory) by dividing it into square blocks and applying edge collapses within each block independently of the others. In this phase, boundary vertices are retained up to the coarsest level: this ensures that no vertex dependencies with adjacent blocks are created, and thus each block has its own local binary tree of vertices. Then, a new hierarchy is built whose most refined level corresponds to the union of the coarsest levels of all blocks. The resulting model has a global upper (i.e., coarser) part and several, mutually independent lower (i.e., finer) parts, one for each block. At run-time, the relevant blocks for the current view are loaded and a triangle mesh is extracted from them to be visualized.

Independently of [Hop97, Hop98b], *Progressive TINs* have been developed by Maheshwari, Morin and Sack [MMS97], which apply the principle of progressive meshes to terrain modeling. They consider a vertex v as dependent from its parent in the binary tree, and from the other endpoints of the two edges that are expanded into triangles when splitting v (see Figure 3.18). They store the binary tree of vertices and retrieve the remaining dependency relations on the fly. In addition, they store an axis-parallel bounding box associated with each vertex v, that completely encloses the area of the domain affected by splitting v and its descendants. Such boxes are used to drive the tree traversal in answering spatial queries on a Progressive TIN. The authors propose an algorithm that extracts a terrain inside a window box and algorithms for various other queries typical of GISs: point location, extraction of isolines, test for the mutual visibility of two points; for the last two operations, height information is also stored at the vertices, in addition to the bounding boxes. Finally, they suggest the use of R-trees [Gut84], based on the vertex bounding boxes, as an access structure for the model on secondary memory.

The model by Luebke and Erikson [LE97] is based on *generalized* vertex splits: a vertex may be split into two or more vertices not necessarily connected by an edge. It is proposed as a unifying model which can be built from any mesh simplification method using vertex clustering, vertex-pair collapse, or edge collapse. In fact, the generalized vertex split is the inverse of all such operations. This model keeps just the mesh at full resolution and the binary forest of vertices, ignoring all other existing vertex dependencies. Any sub-forest containing all the roots determines a mesh at variable resolution; such a mesh is obtained as a deformation of the mesh at the highest resolution (i.e., the one containing all the vertices) by visualizing active vertices (i.e., vertices of the subforest) at their actual positions, and non-active ones at the position of their nearest active ancestor; degenerate triangles are not mapped onto the screen.

Kobbelt et al. [KCVS98] developed a method for storing the positions of the vertices appearing in the finer resolutions of a Progressive Mesh as differences with respect to the positions of vertices appearing in the coarser resolutions. This method is used as a support for the definition of smoothing approximations of a surface, and for multiresolution mesh editing, based on a similar principle as in wavelets.

3.3.4.3 DAG-Based Models

Guéziec et al. [GTLH98] propose a model based on the edge-collapse operator, which explicitly encodes dependencies between modifications through a directed acyclic graph (DAG). Each node of the DAG represents an edge collapse, its children represent collapses that are blocked by it, and its parents represent collapses that block it. They consider edge collapses which contract an edge into one of its two endpoints. The data structure is especially compact: it exploits the fact that only the geometry of triangles changes in an edge collapse (possibly causing some triangle to become degenerate), but not their connectivity. Thus, just the triangles of the mesh at maximum resolution are stored as three vertex references each. An extracted mesh is characterized by a subset of collapses that have been performed. Depending on whether the collapse removing a vertex v has been done or not, v can be represented, in a triangle using v, by v itself, or by another vertex w onto which v has been collapsed. The data structure maintains a table which allows to retrieve the correct vertex representatives.

A *dynamic* algorithm for extracting a mesh at a uniform accuracy is described in [GTLH98]. The algorithm uses two priority queues of edge collapses:

- a first queue contains the collapses that have been performed in the current mesh (and that may be undone, i.e., the vertex may be split), indexed by decreasing error of the mesh before undoing the collapse (the idea is that collapses that cause a bigger error must be undone first);
- another queue contains the collapses that have not been performed in the current mesh (and that may be done), indexed by increasing error the mesh resulting from doing the collapse (the idea is that collapses that cause a smaller error increase must be done first).

Given the current content of the two queues, and a new target accuracy for the mesh, the algorithm first undoes collapses from the first queue (i.e., it splits vertices) until the error drops below the bound, then it perform collapses from the second queue, as long as the error remains within the bound.

Real-Time Optimally Adapting Meshes (ROAM), by Duchaineau et al. [DWS+97], consider a dependency graph corresponding to a hierarchy of right triangles and apply the same principle as [GTLH98] for extracting a triangulation at variable resolution. Priorities are recomputed and the two priority queues are rearranged each time the error threshold function changes. Pajarola [Paj98] uses a similar model and proposes two algorithms for extracting a mesh, based on a top-down and on a bottom-up traversal of the DAG, respectively. In both papers, the intended application is the interactive visualization of terrains, and view-dependent error bounds are considered.

The model in [Paj98] is also proposed for progressive transmission. In addition, a technique is proposed for the management of a terrain database built on the multiresolution model. A large

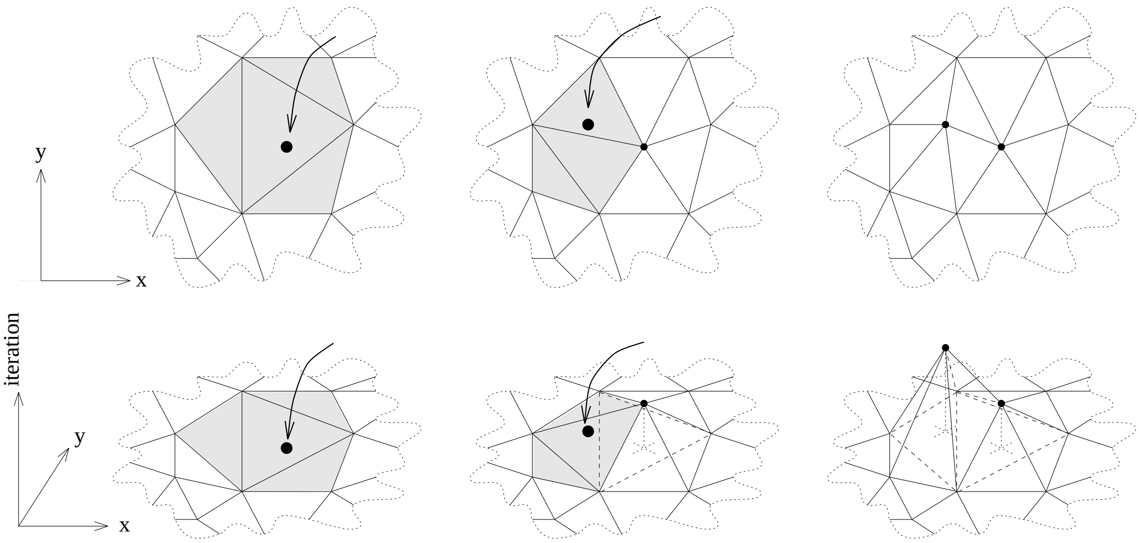


Figure 3.19: Representing modifications of a triangulation through embedding into a higher-dimensional space.

dataset is divided into square blocks; at run-time, the relevant blocks for the current view are loaded and a triangle mesh is extracted from them to be visualized. Most dependencies in the DAG are local to a block; however, dependency relations for boundary vertices may involve points in adjacent blocks, and must be propagated.

The *Multi-Triangulation* is also based on encoding dependency in a set of modifications under the form of a DAG. The model does not rely on the use of a specific algorithm for generating the modifications (it does not assume, for instance, that they consist in collapsing an edge / splitting a vertex, or they consist of removing / inserting a vertex, or any other specific strategy). This model was first proposed in [Pup96, Pup98] for triangulations in the context of terrain modeling, and lately extended to d-dimensional scalar fields represented as simplicial complexes in $\mathbb{E}^d$ [DFPM97], and to two-dimensional surfaces as triangle meshes in $\mathbb{E}^3$ [DFMP97b]. It supports the extraction of meshes at variable accuracy with a static or with dynamic algorithm, and focusing on an area of interest [DFMP98c]. This model is not reviewed here because it is a straight-forward instance of the general model introduced in Chapter 4 for multiresolution cell complexes.

3.3.4.4 The Hyper-Triangulation

The *Hyper-Triangulation* [CPS97] can be viewed as a model that encodes dependencies between modifications by reducing them to adjacency relations.

It considers a sequence of modifications refining a triangle mesh. The triangles involved in the given modifications are embedded into a higher dimensional space, where an additional coordinate axis means the time of a modification in the original sequence. For each modification, consisting of removing a set Γ_1 of triangles and replacing them with Γ_2, we "paste" Γ_2 over Γ_1, as shown in Figure 3.19. Each modification is thus visualized as a pair of triangle meshes sewed at their common boundary edges, and called a *bubble*. The data structure encodes the triangles and their adjacency relations in the higher-dimensional space by using a facet-edge data structure [DL89]. Each triangle in the Hyper-Triangulation is tagged with its *life*, defined as explained in Section 3.3.3. The resulting data structure is quite heavy.

In [CPS97] an algorithm is proposed to extract triangle meshes at variable accuracy in the case of error thresholds monotonically increasing with distance from a given viewpoint (e.g., those needed in flight simulation). The algorithm starts from a triangle near the viewpoint, and incrementally extends the mesh by adding one adjacent triangle. While moving away from the viewpoint, the resolution is eventually reduced according to the given threshold. The algorithm checks the lives of triangles to ensure that, at each moment, the current partial triangulation can be completed to cover the whole domain. The extracted model is guaranteed to satisfy the threshold function everywhere. The computational complexity is suboptimal in the worst case, i.e., $O(n \log n)$, where n is the size of the output mesh.

Hyper-Triangulations built from a Delaunay refinement technique [FL79] are used to model terrain surfaces in [CPS97]. Hyper-Triangulations built from a decimation technique [CCMS97] are used for representing manifold surfaces in the context of an interactive multiresolution modeler [CMRS97]. The Hyper-Triangulation directly extends to higher dimensional cases [Cig97], but the size of the data structure based on adjacency information can grow exponentially with the dimension of the complex, and its cost rapidly becomes prohibitive.

3.4 Compression of Triangle Meshes

Techniques for encoding a geometric complex as a sequential bitstream with small storage requirements have relevant impact in saving disk storage, and in speeding up transmission, and rendering. The study of such techniques is generally referred to as *geometric compression* (see Section 2.6). Compression methods have been developed for triangle meshes and are aimed at two complementary tasks:

- compression of vertices, i.e., of numerical information attached to each vertex of the mesh (position and attributes: surface normal, color, texture, etc.);
- compression of connectivity, i.e., of information describing the triangles as triplets of vertices.

Compression of connectivity is related to the topic of this thesis, since multiresolution structures can be used as the basis for compression methods. Thus, after briefly mentioning some approaches to compression of vertices, we will review existing methods for compression of connectivity.

The aim of methods for *compression of vertices* is minimizing the number of bits needed to store the position of a vertex v, the surface normal at v, and pictorial attributes like color and texture.

Deering [Dee95] reduces positions, normal and color to an arbitrary (small) number q of bits through quantization and delta-encoding. *Quantization* reduces positions from to q bits by normalizing the dataset to a unit cube and truncating the exceeding less significant bits. This is a lossy method which involves overlaying a grid with cell size $1/2^q$, and thus varying the value of q provides different quantization levels. In addition, positions of vertices appearing in a sequential bitstream can be *delta-encoded*: just the difference between a vertex position and that of its predecessor in the bitstream is encoded. Since the difference is likely to be small, it needs fewer bits to be represented.

Chow [Cho97] studied the problem of selecting the right quantization and proposes a quantization assignment method that first separates the mesh into regions of similar detail based on triangle size and curvature and then assigns different quantization levels for different parts of the mesh, with a careful treatment at the boundary of two regions with different quantization to avoid cracks.

Techniques for *compression of connectivity information* of a triangle mesh can be roughly classified into:

- *Direct methods*, which are aimed at minimizing the number of bits to encode a given triangle mesh "as it is". The bitstream must be read and processed completely by a decompression algorithm in order to obtain an description for the mesh.
- *Progressive methods*, which encode a coarse approximation of the mesh and a sequence of details which must be added to it in order to recover the initial mesh. An interrupted bitstream provides an *approximation* of the whole surface at a lower resolution.

3.4.1 Direct Methods

Classical direct methods proposed in the literature are based on the decomposition of a triangle mesh into *triangle strips*, or generalizations of triangle strips [ESV96, Cho97, Dee95]. Such methods are especially well-suited for visualization, but less to mesh transmission or encoding on disk because they do not provide triangle adjacencies, which should be reconstructed separately at an extra cost. Other methods allow reconstruction of adjacency relations from the encoding bitstream [TG98, DFMP98a].

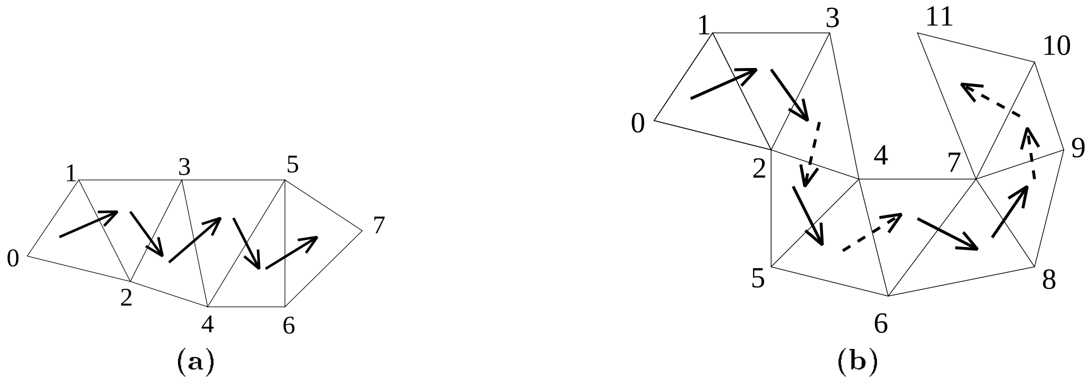


Figure 3.20: (a) A triangle strip; the corresponding vertex sequence is $0, 1, 2, 3, 4, 5, 6, 7$. (b) A generalized triangle strip; solid arrow correspond to new triangles added as in a plain strip, i.e., by turning in the opposite way to the previous turn (zig-zag pattern); dashed arrows correspond to new triangles added by turning in the same direction as the previous turn (fan-like pattern).

3.4.1.1 Methods Based on Triangle Strips

A *triangle strip* is simply specified by listing the vertices of its triangles in a zig-zag order (see Figure 3.20 (a)); each vertex appears only once in the sequence. The drawback of such methods is that several strips are necessary to cover a triangulation and vertices shared by more than one strip need to be replicated.

A more general form of strip is given by so-called *generalized triangle strips*, which are standard technology in graphics API like OpenGl. In this case we do not have a zig-zag sequence, but each new vertex may correspond either to a right or to a left turn in the pattern. The encoding bitstream either uses a binary code, called a swapcode, to specify how to connect each new vertex to the last encoded/decoded triangle, or it repeats the vertex in case of two successive turns with the same orientation (see Figure 3.20 (b)).

On the average, each vertex appears twice in the resulting bitstream. It has been shown [ESV96] that the problem of partitioning a triangle mesh into a minimum number of strips is NP-complete. Evans et al. [ESV96] give several heuristics in which they try to maximize the number of triangles per strip, to minimize swaps, to avoid strips formed by just one triangle.

Other methods [Dee95, BYG96] also allow the reuse of more of the past vertices than the last two ones (as in generalized triangle strips), by storing the last k vertices in a buffer; small codes indicating how to operate on such buffer are interleaved in the bitstream. Deering [Dee95] proposes a buffer of 16 positions and a syntax for the encoding, giving rise to the so-called generalized meshes. Bar-Yehuda and Gotzman [BYG96] studied the extent to which we can increase the buffer size to reduce vertex duplication. They also show that the problem of minimizing the buffer size for a given mesh is NP-hard. Chow [Cho97] gave algorithms for building generalized meshes which try to optimize through heuristics the use of the buffer.

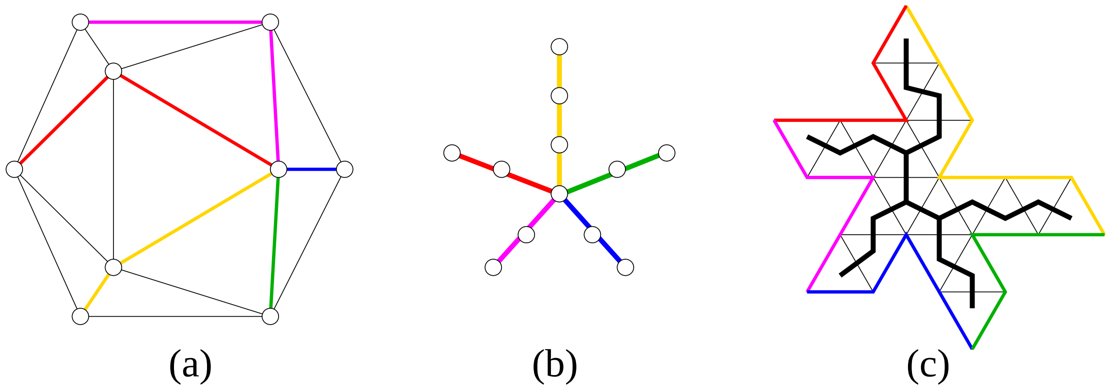


Figure 3.21: (a) The surface of an icosahedron, (b) the vertex spanning tree, and (c) the triangle spanning tree.

Compression rates are about 11 bits of connectivity information per vertex in [Dee95, Cho97] and from 2.5 to 7 in [ESV96].

Such methods encode just the geometry of triangles and not (or only partially, i.e., within a strip) their adjacency relations. They are especially convenient for surface rendering, but they are less suitable for mesh transmission when the received mesh must undergo operations which require adjacency information. In fact, adjacency relations must be reconstructed separately with extra processing.

3.4.1.2 Topological Surgery

Topological surgery by Taubin and Rossignac [TR96, Ros97] cuts a mesh along a set of edges that correspond to a spanning tree of vertices; such tree is determined by selecting the minimum number of edges in the mesh which connect all its vertices. If we imagine to cut the given mesh along the edges of the vertex spanning tree, we obtain another triangle mesh which contains the same triangles as the original one, and whose dual graph[6] is structured as tree, thus forming a spanning tree of triangles of the original mesh (see Figure 3.21).

This is true for meshes of genus zero, but a small number of edges can be added to the edges of the vertex spanning tree to give the same decomposition in the case of meshes of arbitrary genus. The authors show that the number of these additional edges is twice the value of the genus. The triangle spanning tree can be decomposed into a collection of triangle strips, that the authors call *runs*. Runs are connected through branching triangles, each of which connects three runs.

The bitstream produced by the method contains the two spanning trees of vertices and of

[6] The *dual graph* of a triangle mesh is a graph whose nodes correspond to triangles and whose arcs correspond to pairs of adjacent triangles in the mesh.

triangles, and a rather complicated algorithm permits the reconstruction of the original mesh. Through experiments on manifold meshes Taubin and Rossignac show a compression of 4 bits per vertex on average. The proximity of the vertices on the vertex spanning tree is exploited to efficiently encode positions: they use the ancestors in the tree to predict vertex positions, and thus need to encode only the differences between predicted and actual vertex positions; the correction terms are then compressed by entropy encoding. The method produce longer strips than the previously discussed ones; however, it requires that all vertices are kept in main memory during both compression and decompression and both the coding and the decoding algorithms perform consistent graph manipulation and are not easy to implement.

3.4.1.3 A Method Growing a Polygon

Touma and Gotsman [TG98] propose an compression method for manifold triangle meshes aimed at efficient transmission. The method works for surfaces without boundary, and extends to surfaces with boundary by adding dummy vertices.

Both the coding and decoding algorithms maintain an active list of edges which bounds a (not necessarily simply-connected) polygon. Such polygon separates triangles already encoded / decoded (located inside the polygon) from triangles that have still to be to encoded / decoded (located outside). At each step, the polygon is enlarged by encoding/decoding all the triangles lying outside it which are incident in one of its vertices.

The resulting bitstream contains each vertex once, and three types of commands: ADD, SPLIT and MERGE, where the the first two commands have one argument and the third command has two arguments. The argument of ADD is the degree d of a new vertex to be inserted, the arguments of SPLIT and MERGE are integer numbers with the meaning of indexes and offsets. Such commands mark those points, during the encoding/decoding process, where the number of boundary curves of the current polygon increases by splitting one curve into two, or decreases by merging two curves into one, respectively (recall that the polygon may not be simply-connected).

With the exception of MERGE operations (which are rare), both the encoding and the decoding algorithms run in linear time in the number of mesh edges.

The method generates an ADD command for each vertex of the mesh. The number of SPLIT and MERGE commands depends on the mesh. The authors do not report any evaluation of the number of SPLIT or MERGE commands that can occur: they just observe that MERGE commands can occur only for meshes of positive genus.

If represented "as is", the encoding sequence would require at least 5 bits per vertex: two bits for the command (since we have three possible commands) and three bits for the degree (assuming that the degree of any vertex is less than 8); the bits necessary to represent possible SPLIT and MERGE commands add to these. But Touma and Gotsman observe that the sequence of codes can be compressed further. Since the average degree of a vertex is 6, there is a spread of degrees around this value; moreover, there are easily long subsequence repeating the same commands and the same arguments. Thus, they use entropy coding combined with run-length encoding, getting experimentally compression rates from 0.2 to 2.4 bits per vertex for connectivity.

3.4.1.4 A Method Growing a Simple Polygon

In [DFMP98a, DFMP98b], we have proposed a direct compression method based on shelling. As the previous one, this method is designed for mesh transmission or disk storage because it encodes both triangles and their adjacency relations.

A *shelling sequence* of a triangle mesh Σ is a sequence $\sigma_1, \sigma_2, \ldots, \sigma_n$ of triangles, containing each triangle of Σ once, and such that, for any prefix subsequence $\sigma_1, \sigma_2, \ldots, \sigma_m$ (with $m \leq n$), the boundary of the union of the set of triangles $\{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is a simple polygon. The method applies to triangle meshes that are *extendably shellable*. A mesh is extendably shellable if it admits a shelling sequence (i.e., it is *shellable*), and moreover every shelling sequence defined for a subset of its triangles can be extended to a shelling sequence for the whole mesh. Extendably shellable meshes include all the triangulations of surfaces homeomorphic to a sphere or a disk [BM71]; surfaces with a non-null genus are not extendably shellable. The method can also deal with generic triangle meshes: if the input mesh is not extendably shellable, then it is automatically partitioned into shellable patches and each patch is encoded independently.

The bitstream implicitly encodes a shelling sequence of the given mesh. It contains the vertices of the mesh (each vertex exactly once) interleaved with control codes providing connectivity information. There are four control codes: VERTEX, SKIP, LEFT, and RIGHT, each represented on two bits. Coding and decoding algorithms are very simple and work in time linear in the size of the mesh without performing any numerical computation. Both the triangles and their adjacency links are recovered from the bitstream with no extra processing. Moreover, only the boundary of the current polygon and triangles incident to its edges have to be kept in memory, while encoded/decoded triangles lying completely inside the polygon can be discarded.

Both the encoding and decoding algorithms maintain the *simple polygon* bounding the currently encoded/decoded portion of the triangulation in a doubly-linked list; they also keep a *queue of active edges*, which are those edges of the current boundary that have not been tested yet to encode/decode the triangle externally adjacent to them; a control code is sent/read for each active edge.

The encoding algorithm starts from an arbitrary triangle of the mesh, whose edges define the initial polygon, and sends its three vertices. Then, it examines each edge e of the current polygon in turn and tries to enlarge the polygon by including the triangle σ externally adjacent to e. It performs the following actions on e:

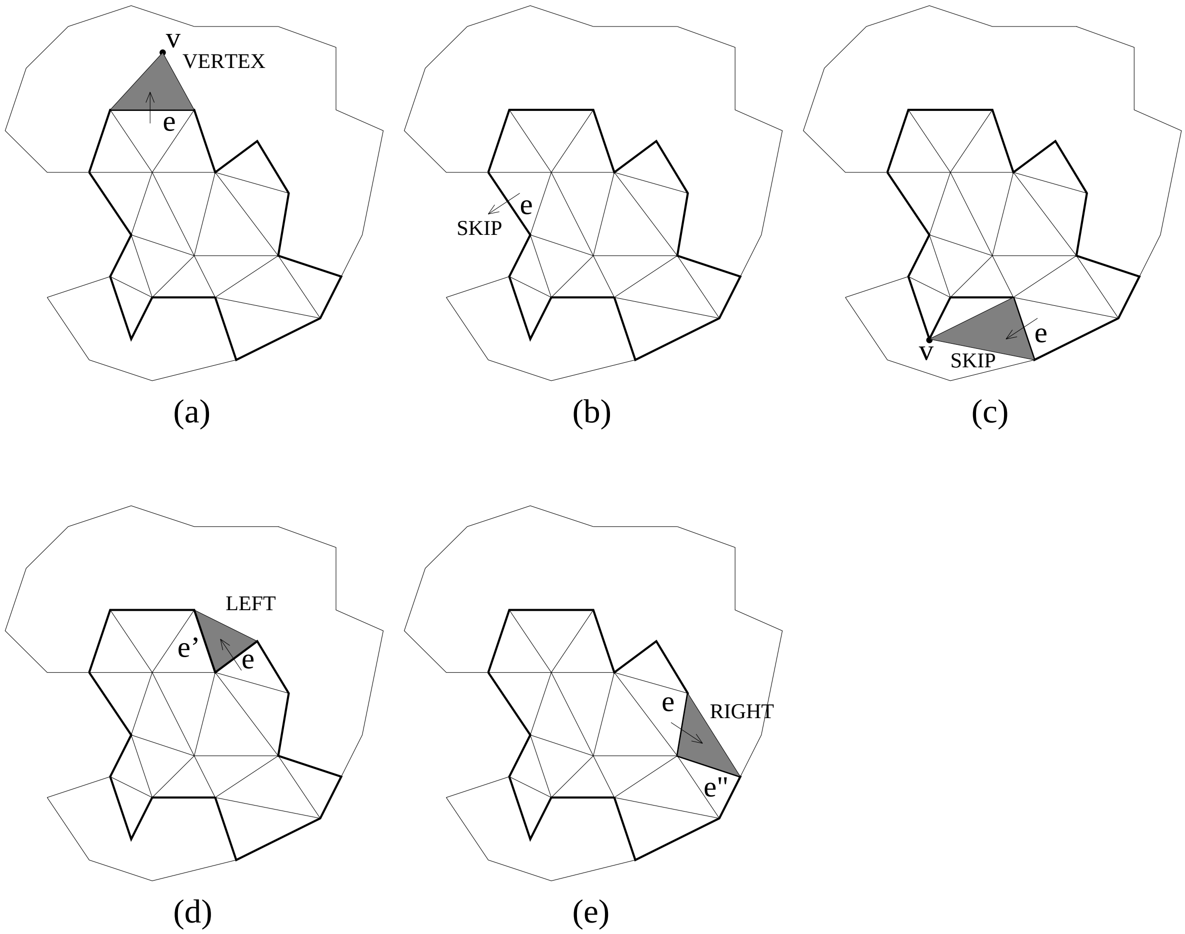


Figure 3.22: The mesh encoding algorithm of [DFMP98a] at work.

1. if the third vertex v of σ does not lie on the polygon then include σ and send a VERTEX code followed by vertex v (see Figure 3.22 (a));

2. if σ does not exist (i.e., edge e is on the boundary of the mesh), or the third vertex v of σ lies on the polygon, and it is not the other vertex of one of the edges adjacent to e along the polygon (i.e., σ cannot be included since it makes the polygon not simple), then send a SKIP code (see Figure 3.22 (b) and (c)); edge e will not be examined again;

3. if the third vertex v of σ lies on the polygon, and it is the other vertex of the edge adjacent to the left of e, then include σ and send a LEFT code (see Figure 3.22 (d));

4. if the third vertex v of σ lies on the polygon, and it is the other vertex of the edge adjacent to the right of e, then include σ and send a RIGHT code (see Figure 3.22 (e)).

The loop terminates when all the triangles have been encoded. In the case of a triangle mesh

Mesh	vert	tri	code bits	bits / vert	encoding time (tri/sec)	decoding time (tri/sec)
Marcy-U1	42943	85290	182674	4.2538	3.82 (22327)	2.33 (36605)
Marcy-U2	28510	56540	123086	4.3173	2.53 (22347)	1.55 (36477)
Marcy-U3	13057	25818	57316	4.3897	1.15 (22450)	0.71 (36363)
Marcy-U4	6221	12240	27180	4.3690	0.56 (21857)	0.33 (37090)
Marcy-A1	15389	30566	64678	4.2029	1.34 (22810)	0.83 (36826)
Marcy-A2	15233	30235	63958	4.1986	1.36 (22231)	0.82 (36871)
Marcy-A3	15515	30818	65210	4.2030	1.36 (22660)	0.84 (36688)
Marcy-A4	15624	31042	65520	4.1935	1.39 (22332)	0.85 (36520)
Shape	2562	5120	10478	4.0897	0.16 (32000)	0.15 (34133)
Fandisk	6475	12946	27298	4.2159	0.47 (27544)	0.41 (31575)

Table 3.1: Experimental results from the compression method based on shelling, run on extendably shellable triangle meshes. Meshes Marcy U1–4 are terrains at uniform accuracy, meshes Marcy A1–4 are terrains at variable accuracy, the remaining meshes represent three dimensional objects homeomorphic to a sphere. Times are in seconds.

whose carrier is a surface without boundary, the last triangle is encoded by sending either a LEFT or a RIGHT code.

Each vertex is sent exactly once, and a code is sent for every examined edge, i.e., *at most one* code for each mesh edge. Indeed, not all edges are examined: LEFT and RIGHT codes eliminate an edge each, which therefore does not generate a code; moreover, the edges of the boundary remaining after having encoded all the triangles do not generate any code. Thus, the number of control codes is always less than the number of edges in the given mesh, i.e., $3n$ (equivalent to $6n$ bits), where n is the number of vertices in the mesh. Experimental results shown in Table 3.1 give better compression rates, i.e., lower than $4.5n$ bits. Experiments have been performed on a Pentium Pro (200 Mhz, 64 Mb RAM).

The decoding algorithm works in a similar way. It builds the first triangle from the three vertices found at the beginning of the bitstream. Then, it examines each edge e of the current polygon in turn, and tries to enlarge the current polygon by creating a new triangle adjacent to e, according to the directions contained in the next control code read from the bitstream:

1. in case of a VERTEX code: read a vertex v from the bitstream, create a new triangle σ from e and v, and set an adjacency link between σ and the triangle adjacent to the other side of e;

2. in case of a SKIP code: no new triangle is created; edge e will not be examined again;

3. in case of a LEFT code: create a new triangle σ from e and the edge e' adjacent to the left of e along the polygon; set adjacency links between σ and the triangles adjacent to the other side of e and e';

Mesh	vert	tri	pat-ches	repeated vert(%)	code bits	bits / vert	encoding time(tri/sec)	decoding time(tri/sec)
eight	766	1536	6	198(25.8)	3856	5.0339	0.057(26947)	0.06(25600)
cow	3078	5804	25	356(11.4)	13984	4.5432	0.2(29020)	0.2(29020)
femur	3897	7798	5	124(3.18)	18894	4.8483	0.36(21661)	0.27(28881)
pieta	3475	6976	15	468(13.4)	17124	4.9278	0.34(20517)	0.26(26830)
skull	10952	22104	80	3242(29.6)	58150	5.3105	1.19(18574)	0.86(25702)
bunny	34834	69451	3	323(0.92)	146986	4.2196	2.38(29181)	2.23(31143)
phone	33204	66287	3	12(0.0)	149058	4.4891	2.94(22546)	2.2(30130)

Table 3.2: Experimental results from the compression method based on shelling, run on non-extendably shellable triangle meshes representing three dimensional objects.

Mesh	vert	tetra	pat-ches	repeated vert(%)	code bits	bits / vert	encoding time(tri/sec)	decoding time(tri/sec)
blunt-2.0	1012	5615	2	12	35184	34.766	0.3(18716)	0.3(18716)
blunt-1.5	1436	8130	1	0	50060	34.860	0.44(18477)	0.43(18906)
blunt-1.0	2278	13094	1	0	78792	34.588	0.71(18442)	0.69(18976)
sod32-4.0	2080	11391	1	0	71064	34.165	0.65(17524)	0.58(19639)
sod32-3.0	3065	17071	1	0	105992	34.581	0.98(17419)	0.87(19621)
sod32-2.0	5022	28506	1	0	175908	35.027	1.64(17381)	1.46(19524)

Table 3.3: Experimental results from the compression method based on shelling, run on tetrahedral meshes representing scalar fields.

4. in case of a RIGHT code: create a new triangle σ from e and the edge e' adjacent to the right of e along the polygon; set adjacency links between σ and the triangles adjacent to the other side of e and e';

In the case of a LEFT or RIGHT code, and a current polygon having just three edges, then the newly created triangle σ is the last triangle of a mesh without boundary: adjacency links are set at the three edges of σ.

Encoding non-extendably shellable meshes requires a small overhead due to replication of vertices shared by more than one patch. Experimental results show that, still in this case, no more than $5.5n$ bits are needed for the control codes, while the number of repeated vertices is highly variable depending on the mesh, but no more than 30% of the whole (see Table 3.2).

The same technique extends to higher dimensions. The number of necessary control codes increases, and so does the number of bits for each code. Four bits are needed to compress tetrahedral meshes. Moreover, tetrahedralizations are not extendably shellable; thus, they are cut into shellable parts by the encoding algorithm. Experimental results for tetrahedral meshes are shown in Table 3.3.

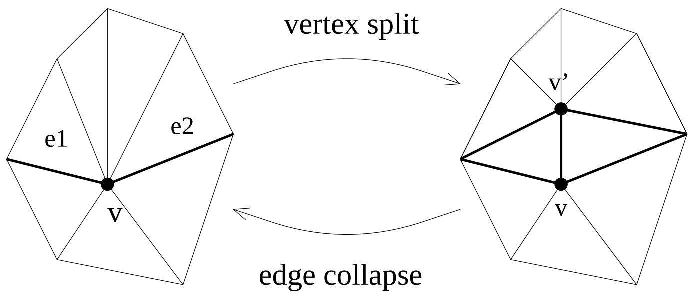


Figure 3.23: Edge collapse and its reverse vertex split.

The method is as simple as (generalized) triangle strips [Dee95, ESV96, Cho97], but it achieves higher compression rates, comparable to those of more complicated algorithms [TR96]. The algorithm in [TG98] is comparable for simplicity and produces more compact bitstreams. However, the compression rates in [TG98] are obtained by further compressing the command sequence generated by their basic encoding algorithm. A run-length encoding can be applied to the sequence generated by the shelling-based algorithm as well, since we have observed that typical bitstreams contain long sequences of VERTEX codes or of SKIP codes. We think that such improvement could lead to rates comparable to those of [TG98].

3.4.2 Progressive Methods

Progressive methods encode a coarse approximation of a mesh and a sequence of modifications which incrementally refine it into the original mesh [Hop96, SvK97, DFMP98a]. Their relation with evolutionary multiresolution models (reviewed in Sections 3.3.3, 3.3.4) is evident. An interrupted bitstream provides an approximation of the whole surface, which can be seen as a form of *lossy* compression. Progressive compression methods are especially useful for mesh transmission or as compact formats for secondary storage since they allow a trade-off between loading/transmission time and loss of accuracy. This additional feature is paid with compression rates that are usually lower than those of direct methods.

A sequence of progressive resolutions for a mesh is generated by iteratively applying a *destructive operator*, which removes details somewhere in the mesh. A destructive operator has an inverse *constructive operator*, which recovers such details. A compressed bitstream contains the coarse mesh (i.e., the result of a sequence of destructive operations) along with the sequence of constructive operations necessary to recover the original mesh.

A progressive compression method has been developed based on *Progressive Meshes* [Hop96, Hop98a] (see Section 3.3.3). Here, the destructive operator is *edge collapse*, which replaces an edge e with one of its endpoints v, and the two triangles sharing e with two edges incident at v. The corresponding constructive operator is *vertex split*, which expands a vertex v into an edge $e = vv'$ and two edges e_1 and e_2 among those incident at v into two triangles (see Figure

3.23). Each vertex split in the sequence is compactly encoded by providing the new vertex v', a reference to an old vertex v, and a code specifying the position of e_1 and e_2 around v. The number of bits necessary to encode connectivity is $n(\log n + \log(b(b-1)))$ for a mesh of n vertices, if b is the maximum degree of a vertex, since the position of e_1 is a number in the range $0 \dots b$ and that of e_2 a number in the range $0 \dots b-1$ (as one position has already been chosen for e_1).

Snoeyink and Van Kreveld [SvK97] have developed a progressive method which applies to triangle meshes in $\mathbb{E}^2$ based on Delaunay triangulations, and guarantees that all intermediate meshes are Delaunay triangulations. The destructive operator removes a vertex from a Delaunay triangulation; the corresponding constructive operator re-inserts such vertex. The encoding algorithm performs several stages starting from the given mesh: each stage removes a maximal set of independent vertices from the current Delaunay triangulation. The process is guaranteed to terminate in a logarithmic number of stages. The bitstream contains the final coarse mesh plus, for each stage in reverse order, the sequence of removed vertices (which must be re-inserted by the decoding algorithm). The sequence of vertices removed in a stage is sorted in the same order in which the triangle containing each vertex is encountered in the mesh traversal algorithm of de Berg et al. [dBvKvOO97]. Thus, the decoding algorithm at each stage can locate all the triangles containing the new vertices in linear time in the size of the current mesh. The length of the resulting bitstream is $O(\log^2 n)$ where n is the number of vertices and the constant is not known. A disadvantage of this method is in requiring heavy numerical calculations (i.e., the incremental reconstruction of a Delaunay triangulation) for decoding, which may cause not only time overhead, but also numerical and inconsistency problems (e.g., different triangulation of co-circular points in the original and in the reconstructed mesh). No experimental results are available.

Both methods above assume a specific strategy for retriangulating the mesh after the insertion of each vertex (i.e., vertex split in [Hop96], and Delaunay update in [SvK97]). In [DFMP98a], we have proposed a progressive method which does not depend on the retriangulation strategy. Such a method is not reviewed here because it is based on the multiresolution model proposed in Chapter 4; it will be described in Section 8.6.

3.5 Summary

In this chapter, we have reviewed the existing literature on three basic problems:

1. construction of an (approximated) cell complex representing a geometric object, starting from some available data for it;
2. multiresolution modeling of geometric objects based on cell complexes, and related operations: in particular, operations for extracting representations at a user-defined (variable) resolution and accuracy;

3. geometric compression of cell complexes: in particular, of triangle meshes.

Multiresolution modeling is an active research topic, and much work has been and is being done in the scientific community. However, from the heterogeneous approaches proposed in the literature it appears that a general framework for multiresolution cell complexes is still missing. By a framework, we mean an abstract notion of a multiresolution cell complex and of its properties, and an abstract definition of the relevant operations on a multiresolution complex, such as the extraction of a cell complex satisfying a certain error threshold function. A framework is necessary as a common platform for comparing different models based on the properties they have and on their efficiency in supporting the same operations.

In the current state-of-the-art, we note a generalized lack of a well-defined boundary between the conceptual and representation level. In most cases, the abstract definition of a multiresolution model is confused with the data structure used to represent it, and, consequently, the operations are confused with their implementation. Thus, it is difficult, if possible at all, to compare the behavior of different models because, for instance, the expressive power and/or the operations considered on each model may be similar but not exactly the same. Moreover, sometimes the algorithms proposed for some operations are heuristics, and different models may adopt different heuristics, thus preventing any meaningful comparison.

For instance, the various types of vertex hierarchies proposed differ for the way in which they track dependencies between vertices. Thus, even if built from the same sequence of edge collapses, they may provide different triangle meshes for the same extraction parameters because some model considers two vertex splits as dependent on each other, and some other model does not. It is not possible to tell which one is the correct solution because there is not a precise and generally accepted definition of what the solution should be.

Most existing models are designed for specific applications (e.g., for real-time visualization of surfaces in virtual reality, for terrain processing in GISs, etc.) and consequently they only consider operations relevant to that application field (e.g., obtaining meshes that fit certain view parameters in VR, spatial search in GISs, etc.).

Existing models usually rely on specific construction techniques (e.g., recursive quaternary subdivision for quadtrees, edge collapse for triangle meshes, vertex insertion or removal, etc.). Thus, representations at variable resolution which can be obtained from them are constrained to have certain characteristics which depend on the construction strategy.

Chapter 4

A Model for Multiresolution Cell Complexes

4.1 Introduction

Existing multiresolution models of cell complexes are all based on encoding information about how an initial complex can be progressively modified to increase/decrease its resolution. With the exception of layered models, modifications are *local*: they affect only a small part of the object, i.e., only a small subset of cells of the complex (see nested and evolutionary models in Section 3.3). Intuitively, smaller modifications give a wider range of possible ways to combine them into variable-resolution cell complexes, thus enhancing the *expressive power* of the model.

Let us consider a sequence of modifications performed on an initial cell complex. Figure 4.1 shows a sequence of two-dimensional cell complexes (triangle meshes) generated through incremental refinement, and the sequence of modifications corresponding to the process. The same sequence can be read in the opposite way as a sequence of triangle meshes generated through incremental coarsening.

Some modifications of the sequence are *independent* (i.e., no matter which one is performed first, because they affect distinct portions on the complex), while other modifications *depend on each other* (i.e., they must be performed in that order because the second one modifies some cells introduced by the first one). For instance, in Figure 4.1, modification 2 depends on modification 1, while modifications 2 and 3, and 1 and 3, are independent. Dependency relations define a *partial order* over the set of modifications, which can be represented as a DAG (see Figure 4.2). Any totally ordered sequence of modifications (a permutation of the original sequence) that is compatible with the partial order provides a way for progressively modifying the initial cell complex into the final one (see Figure 4.3). Although the final result is the same, the set of intermediate cell complexes of the process may be different; a variety of representations of the target geometric object at different (variable) resolutions and accuracies

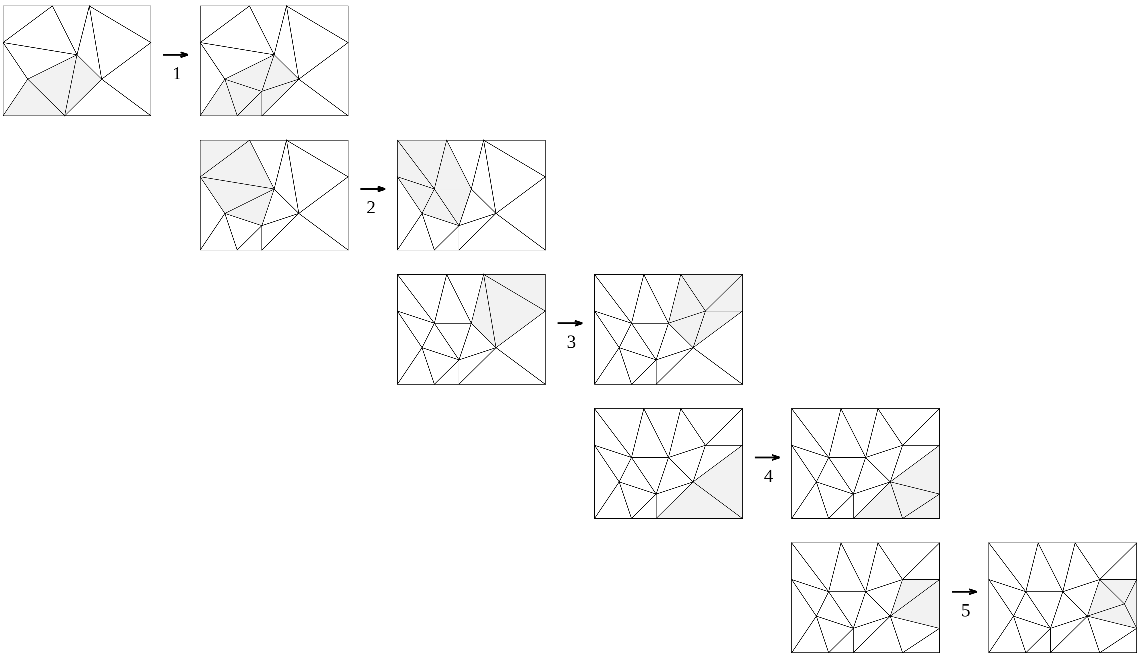


Figure 4.1: A sequence of modifications applied to a triangulation in $\mathbb{E}^2$.

can be obtained from the partial order by truncating a consistently sorted sequence of the modifications.

A partially ordered set of modifications is at the heart of any multiresolution model, and forms what we call a *Multiresolution Cell Complex*, or *Multi-Complex* (MC).

Most multiresolution models based on cell decomposition proposed in the literature can be interpreted as special cases of Multi-Complexes: every model is characterized by the rules used for generating the local modifications, by the data structures adopted to encode them, by the operations supported, and by the efficiency of accessing algorithms. Viewing the model as an MC helps understanding its expressive power independently of the specific data structures and algorithms.

In this chapter, we formally define the Multi-Complex, present its properties, and explain how it encompasses previous multiresolution models. Operations on MCs will be discussed in Chapter 5.

The Multi-Complex extends the Multiresolution Triangulation, or Multi-Triangulation, proposed in [Pup96, DFPM97, Pup98]; such models consider d-dimensional simplicial complexes embedded in $\mathbb{E}^d$ (e.g., representing the domain of scalar fields). Here, we deal with generic k-dimensional manifold cell complexes in $\mathbb{E}^d$ and we allow more general modifications, including refining boundaries, merging and splitting connected components.

This chapter is organized as follows:

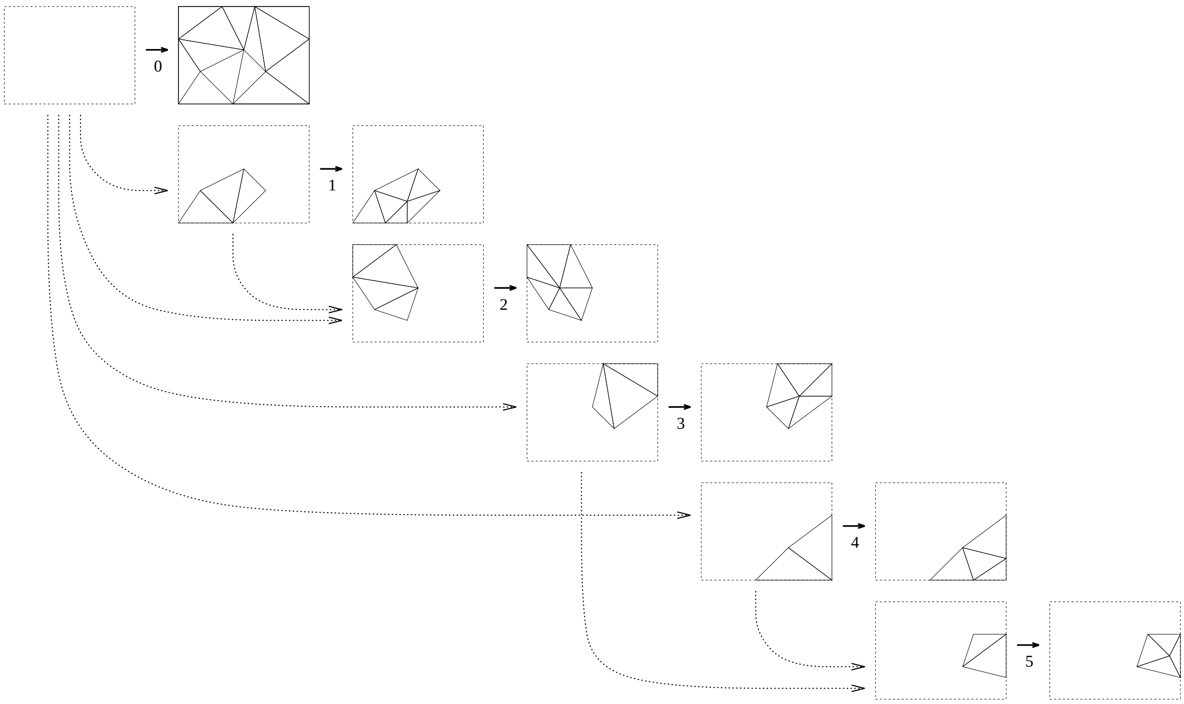


Figure 4.2: The partial order in the sequence of modifications of Figure 4.1. An extra modification representing the creation of the initial complex has been added; each modification has been isolated from its context, showing just the triangles taking part in the replacement; a dotted arrow from i to j means that j depends on i.

- In Section 4.2, we introduce the concept of *modification pattern*, an entity which describes a modification of a cell complex and which plays a fundamental role in the definition of the Multi-Complex.

- In Section 4.3, we define the *Multi-Complex* (MC) as a structure which organizes a partially ordered set of modification patterns in the form of a directed acyclic graph (DAG).

- In Section 4.4, we characterize the *cell complexes at variable resolutions* which can be obtained from a Multi-Complex; the number of such complexes determines the expressive power of an MC.

- In Section 4.5, we discuss *adjacency relations* between the cells involved in a Multi-Complex.

- In Section 4.6, we define some *quality parameters* for an MC which are relevant for the efficiency of the algorithms operating on it.

- Finally, in Section 4.7 we show how existing multiresolution models can be seen as *instances* of a Multi-Complex.

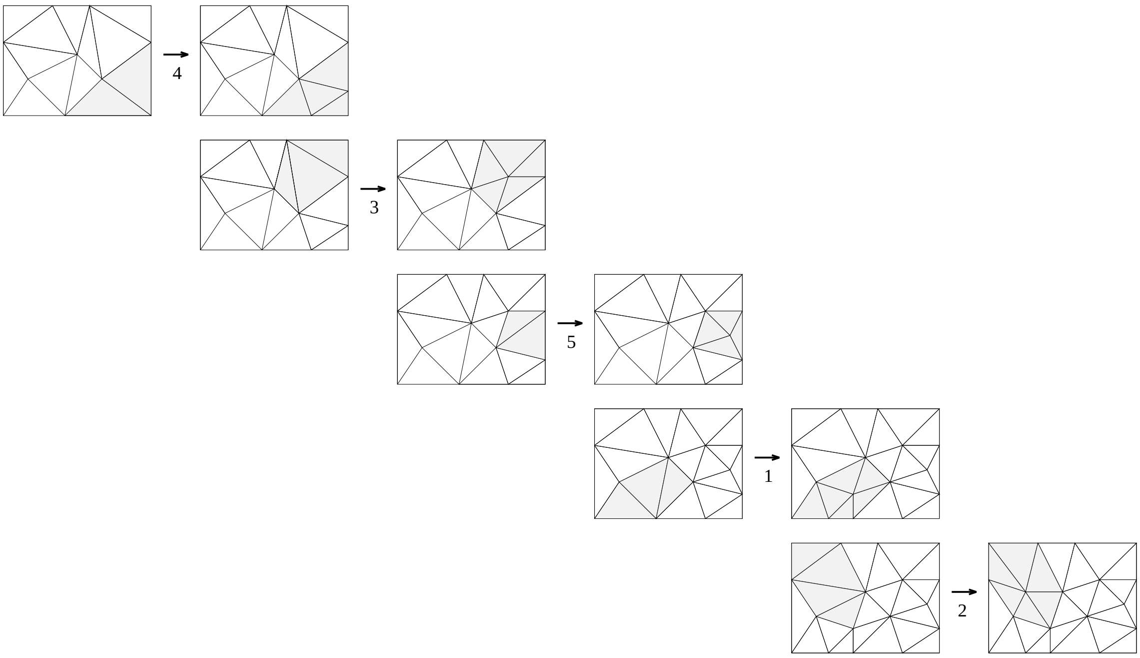


Figure 4.3: A permutation of the sequence in Figure 4.1, which is consistent with the partial order shown in Figure 4.2. The initial and final triangulations are the same, but the ones in intermediate stages are different from those of Figure 4.1.

4.2 Preliminaries

In Section 2.4.4, we have summarized the basic elements in a model for a geometric entity, according to the approach based on cell complexes. Such elements are:

- a *regular and manifold cell complex* Γ, where each top cell has an associated *approximation error* (and where, in general, each cell may have application-dependent *attributes* associated with it);
- possibly, a *function* f piecewise defined on the top cells of Γ.

Here, we are interested in modeling at multiple resolutions the cell complex Γ underlying the model of an entity, independently of the function f defined on its top cells, and on the attributes of cells.

We restrict our attention to regular k-dimensional cell complexes representing k-dimensional *manifolds* in $\mathbb{E}^d$. In particular, all cell complexes at variable resolution implicitly represented in our multiresolution model will be k-manifolds. This means that each of the modifications involved in the model will take a manifold complex and produce a manifold complex; changes in the topological type of the complex (e.g., creating and/or closing holes, merging or splitting

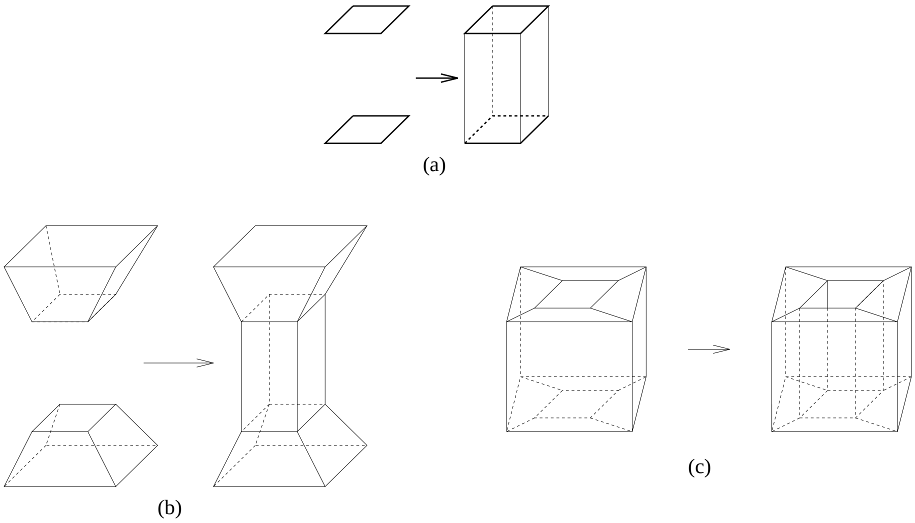


Figure 4.4: (a) A two-dimensional modification pattern $C = (\Gamma_{old}, \Gamma_{new})$: Γ_{old} consists of two squares and Γ_{new} consists of the four vertical faces of a square-based prism (thick lines denote boundary edges). (b) An example where the application of C merges two connected components, and (c) an example where its application creates a through hole.

connected components, etc., as in Figure 4.4 (b) and (c)), are allowed.

Definitions and properties will be given in the general case of k-dimensional cell complexes, and then used in the interesting cases of k-simplicial complexes, and, in particular, of triangle and tetrahedral meshes.

4.2.1 Modifications of a Cell Complex

A *modification* is an operation that replaces a cell complex Γ_{old} (possibly contained in a bigger complex Γ) with another cell complex Γ_{new}. The interesting case is when Γ_{old} and Γ_{new} are two approximations, at two different resolutions, of the same (portion of a) geometric object. A modification pattern specifies a modification as a pair of cell complexes:

Definition 4.1 *A k-dimensional* modification pattern *(or k-modification pattern, for short) in $\mathbb{E}^d$ is a pair $C = (\Gamma_{old}, \Gamma_{new})$, where Γ_{old} and Γ_{new} are regular and manifold k-dimensional cell complexes in $\mathbb{E}^d$.*

Intuitively, Γ_{old} is the complex that exists before performing the modification and Γ_{new} is the complex that replaces it after the modification. We use the symbols $\Pi_1(C)$ and $\Pi_2(C)$ to denote the first and second cell complex, respectively, of the pair forming a modification pattern C.

Applying a modification pattern $C = (\Gamma_{old}, \Gamma_{new})$ to a cell complex Γ consists of replacing Γ_{old} with Γ_{new} in Γ:

Definition 4.2 *Let Γ be a regular k-dimensional cell complex in $\mathbb{E}^d$, and $C = (\Gamma_{old}, \Gamma_{new})$ be a k-dimensional modification pattern in $\mathbb{E}^d$. The* result of applying C to Γ, *denoted with $\Gamma[C]$ or $\Gamma[\Gamma_{new}/\Gamma_{old}]$, is the set of cells $(\Gamma \setminus \Gamma_{old}) \cup \Gamma_{new}$.*

Set $\Gamma[C]$ is not always a cell complex. We are interested in applying C just to complexes Γ such that Γ contains Γ_{old}, $\Gamma[C]$ is cell complex, and C does not open or close holes in the boundary of Γ when trasforming it into $\Gamma[C]$. This corresponds to the following notion of *validity* of a modification pattern for a cell complex:

Definition 4.3 *Let Γ be a regular k-dimensional cell complex in $\mathbb{E}^d$, and $C = (\Gamma_{old}, \Gamma_{new})$ be a k-dimensional modification pattern in $\mathbb{E}^d$. C is a* valid *modification pattern for Γ if the following three conditions hold:*

1. $\Gamma_{old} \subseteq \Gamma$ *(as sets of cells);*
2. $\Delta(\Gamma_{new}) \cap \Delta(\Gamma \setminus \Gamma_{old}) \subseteq \Delta(\partial(\Gamma_{new}) \cap \partial(\Gamma_{old}))$*;*
3. $\partial(\Gamma_{old}) \subseteq \partial(\Gamma_{new}) \cup \partial(\Gamma)$ *(as sets of cells).*

Figure 4.5 shows some examples of valid and invalid modification patterns in the two-dimensional case:

- Modification pattern C_1 is valid for Γ. Note that the boundary complexes of $\Pi_1(C_1)$ and $\Pi_2(C_1)$ are the same; thus, $\Pi_2(C_1)$ fits the hole left after deleting $\Pi_1(C_1)$ from Γ.
- Modification pattern C_2 is valid for Γ. The deletion of $\Pi_1(C_2)$ leaves an "open" hole in Γ, where some cells of the boundary complex of Γ disappear. $\Pi_2(C_2)$ fits the existing border of the hole, and replaces the missing part with a new chain of edges.
- Modification pattern C_3 violates Conditions 2 and 3. Condition 2 is violated because if we look at triangle $\sigma = p_1p_2p_3$, we have that $\sigma \in (\Gamma \setminus \Pi_1(C_3))$ and the interior of σ intersects the interior of a triangle in $\Pi_2(C_3)$; such intersection contributes to the intersection of the domains of $\Pi_2(C_3)$ and $\Gamma \setminus \Pi_1(C_3)$, and it is two-dimensional; thus, it cannot be covered by the domain of the intersection of the boundary complexes of $\Pi_2(C_3)$ and $\Pi_1(C_3)$. Condition 3 is violated because edge $e = p_1p_2$ is in $\partial(\Pi_1(C_3))$ but not in $\partial(\Pi_2(C_3)) \cup \partial(\Gamma)$.
- Modification pattern C_4 violates Condition 3 because edge $e = p_1p_2$ is in $\partial(\Pi_1(C_4))$ but it is not in $\partial(\Pi_2(C_4)) \cup \partial(\Gamma)$.

- Modification pattern C_5 violates Condition 2 because edge $e = p_2p_3$ is in $\Pi_2(C_5) \cap (\Gamma \setminus \Pi_1(C_5))$ but not in $\partial(\Pi_1(C_5)) \cap \partial(\Pi_2(C_5))$.

- Modification pattern C_6 violates Condition 3 because edge p_0p_1 is in $\partial(\Pi_1(C_6))$ but not in $\partial(\Pi_2(C_6)) \cup \partial(\Gamma)$.

Figure 4.6 better illustrates the role of Condition 2 for a one-dimensional complex in $\mathbb{E}^2$: modification pattern C_2 violates Condition 2 because $\Delta(\Pi_2(C_2)) \cap \Delta(\Gamma \setminus \Pi_1(C_2))$ contains point c, which does not belong to $\Delta(\partial(\Pi_2(C_2)) \cap \partial(\Pi_1(C_2)))$; if we try to apply C_2 to Γ we do not get a cell complex, since some cells intersect in their interiors.

Condition 2 ensures that the set of cells remaining after having deleted Γ_{old} from Γ does not intersect Γ_{new} outside the common cells of the boundaries of Γ_{old} and Γ_{new}. This condition has two implications. First, it prevents that two cells of Γ_{new} and $\Gamma \setminus \Gamma_{old}$ may intersect outside a common face (as modification pattern C_3 in Figure 4.5), thus ensuring that the result of substituting Γ_{new} to Γ_{old} in Γ, i.e., $\Gamma[\Gamma_{new}/\Gamma_{old}]$, is a cell complex. Moreover, it ensures that replacing Γ_{old} with Γ_{new} in Γ does not glue separate parts of the boundary or close up holes that existed in Γ (as modification pattern C_5 in Figure 4.5 does).

Condition 3 requires that every cell of the boundary complex of Γ_{old}, which is internal to Γ, must appear in the boundary complex of Γ_{new} as well. In other words, the boundary of Γ_{new} must match with the boundary of the "hole" created in Γ when removing Γ_{old}. This condition ensures that replacing Γ_{new} to Γ_{old} in Γ does not open holes that did not exist before (as modification pattern C_6 in Figure 4.5 does; if we take into account just topology and not geometry, we can say that also C_3 and C_4 leave a hole in the triangulation).

Theorem 4.1 *The result $\Gamma[C]$ of applying a valid k-dimensional modification pattern C to a regular and manifold k-cell complex Γ is either empty or is a regular and manifold k-dimensional cell complex.*

Proof If $\Gamma \setminus \Gamma_{old} = \emptyset$, or $\Gamma_{new} = \emptyset$, then the theorem is trivially true. Otherwise, we show that Conditions 1 and 2 of the characterization of a cell complex (Property 2.1) are verified.

Condition 1 of Property 2.1 requires that the interiors of any two distinct cells γ_1, γ_2 of $\Gamma[C]$ are disjoint. The only non-trivial case is when one of the two cells (say γ_1) belongs to $\Gamma \setminus \Gamma_{old}$, and the other one (say γ_2) to Γ_{new}. The intersection of the interiors of γ_1 and γ_2 is contained in $\Delta(\Gamma_{new}) \cap \Delta(\Gamma \setminus \Gamma_{old})$, and thus, for Condition 2 of Definition 4.3, in $\Delta(\partial(\Gamma_{old}) \cap \partial(\Gamma_{new}))$. This may happen only if $\gamma_1 \in \partial(\Gamma_{old})$ and $\gamma_2 \in \partial(\Gamma_{new})$, and γ_1 and γ_2 coincide, violating the assumption that they are distinct.

Condition 2 of Property 2.1 requires that the boundary of any cell $\gamma \in \Gamma[C]$ is the union of the interiors of a subset of cells of $\Gamma[C]$. If $\gamma \in \Gamma_{new}$, then, trivially, all the faces of γ in Γ_{new} belong to $\Gamma[C]$ as well, and thus the condition is verified. Otherwise, $\gamma \in \Gamma \setminus \Gamma_{old}$; we show

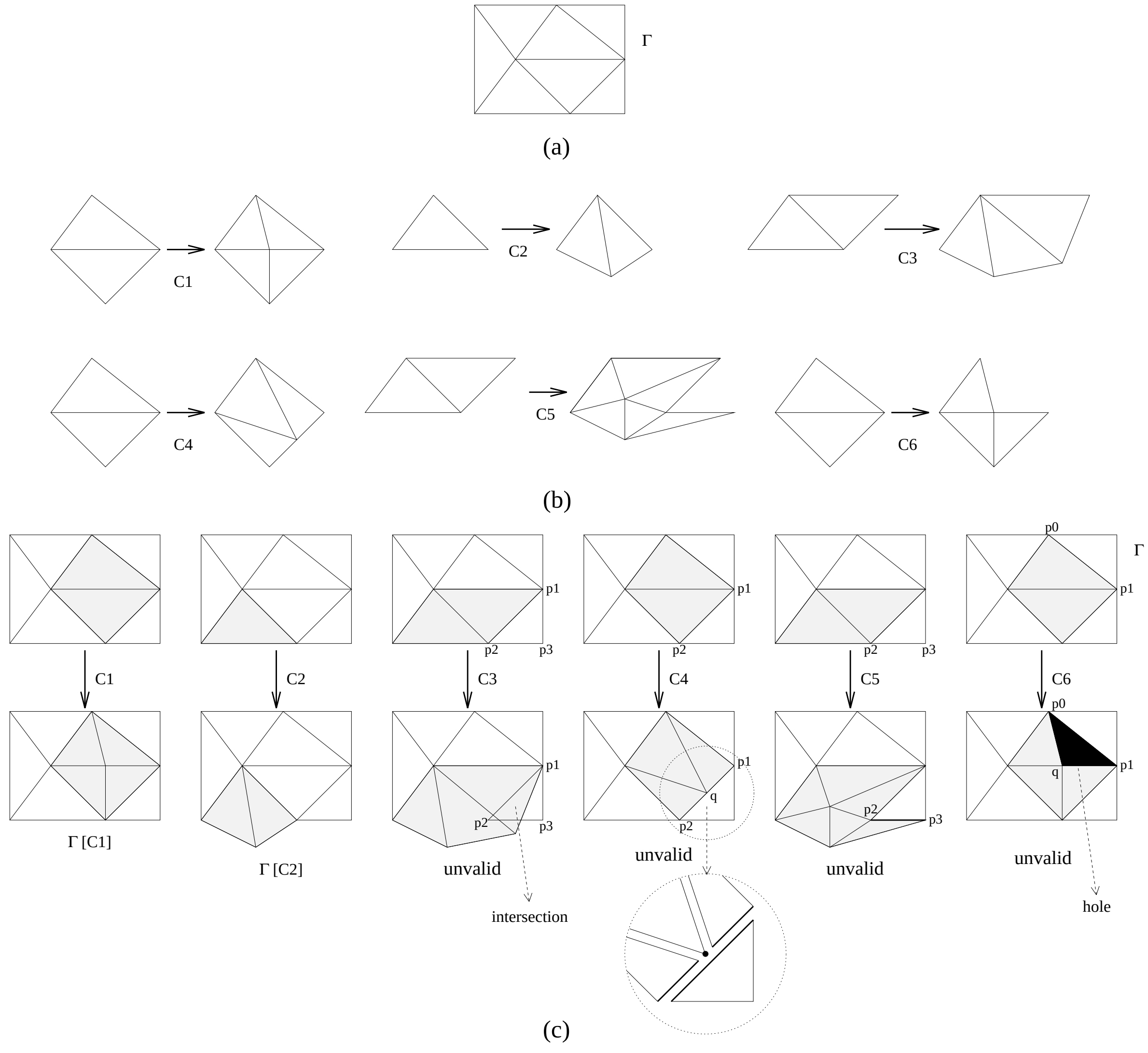


Figure 4.5: (a) A two-dimensional cell complex Γ; (b) six modification patterns, and (c) the result of applying each of them to Γ. Modification patterns C_1 and C_2 are valid for Γ; C_3 is not valid since it violates Condition 2 and 3; C_4 is not valid since it violates Condition 3; C_5 is not valid since it violates Condition 2; C_6 is not valid since it violates Condition 3.

that all the faces of γ in Γ are in $\Gamma[C]$ as well. The only non-trivial case is that of faces γ' such that $\gamma' \in \Gamma_{old}$. We note that such a face γ' must be in $\partial(\Gamma_{old})$. Moreover, for Condition 2 of Definition 4.3, γ and, thus, $\gamma' \subseteq \Delta(\partial(\Gamma_{old}) \cap \partial(\Gamma_{new}))$. Since $\gamma' \in \Gamma_{old}$, this can only happen if $\gamma' \in \Gamma_{new}$, therefore, γ' belongs to $\Gamma[C]$.

$\Gamma[C]$ is a regular k-cell complex since it is the union of two regular cell complexes of the same dimension.

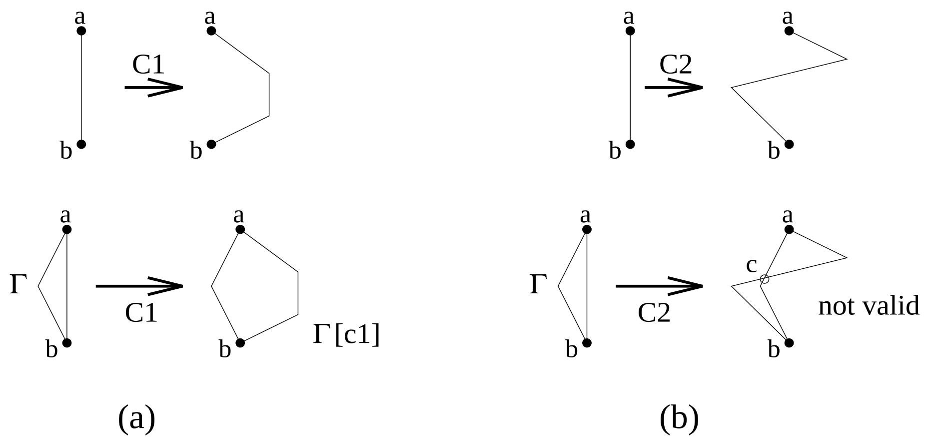


Figure 4.6: (a) A valid modification pattern C_1 for Γ, and the result of applying C_1 to Γ. (b) Modification pattern C_2 violates Condition 2 because $\Delta(\partial(\Pi_2(C_2)) \cap \partial(\Pi_1(C_2))) = \{a, b\}$ while $\Delta(\Pi_2(C_2)) \cap \Delta(\Gamma \setminus \Pi_1(C_2))$ contains, in addition, the intersection point c; the result of applying C_2 to Γ is not a cell complex.

Finally, $\Gamma[C]$ is a manifold since Γ and Γ_{new} are manifolds, and (for Condition 3 of Definition 4.3) Γ_{new} fits the hole left in Γ by the removal of Γ_{old}. $\square$

The following lemmas are introduced here since they will be useful later in the chapter:

Lemma 4.1 *Let Γ be a cell complex, and C be a valid modification pattern for Γ. Then, for every top-cell of $\Gamma[C]$ such that $\gamma \in \Gamma \setminus \Pi_1(C)$, the combinatorial boundary of γ is the same in Γ and in $\Gamma[C]$.*

The above lemma follows from the fact that the boundary cells of $\Pi_2(C)$ match with the boundary cells of the hole left by removing $\Pi_1(C)$.

Lemma 4.2 *Let Γ be a cell complex, and C be a valid modification pattern for Γ. Let $\gamma' \in \partial(\Pi_1(C)) \cap \partial(\Pi_2(C))$. Then, $\gamma' \in \partial(\Gamma)$ if and only if $\gamma' \in \partial(\Gamma[C])$.*

The above lemma follows from the fact that applying a valid modification pattern does not glue or separate parts of the boundary of a complex (e.g., open or close holes).

4.2.2 Properties of Modification Patterns

In this subsection, we define a number of interesting properties for a modification pattern.

A modification pattern is *minimal* if the modification it represents cannot be split in two modifications that can be performed independently:

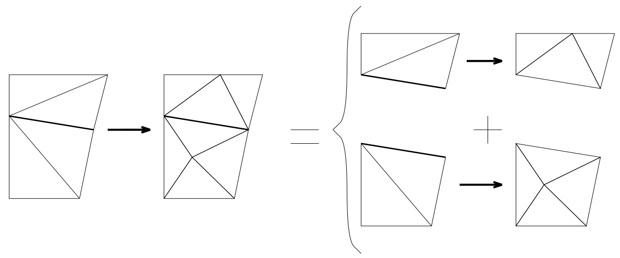


Figure 4.7: A non-minimal modification pattern and its decomposition into minimal patterns.

Definition 4.4 *A k-dimensional modification pattern $C = (\Gamma_{old}, \Gamma_{new})$ is* minimal *if there is no modification pattern $(\Gamma'_{old}, \Gamma'_{new})$, with ${\Gamma'_{old}}^{(k)} \neq \emptyset$ and ${\Gamma'_{old}}^{(k)} \subset {\Gamma_{old}}^{(k)}$, ${\Gamma'_{new}}^{(k)} \neq \emptyset$ and ${\Gamma'_{new}}^{(k)} \subset {\Gamma_{new}}^{(k)}$, such that $(\Gamma'_{old}, \Gamma'_{new})$ and $(\mathrm{sub}(\Gamma_{old} \setminus \Gamma'_{old}, \Gamma_{old}), \mathrm{sub}(\Gamma_{new} \setminus \Gamma'_{new}, \Gamma_{old}))$ are both valid k-modification patterns for Γ_{old}.*

Figure 4.7 shows a non-minimal modification pattern. Intuitively, C is not minimal if there is a set of $(k-1)$-cells that belong to both Γ_{old} and Γ_{new} and separate the k-cells of Γ_{old} and Γ_{new} into two non-empty subsets. A non-minimal modification pattern can be split into two or more minimal ones (as in the example of Figure 4.7).

A k-dimensional *refinement pattern* is a k-modification pattern where Γ_{new} contains more k-cells than Γ_{old}. Conversely, a k-dimensional *coarsening pattern* is a k-modification pattern where Γ_{new} has fewer k-cells than Γ_{old}. Intuitively, it is likely that the accuracy of Γ_{new} in representing the corresponding portion of an object is better than that of Γ_{old} in a refinement pattern, and worse in a coarsening one.

A modification pattern can be reversed:

Definition 4.5 *Let $C_1 = (\Gamma_{old}, \Gamma_{new})$ be a modification pattern. The* reverse *of C_1 is the modification pattern $C_2 = (\Gamma_{new}, \Gamma_{old})$.*

It is easy to see that, if C_1 is a modification pattern valid for a complex Γ_1, then its reverse C_2 is a modification pattern valid for $\Gamma_2 = \Gamma_1[C_1]$, and $\Gamma_2[C_2] = \Gamma_1$ (see Figure 4.8). The reverse of a refinement pattern is a coarsening pattern, and vice versa.

4.2.3 Sequences of Modification Patterns

Our purpose is applying several modification patterns to an initial complex. In this subsection, we introduce some fundamental properties of *sequences* of modification patterns.

A sequence of modification patterns is *valid* for a cell complex Γ_0 if every modification pattern C_i in the sequence is valid for the complex obtained as the result of successively applying to

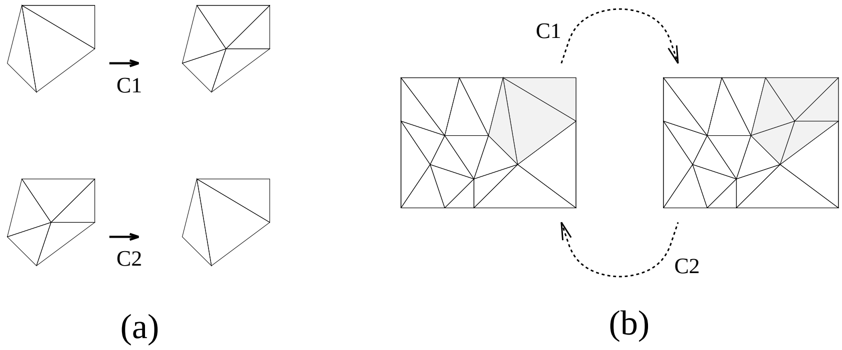


Figure 4.8: (a) Two mutually inverse modification patterns, and (b) the result of their application to a cell complex. C_1 is a refinement pattern; C_2 is a coarsening pattern.

Γ_0 all the patterns preceding C_i in the sequence. This corresponds to the following inductive definition:

Definition 4.6 *A sequence $C_1, C_2, \ldots, C_n$ is a* valid sequence *of modification patterns for a regular and manifold cell complex Γ_0 if*

- *$n = 1$, and C_1 is a valid modification pattern for Γ_0, or*
- *$n > 1$, the sequence $C_1, C_2, \ldots, C_{n-1}$ is a valid sequence of modification patterns for Γ_0, and C_n is a valid modification pattern for $\Gamma_0[C_1]\cdots[C_{n-1}]$.*

It is easy to see that, if $C_1, C_2, \ldots, C_{n-1}$ is a valid sequence of modification patterns for Γ_0, then $\Gamma_0[C_1]\cdots[C_n]$ is a regular and manifold cell complex (the proof reduces to repeated applications of Theorem 4.1).

We say that a modification pattern C' *directly depends on* another modification pattern C (and, thus, C *directly blocks* C') if and only if the effect of applying C' is removing some of the cells introduced by applying C:

Definition 4.7 *Let $C = (\Gamma_{old}, \Gamma_{new})$ and $C' = (\Gamma'_{old}, \Gamma'_{new})$ be two k-dimensional modification patterns. C'* directly depends on *C (and, thus, C* directly blocks *C') if ${\Gamma_{new}}^{(k)}$ and ${\Gamma'_{old}}^{(k)}$ have a non-empty intersection (as sets of cells).*

In Figure 4.9, modification pattern C_2 directly depends on C_1 since it replaces the shaded cells.

In a set of modification patterns, two modification patterns *depend* on each other if they are in the transitive closure of the relation of direct dependency; in a similar way, we define when one modification pattern *blocks* another modification pattern:

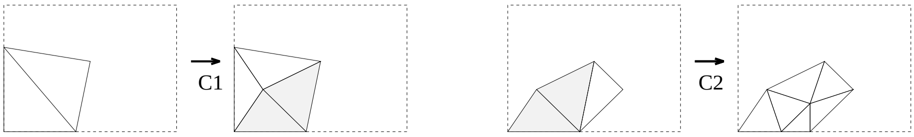


Figure 4.9: Modification pattern C_2 directly depends on C_1, and C_1 directly blocks C_2; the common triangles of $\Pi_2(C_1)$ and $\Pi_1(C_2)$ are shaded.

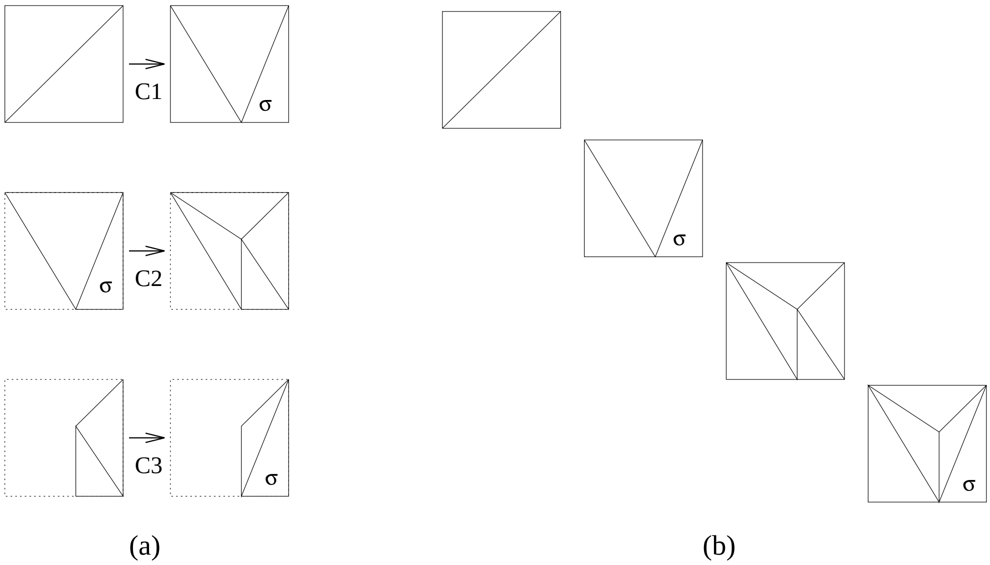


Figure 4.10: (a) Three modification patterns in two dimensions; (b) an example of their application in sequence C_1, C_2, C_3: triangle σ is created, destroyed and then created again. The relation of dependency in the set of modification patterns is not a partial order since C_3 depends on C_2 and, in turn, C_2 depends on C_3.

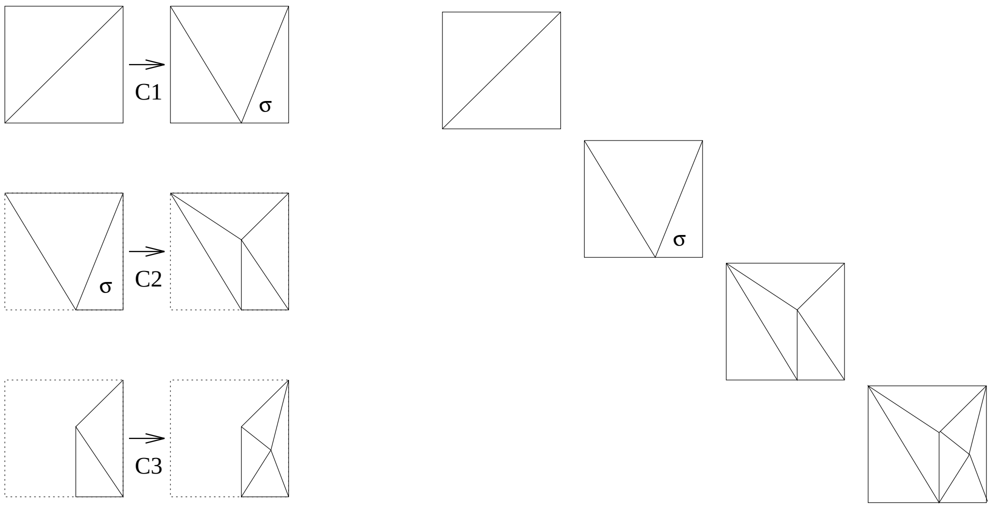
Definition 4.8 *Let $\mathcal{C}$ be a set of k-dimensional modification patterns, and let $C, C' \in \mathcal{C}$. C'* depends on *C (and C* blocks *C') if*

- *either C' directly depends on C,*
- *or there exists a modification pattern $C'' \in \mathcal{C}$, different from C and C', such that C' directly depends on C'', and C'' depends on C.*

Otherwise, we say that C and C' are mutually independent*.*

4.2.3.1 Non-Redundant Sequences

Let $C_1, C_2, \ldots, C_n$ be a valid sequence of modification patterns for a complex Γ_0; we note that, for every C_i, each k-cell $\gamma \in \Pi_1(C_i)$ either belongs to Γ_0 (and does not belong to any $\Pi_1(C_l)$ for



(a) (b)

Figure 4.11: (a) A sequence of three modification patterns in two dimensions, where each is non-redundant with respect to the previous ones; (b) an example of their application to a cell complex. Triangle σ is created and destroyed only once. The relation of dependency in the set of modification patterns is a partial order.

$l < i$), or belongs to $\Pi_2(C_j)$ for some $j < i$ (and does not belong to any $\Pi_1(C_l)$ for $j < l < i$). In other words, every cell removed by C_i must exist in the initial complex or have been created by a previous modification (and, in both cases, it must not have been removed before C_i). In a valid sequence a cell γ may be created and removed several times, and thus the dependency relation may be cyclic. In Figure 4.10, triangle σ is created twice (i.e., by applying C_1 and C_3), and modification pattern C_2 (whose application removes σ) depends on both C_1 and C_3, while C_3 depends on C_2, thus forming a cycle in the relation.

For the purpose of multiresolution, we are interested in sequences that monotonically increase (or decrease) the resolution of a cell complex. Thus, we introduce the notion of non-redundancy. A modification pattern is *non-redundant* with respect to a set of modification patterns if it does not recreate k-cells eliminated by some other modification pattern of the set:

Definition 4.9 *Let $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ be a set of k-modification patterns. A k-modification pattern $C = (\Gamma_{old}, \Gamma_{new}) \notin \mathcal{C}$ is* non-redundant *with respect to $\mathcal{C}$ if, for all $1 \leq i \leq n$, the set of k-cells of Γ_{new} does not intersect that of $\Pi_2(C_i)$.*

In Figure 4.10, C_3 is redundant with respect to set $\{C_1, C_2\}$ because $\Pi_2(C_3)^{(k)} \cap \Pi_1(C_1)^{(k)} = \{\sigma\} \neq \emptyset$. In Figure 4.11, C_3 is non-redundant with respect to set $\{C_1, C_2\}$.

A *non-redundant sequence* of modification patterns is a sequence $C_1, C_2, \ldots, C_n$ where, for each $i = 2 \ldots n$, C_i is non-redundant with respect to the set $\{C_j | 1 \leq j < i\}$.

It is easy to verify that, in a valid and non-redundant sequence, each k-cell either is in the initial complex, or there is exactly one modification pattern specifying its "creation"; and each cell either is in the final complex, or there is exactly one modification pattern specifying its "deletion":

Property 4.1 *Let Γ be a regular and manifold k-dimensional cell complex, and $C_1, C_2, \ldots, C_n$ be a non-redundant and valid sequence of modification patterns for Γ. Then, for every k-cell $\gamma \in \cup_{i=1}^{n}(\Pi_1(C_i) \cup \Pi_2(C_i)) \cup \Gamma$:*

- *either $\gamma \in \Gamma$ and $\gamma \notin \Pi_2(C_i)$ for all $i = 1 \ldots n$, or $\gamma \notin \Gamma$ and $\gamma \in \Pi_2(C_i)$ for exactly one $1 \leq i \leq n$;*
- *either $\gamma \in \Gamma[C_1] \cdots [C_n]$ and $\gamma \notin \Pi_1(C_i)$ for all $i = 1 \ldots n$, or $\gamma \notin \Gamma[C_1] \cdots [C_n]$ and $\gamma \in \Pi_1(C_i)$ for exactly one $1 \leq i \leq n$.*

In order to define our multiresolution model, we are only interested in sequences of modification patterns $C_1, C_2, \ldots, C_n$, valid for a cell complex Γ_0, and non-redundant.

Such condition also implies that the relation of dependency in set $\{C_j | 1 \leq j < n\} \cup \{(\emptyset, \Gamma_0)\}$ is not cyclic, i.e., it defines a partial order:

Theorem 4.2 *Let $C_1, C_2, \ldots, C_n$ be a non-redundant sequence of modification pattern, valid for a cell complex Γ_0. Then, the relation of dependency in set $\{C_j | 1 \leq j \leq n\} \cup \{(\emptyset, \Gamma_0)\}$ is not cyclic.*

Proof Let $C_0 = (\emptyset, \Gamma_0)$; since $\Pi_1(C_0)$ is empty, then sequence $C_0, C_1, \ldots C_n$ is non-redundant.

We consider the set of k-cells $\Gamma^{(k)} = \{\gamma | \gamma \in \Pi_1(C_i) | 0 \leq j < n\} \cup \{\gamma | \gamma \in \Pi_2(C_i) | 0 \leq j < n\}$. From Property 4.1, every cell $\gamma \in \Gamma^{(k)}$ belongs to $\Pi_2(C_j)$ for *exactly one* j, and γ belongs to $\Pi_1(C_i)$ for *at most one* i; moreover, if such i exists, then $i > j$.

Thus, every cell γ may only cause a modification pattern C_i (the one such that $\gamma \in \Pi_1(C_i)$, if it exists) to directly depend on a modification pattern C_j (the one such that $\gamma \in \Pi_2(C_j)$), with $j < i$. Therefore, cycles are not possible. □

4.2.3.2 Conflict-Free Sequences

If two modification patterns are independent, this does not mean that they can be applied to the same cell complex without interfering. Thus, we introduce the notion of *conflict*:

Definition 4.10 *Two k-dimensional modification patterns C_i and C_j are* not in conflict *if, for all Γ_i, Γ_j, with $\Gamma_i \in \{\Pi_1(C_i), \Pi_2(C_i)\}$ and $\Gamma_j \in \{\Pi_1(C_j), \Pi_2(C_j)\}$, we have that $\Delta(\Gamma_i) \cap$*

$\Delta(\Gamma_j) = \Delta(\partial(\Pi_1(C_i)) \cap \partial(\Pi_2(C_i)) \cap \partial(\Pi_1(C_j)) \cap \partial(\Pi_2(C_j)))$. *Otherwise, C_i and C_j are* in conflict.

The condition of non-conflictness guarantees that C_i and C_j can be applied in any order to a cell complex Γ for which they are both valid (see Figure 4.12).

Intuitively, C_i and C_j are not in conflict when all the pairs of complexes $\Pi_1(C_i)$ and $\Pi_1(C_j)$, $\Pi_1(C_i)$ and $\Pi_2(C_j)$, $\Pi_2(C_i)$ and $\Pi_1(C_j)$, and $\Pi_2(C_i)$ and $\Pi_2(C_j)$ intersect at most in a subset of their boundary cells, which are preserved in the modifications represented by C_i and C_j. This means that any of such pairs share the same subset of boundary cells, and the union of all their cells forms a complex.

We say that a set of modification patterns is *conflict-free* if there are no conflicts between pairs modification patterns that are independent:

Definition 4.11 *Let $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$ be a set of k-modification patterns. Set $\mathcal{C}$ is* conflict-free *if, for any pair of modification patterns $C_i, C_j \in \mathcal{C}$, which are mutually independent, C_i and C_j are not in conflict. Otherwise, set $\mathcal{C}$ is* conflicting.

Intuitively, this means that, for every subset $\mathcal{C}'$ of a conflict-free set $\mathcal{C}$, formed by modification patterns that are mutually independent, the set of cells obtained by taking either $\Pi_1(C_i)$ or $\Pi_2(C)$, for each $C_i \in \mathcal{C}'$, forms a cell complex.

We say that a *sequence* $C_1, C_2, \ldots, C_n$ of modification patterns is *conflict free* [*conflicting*] if the corresponding set $\{C_1, C_2, \ldots, C_n\}$ is conflict free [conflicting].

4.3 The Multi-Complex

In an MC, a *partially ordered set of modification patterns* of a cell complex is encoded in the form of a *directed acyclic graph* (DAG), where nodes represent modification patterns, and arcs represent relations of direct dependency between them. An MC is defined by abstracting the relation of direct dependency from a *valid*, *non-redundant*, and *conflict-free* sequence of *refinement* patterns. The non-redundancy of the sequence ensures that the direct dependency relation is a partial order. Since all nodes are refinement patterns, resolution monotonically increases while traversing the DAG according to the direction of its arcs. The validity and absence of conflicts ensure that, for every subset of modification patterns "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 cell complex.

Definition 4.12 *Let Γ_0 be a k-dimensional cell complex, $C_1, C_2, \ldots, C_n$ be a sequence of k-dimensional modification patterns such that:*

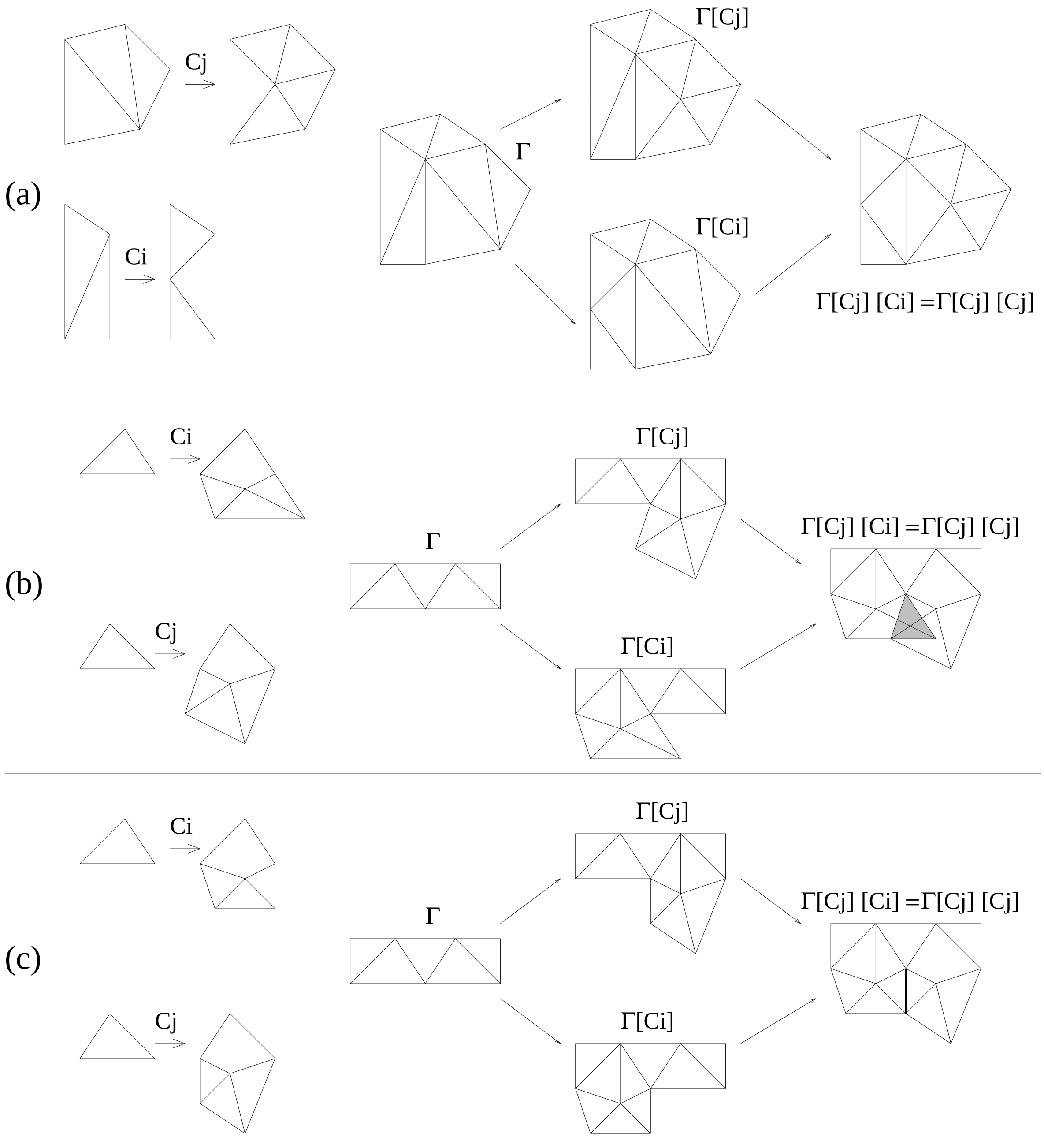


Figure 4.12: (a) Modification patterns C_i and C_j are not in conflict. (b) C_i and C_j are in conflict because the domains of $\Pi_2(C_i)$ and $\Pi_2(C_j)$ intersect outside their boundaries (i.e., in the shaded region). (c) C_i and C_j are in conflict because the domains of $\Pi_2(C_i)$ and $\Pi_2(C_j)$ intersect outside the common boundary cells of $\Pi_1(C_i)$ and $\Pi_1(C_j)$ (i.e., in the thick segment). In (b) and (c), note that one of the two modification patterns is no more valid for Γ after the other one has been applied.

1. *for every* $1 \leq i \leq n$, C_i *is a refinement pattern, it is minimal, and* $\Pi_1(C_i) \neq \emptyset$*;*

2. $C_1, C_2, \ldots, C_n$ *is a valid sequence for* Γ_0*;*

3. *sequence* $C_0, C_1, C_2, \ldots, C_n$ *is non-redundant, where* $C_0 = (\emptyset, \Gamma_0)$*;*

4. *sequence* $C_1, \ldots, C_n$ *is conflict-free.*

Then, a Multi-Complex *(MC) is a directed graph* $\mathcal{M}$ *where:*

- $\mathcal{M}$ *has n nodes, which are in one-to-one correspondence with the modification patterns of set* $\{C_0, \ldots, C_n\}$*;*
- *there is an arc* (C_i, C_j) *if and only if* C_j *directly depends on* C_i*.*

For a DAG, we use the term *leaf* to indicate a node with no outgoing arcs, and *root* for a node with no incoming arcs. We also say, for each arc (C_i, C_j), that node C_i is a *parent* of C_j, and C_j is a *child* of C_i; consequently, we also use the terms *ancestor* and *descendant*.

Nodes of an MC are special cases of nodes in a DAG, since they represent modification patterns. In the reminder of the thesis, call them *MC-nodes* whenever an ambiguity may arise.

We call the (k-dimensional) cells of a Multi-Complex $\mathcal{M}$ the set of (k-dimensional) cells γ such that γ belongs to $\Pi_2(C)$ for some node C in $\mathcal{M}$.

A *simplicial* Multi-Complex is a special case of MC where each modification pattern consists of a pair of simplicial complexes. The MC of Figure 4.13 (a) is a simplicial MC.

It is easy to see that the following property holds:

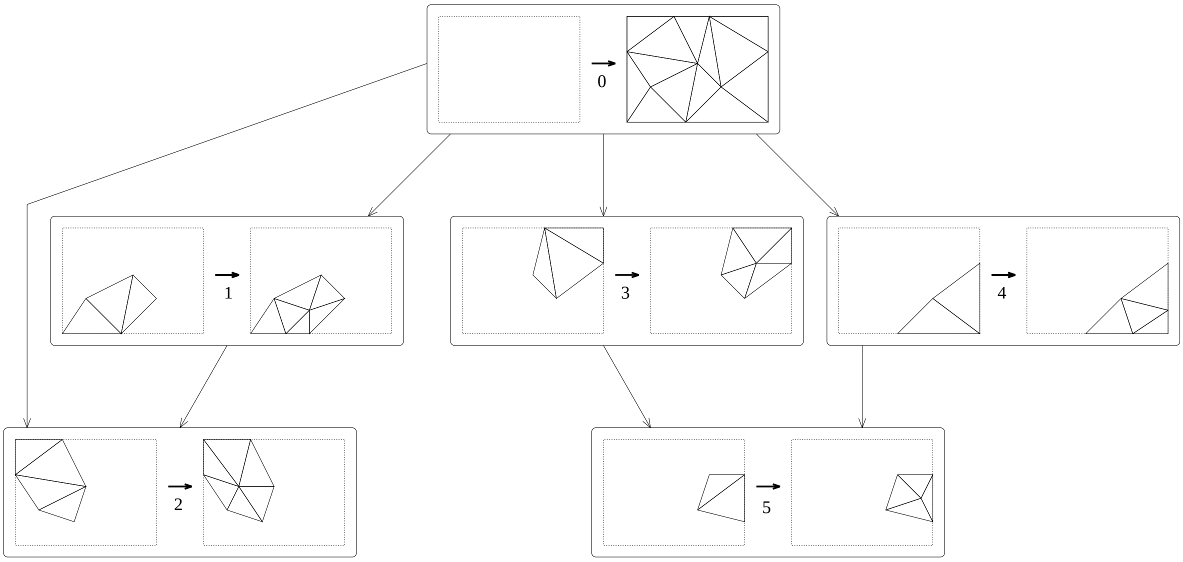
Property 4.2 *An MC* $\mathcal{M}$ *is a directed acyclic graph (DAG), node* C_0 *has no incoming arcs, and every other node is reachable from* C_0 *through a directed path.*

Node C_0 will be called the *root* of $\mathcal{M}$. Intuitively, the modification pattern specified in the root creates an initial coarse cell complex.

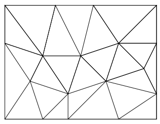
The following definition captures all the possible sequences of nodes of an MC which extend the partial order represented in the DAG to a total order:

Definition 4.13 *A permutation* $C_{i1}, C_{i2}, \ldots, C_{in}$ *of the nodes of an MC is* consistent *if, for every* $1 \leq j \leq n$*, all modification patterns* C_{ip} *that block* C_{ij} *have a position* $p < j$*.*

It is easy to see that, in an MC $\mathcal{M}$ built as in Definition 4.12, the given sequence $C_0, C_1, \ldots, C_n$ is a consistent permutation.



(a)



(b)

Figure 4.13: (a) A Multi-Complex, corresponding to the sequence of modification patterns of Figure 4.1; (b) its front cell complex.

In the following theorem, we show that any consistent permutation (i.e., any totally ordered sequence of the nodes of an MC consistent with the partial order encoded in the DAG) provides a valid sequence of modification patterns for the initial cell complex represented in the root. All such sequences produce the same cell complex as their result, independently of the specific order. This fundamental property permits to "forget" the initial sequence, since all consistent permutations are equivalent in this sense:

Theorem 4.3 *Let $\mathcal{M}$ be a Multi-Complex built from a cell complex Γ_0 and a sequence $\mathcal{S} \equiv C_1, C_2, \ldots, C_n$ of modification patterns, satisfying Conditions 1, 2, 3, and 4 of Definition 4.12. Then, for every consistent permutation $\mathcal{S}' \equiv C_{i1}, C_{i2}, \ldots, C_{in}$ of the nodes of $\mathcal{M}$:*

1. *$C_{i1}, C_{i2}, \ldots, C_{in}$ is a valid sequence for Γ_0;*

2. *$\Gamma_0[C_{i1}][C_{i2}]\cdots[C_{in}] = \Gamma_0[C_1][C_2]\cdots[C_n]$.*

Proof For convenience, we set $\Gamma'_j \equiv \Gamma_0[C_{i1}][C_{i2}]\cdots[C_{i(j-1)}]$, and $\Gamma_j = \Gamma_0[C_1][C_2]\cdots[C_{ij-1}]$, for $1 \leq j \leq n$. We prove Facts 1 and 2 of the theorem separately.

Proof of Fact 1 According to Definition 4.3, we have to show that, for every $1 \leq j \leq n$:

a. $\Pi_1(C_{ij}) \subseteq \Gamma'_j$;

b. $\Delta(\Pi_2(C_{ij})) \cap \Delta(\Gamma'_j \setminus \Pi_1(C_{ij})) \subseteq \Delta(\partial(\Pi_2(C_{ij})) \cap \partial(\Pi_1(C_{ij})))$;

c. $\partial(\Pi_1(C_{ij})) \subseteq \partial(\Pi_2(C_{ij})) \cup \partial(\Gamma'_j)$.

We prove each of the three Facts separately:

a. We first show that, every k-cell $\gamma \in \Pi_1(C_{ij})$ is in Γ'_j as well. For the non-redundancy and validity of sequence $\mathcal{S}$, we have that $\gamma \in \Pi_2(C_{ip})$ for exactly one ancestor C_{ip} of C_{ij} (note that C_{ip} may be root), and $\gamma \notin \Pi_1(C_{iq})$ for every $q \neq j$. Since sequence $\mathcal{S}'$ is consistent, $p < j$; thus, γ must be in Γ'_j.

   Now, we show that Γ'_j contains also the lower-dimensional cells of $\Pi_1(C_{ij})$. Let γ be a lower-dimensional cell of $\Pi_1(C_{ij})$. Let us consider any node C_{ip}, with $p < j$, such that $\gamma \in \Pi_1(C_{ip})$ and/or $\gamma \in \Pi_2(C_{ip})$.

   If C_{ip} and C_{ij} are independent then, since $\mathcal{S}$ is conflict-free, γ must be in $\Pi_1(C_{ip}) \cap \Pi_2(C_{ip})$, and, thus, its presence in Γ'_j is not influenced by having applied C_{ip} or not.

   Otherwise, C_{ip} must block C_{ij} (since sequence $\mathcal{S}'$ is consistent). Since sequence $\mathcal{S}$ is consistent, C_{ip} must be before C_{ij} in $\mathcal{S}$ as well. Therefore, C_{ip} has been applied both in constructing Γ_j and Γ'_j. It remains to show that, because $\mathcal{S}$ is a consistent sequence, the order in which the ancestors C_{ip} of C_{ij} that contain γ are applied has no influence in determining whether γ is in Γ'_j or not. Such nodes C_{ip} are of three types: (i) the ones such that $\gamma \in \Pi_1(C_{ip}) \cap \Pi_2(C_{ip})$; (ii) the ones such that $\gamma \in \Pi_1(C_{ip}) \setminus \Pi_2(C_{ip})$; (iii) the ones such that $\gamma \in \Pi_2(C_{ip}) \setminus \Pi_1(C_{ip})$. Modification patterns of type (i) do not disturb. A modification pattern of type (ii) can only be applied after one of type (iii) has been applied, and conversely. Therefore, since the initial complex is the same, the alternance of nodes of types (ii) and (iii) must be the same in $\mathcal{S}$ and in $\mathcal{S}'$; thus $\gamma \in \Gamma'_j$ if and only if $\gamma \in \Gamma_j$, and, since $\gamma \in \Gamma_j$, then $\gamma \in \Gamma'_j$.

b. We prove Fact b by showing that, for every pair of k-cells $\gamma \in \Pi_2(C_{ij})$ and $\gamma' \in \Gamma'_j \setminus \Pi_1(C_{ij})$, $\gamma \cap \gamma'$ is either empty, or it covers exactly a cell $\gamma'' \in \partial(\Pi_1(C_{ij})) \cap \partial(\Pi_2(C_{ij}))$.

   For the non-redundancy of $\mathcal{S}$, $\gamma' \in \Pi_2(C_{ip})$ for exactly one node C_{ip} in the DAG, different from C_{ij}. Since $\gamma' \in \Gamma'_j$, C_{ip} must appear before C_{ij} in sequence $\mathcal{S}'$.

   If C_{ip} and C_{ij} are independent, then, since $\mathcal{S}$ is conflict-free, $\gamma \cap \gamma'$ either is empty, or it covers exactly a cell γ'' in $\partial(\Pi_1(C_{ij})) \cap \partial(\Pi_2(C_{ij}))$.

Otherwise, C_{ip} must block C_{ij} (since $\mathcal{S}'$ is a consistent sequence). Thus, C_{ip} is before C_{ij} in $\mathcal{S}$. Therefore, $\gamma' \in \Gamma_j \setminus \Pi_1(C_{ij})$, and $\gamma \cap \gamma'$ either is empty, or it covers exactly a cell γ'' in $\partial(\Pi_2(C_{ij}))) \cap \partial(\Pi_1(C_{ij}))$.

c. Fact c reduces to showing that, for every $(k-1)$-cell $\gamma \in \partial(\Pi_1(C_{ij})) \setminus \partial(\Pi_2(C_{ij}))$, then $\gamma \in \partial(\Gamma'_j)$.

   For all C_{ip} that are independent of C_{ij}, γ cannot be in $\Pi_1(C_{ip})$ or in $\Pi_2(C_{ip})$, because this would be a conflict. Thus, all nodes C_{ip} that contain γ must be related with C_{ij} and, thus, be either ancestors or descendants of C_{ij}. Therefore, the subset of such nodes C_{ip} that are before C_{ij} is the same in $\mathcal{S}'$ and in $\mathcal{S}$. Since $\mathcal{S}$ is a valid sequence, for Lemma 4.2 we have that the "state" of γ (i.e., its being on the boundary or in the interior of the current complex) is not changed by any ancestor C_{ip} of C_{ij}. Therefore, the state of γ is the same in Γ_j and in Γ'_j; since we know that $\gamma \in \partial(\Gamma_j)$, then γ is in $\partial(\Gamma'_j)$ as well.

This concludes the proof of Fact 1. $\square$

Proof of Fact 2 Let $\Gamma \equiv \Gamma_0[C_1 \ldots C_n]$ and $\Gamma' \equiv \Gamma_0[C_{i1} \ldots C_{in}]$. For Lemma 4.1, it is sufficient to prove that the top cells of Γ and Γ' are the same, since applying a valid modification pattern preserves their combinatorial boundary.

For the non-redundancy of sequence $\mathcal{S}$, the set of top cells of Γ is $\cup_{i=0}^{n}\Pi_2(C_i) \setminus \cup_{i=0}^{n}\Pi_1(C_i)$; for the same reason, the the set of top cells of Γ' is $\cup_{j=0}^{n}\Pi_2(C_{ij}) \setminus \cup_{i=0}^{n}\Pi_1(C_{ij})$. Since $\mathcal{S}'$ is a permutation of $\mathcal{S}$, $\{C_i | 0 \leq i \leq n\} = \{C_{ij} | 0 \leq j \leq n\}$, and thus the two sets of cells are coincident. $\square$

We define the *front cell complex* of an MC as the result of applying all the modification patterns represented in its nodes to the initial cell complex stored at the root:

Definition 4.14 *Let $\mathcal{M}$ be a Multi-Complex, and $C_0, C_1, \ldots, C_n$ be a consistent sequence of its nodes (including the root C_0). The* front cell complex *of $\mathcal{M}$ is*

$$\text{front}(\mathcal{M}) \equiv \Pi_2(C_0)[C_1] \cdots [C_n].$$

The front cell complex is uniquely defined because, for Theorem 4.3, it is independent on the specific permutation considered. The front cell complex of an MC is a regular and manifold cell complex, still for Theorem 4.3 (see Figure 4.13 (b)).

4.3.1 Sub-MCs

A *sub-MC* of an MC $\mathcal{M}$ is a subgraph $\mathcal{M}'$ of $\mathcal{M}$ containing the root, and *closed* with respect to the partial order:

Definition 4.15 *Let $\mathcal{M}$ be a k-dimensional Multi-Complex in $\mathbb{E}^d$. A subgraph $\mathcal{M}'$ of $\mathcal{M}$ is a* sub-MC *of $\mathcal{M}$ if*

1. *$\mathcal{M}'$ contains the root C_0 of $\mathcal{M}$, and*
2. *(closure) for every node $n \in \mathcal{M}'$, all parents of n are in $\mathcal{M}'$ as well.*

Figure 4.14 (a) shows a sub-MC.

Theorem 4.4 *Any sub-MC $\mathcal{M}'$ of an MC $\mathcal{M}$ is a Multi-Complex.*

Proof It is immediate to see that the arcs of $\mathcal{M}'$ represent the relation of direct dependency among its nodes. The only point to show is that there exists a sequence of the nodes of $\mathcal{M}'$ which is valid for $\Gamma_0 \equiv \Pi_2(C_0)$.

Since the set of nodes of $\mathcal{M}'$ is closed with respect to the partial order, there exists a consistent permutation $\mathcal{S}'$ of the nodes of $\mathcal{M}$, such that all nodes of $\mathcal{M}'$ are at the beginning of $\mathcal{S}'$. If we truncate $\mathcal{S}'$ just before the first node not in $\mathcal{M}'$, we obtain a sequence of the nodes of $\mathcal{M}'$, which is valid for Γ_0 because of Theorem 4.3. $\square$

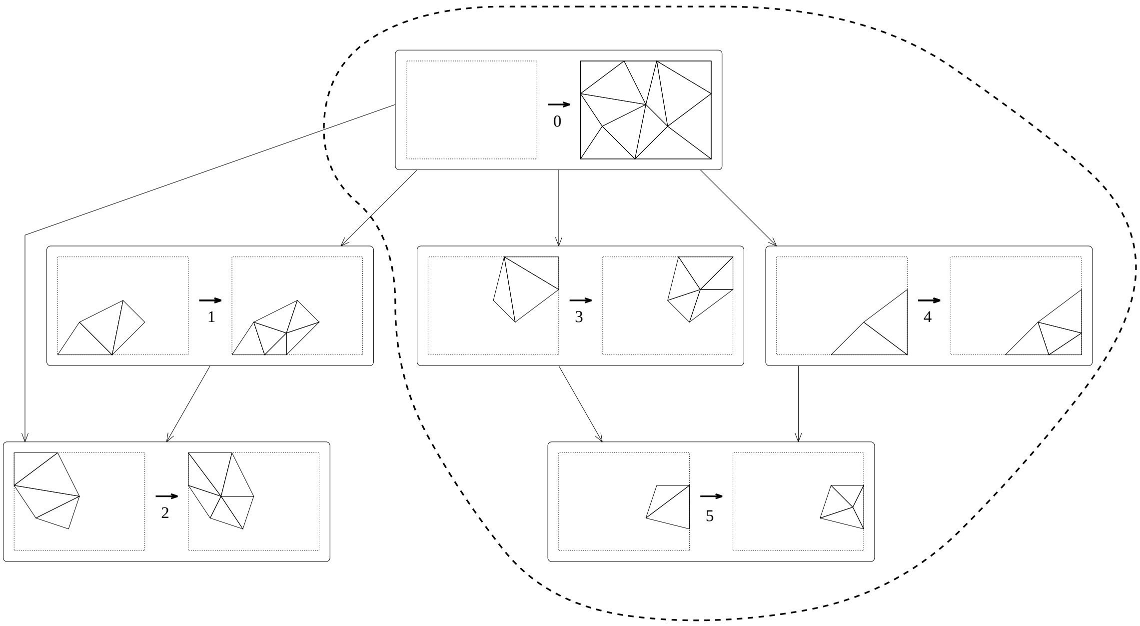
A sub-MC $\mathcal{M}'$ of a Multi-Complex $\mathcal{M}$ identifies a subset of the modification patterns of $\mathcal{M}$. Since $\mathcal{M}'$ is an MC, Theorem 4.3 ensures that such modifications can be applied to the initial cell complex represented in the root of $\mathcal{M}$ (which is the same root of $\mathcal{M}'$) in any order consistent with the partial order encoded in the DAG, and the result is the front cell complex of $\mathcal{M}'$ (see Figure 4.14).

Since MC-nodes represent refinement patterns, front($\mathcal{M}'$) is a cell complex at a lower resolution than the front cell complex of $\mathcal{M}$ (compare Figures 4.13 and 4.14). Different sub-MCs of $\mathcal{M}$ provide cell complexes at different resolutions and accuracies. Thus, the concept of sub-MC permits to obtain cell complexes at *intermediate, possibily variable resolutions* from an MC.

4.3.2 Enriched MCs

In this subsection, we define, for every Multi-Complex $\mathcal{M}$, a transformed DAG, called the *enriched MC* associated with $\mathcal{M}$, where arcs are labelled with sets of cells. The enriched MC permits to visualize some properties of an MC easier and has the advantage of providing a uniform treatment of all cells; basic primitives for the MC will be defined (in Section 5.8) by referring to the enriched MC, and data structures for MCs (presented in Chapter 6) will represent the enriched MC.

In an MC, each arc (C_i, C_j) corresponds to a non-empty set of k-cells belonging to both $\Pi_2(C_i)$ and $\Pi_1(C_j)$, and, conversely, whenever a k-cell γ belongs to both $\Pi_2(C_i)$ and $\Pi_1(C_j)$, then $\mathcal{M}$



(a)

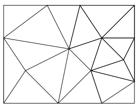

(b)

Figure 4.14: (a) A sub-MC containing nodes $\{0, 3, 4, 5\}$, and (b) its front cell complex.

contains arc (C_i, C_j). If we associate with an arc (C_i, C_j) the set of cells $\gamma \in \Pi_2(C_i) \cap \Pi_1(C_j)$, then each k-cell is associated with some arc, except for those belonging to the front complex. In order to have a consistent treatment of all top-cells, we introduce "dummy" arcs, joining each node C_i, such that at least one top cell of $\Pi_2(C)$ is in front($\mathcal{M}$), to an additional dummy node.

Based on such idea, we define the *enriched MC* as a transformed DAG obtained from an MC. The definition is constructive:

Definition 4.16 *The* enriched Multi-Complex *associated with a Multi-Complex $\mathcal{M}$ is the DAG obtained from $\mathcal{M}$ by:*

1. *adding a new node C_D, corresponding to a modification pattern of the form $C_D \equiv (\Gamma, \emptyset)$, where Γ is the front cell complex of $\mathcal{M}$;*
2. *adding an arc (C_i, C_D) for each node C_i of $\mathcal{M}$ such that at least one k-cell of $\Pi_2(C_i)$ is in Γ;*

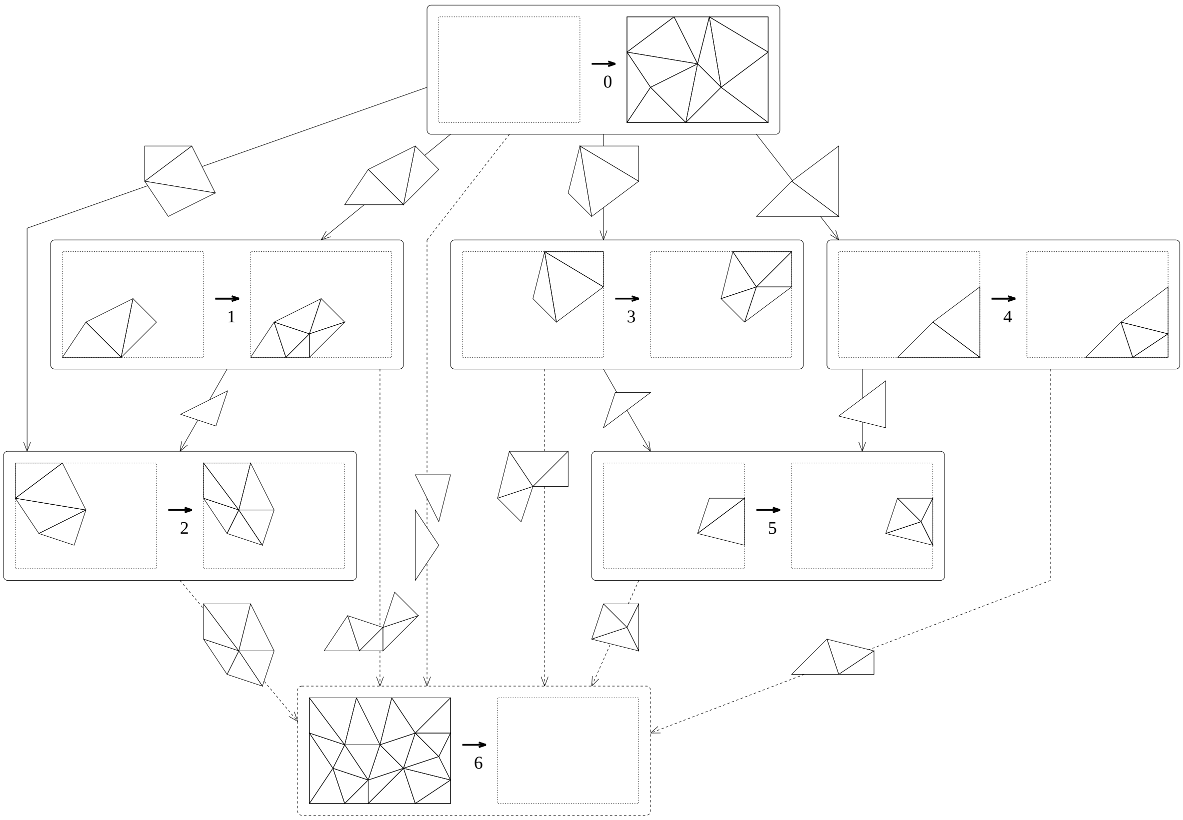


Figure 4.15: The enriched MC corresponding to the Multi-Complex of Figure 4.13; dummy arcs are dashed.

3. *labelling each arc* (C_i, C_j) *with the cell complex* $\mathrm{sub}(\Pi_2(C_i) \cap \Pi_1(C_j), \Pi_2(C_i))$ *(or, equivalently, with* $\mathrm{sub}(\Pi_2(C_i) \cap \Pi_1(C_j), \Pi_1(C_j))$*).*

Figure 4.15 shows the enriched version of the MC of Figure 4.13.

The new node C_D added is called the *drain* since it has no outgoing arcs. The drain can be viewed as a modification pattern specifying the destruction of the front cell complex of $\mathcal{M}$. The dummy arcs introduced in Definition 4.16 are called *front arcs*, since they are labelled with the cells of the front complex.

The following properties follow immediately from Definition 4.16:

Property 4.3 *For every node* C *of an MC, the union of the labels of its incoming arcs in the enriched MC is equal to* $\Pi_1(C)$*, and the union the labels of its outgoing arcs is equal to* $\Pi_2(C)$*.*

Property 4.4 *The front cell complex of a* k*-dimensional Multi-Complex* $\mathcal{M}$ *is the union of the labels of the front arcs of the enriched version of* $\mathcal{M}$*.*

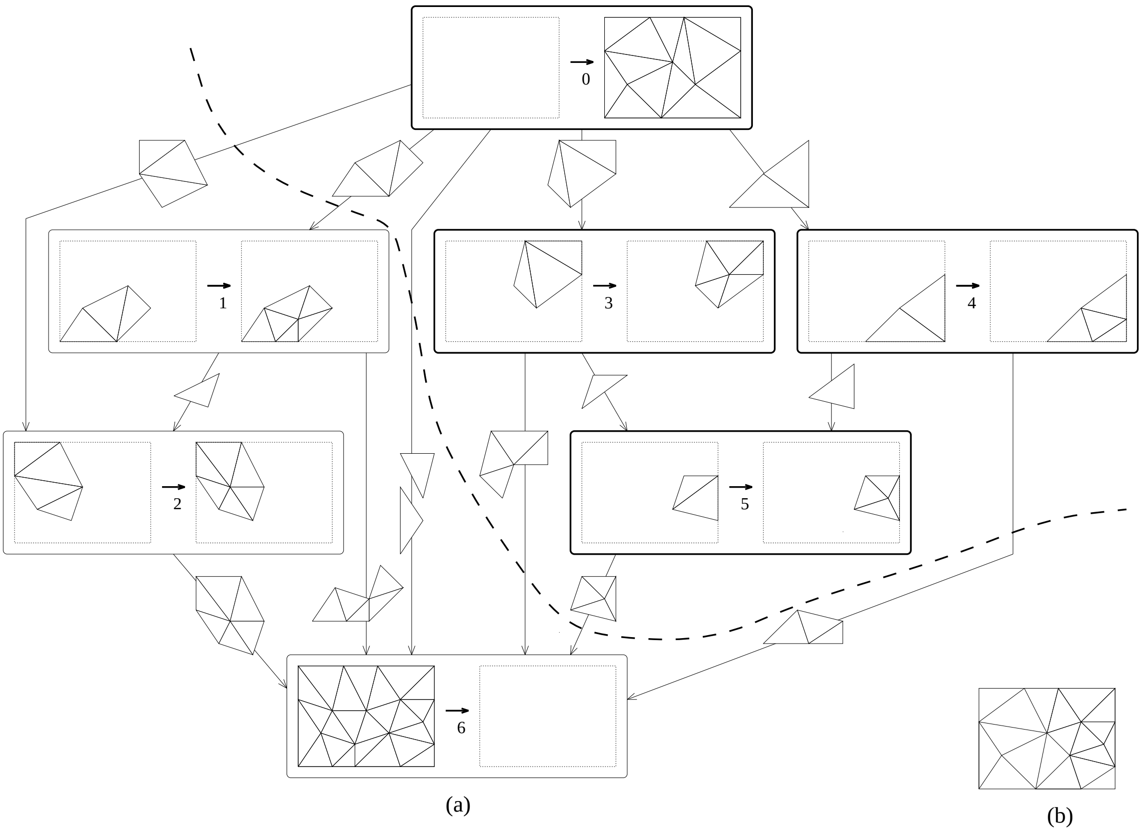


Figure 4.16: (a) A sub-MC, containing nodes $\{0, 3, 4, 5\}$, visualized on the enriched MC with the arcs of its front shown intersected by a dashed line; (b) its front cell complex.

For convenience, we define the *source* and the *destination* of a top-cell in the enriched MC as the ones of the arc containing γ in its label:

Definition 4.17 *The* source *of a cell γ in a Multi-Complex is the node C, such that $\gamma \in \Pi_2(C)$. The* destination *of γ is the node C of the enriched MC, such that $\gamma \in \Pi_1(C)$.*

Note that the source of a cell may be the root, and its destination may be the drain.

The enriched MC for a sub-MC $\mathcal{M}'$ of a Multi-Complex $\mathcal{M}$ is obtained by applying Definition 4.16 to $\mathcal{M}'$, considered as an MC on its own. However, it is more useful to visualize a sub-MC $\mathcal{M}'$ as a cut on the enriched MC of $\mathcal{M}$, as explained below.

Let $\mathcal{M}$ be a Multi-Complex, and $\mathcal{M}'$ a sub-MC of $\mathcal{M}$. In the enriched MC of $\mathcal{M}$, we consider the set of the arcs joining a node of $\mathcal{M}'$ to a node belonging to $\mathcal{M}$ but not to $\mathcal{M}'$. The union of the labels of such arcs gives the front complex of $\mathcal{M}'$. We note that such set contains exactly one arc on every root-to-drain path; in other words, it forms what we call an *MC-front*:

Definition 4.18 *Let $\mathcal{M}$ be a Multi-Complex. An* MC-front *is a subset $\mathcal{A}$ if the arcs of the enriched MC associated with $\mathcal{M}$, such that $\mathcal{A}$ contains exactly one arc for each root-to-drain path.*

Figure 4.16 shows a sub-MC and the corresponding MC-front. Conversely, given any MC-front on an enriched Multi-Complex, it is easy to verify that the set of nodes, which can be reached from the root without traversing an arc of the MC-front, forms a sub-MC. Thus, sub-MCs of a Multi-Complex $\mathcal{M}$ are in one-to-one correspondence with MC-fronts of the enriched version of $\mathcal{M}$:

Property 4.5 *Let $\mathcal{M}$ be a Multi-Complex. For each sub-MC $\mathcal{M}'$ of $\mathcal{M}$, the set of arcs (C_i, C_j) of the enriched MC corresponding to $\mathcal{M}$, such that C_i is in $\mathcal{M}'$ and C_j is not in $\mathcal{M}'$, is an MC-front.*

Property 4.6 *Let $\mathcal{M}$ be a Multi-Complex. For each MC-front $\mathcal{A}$ of the enriched MC corresponding to $\mathcal{M}$, the set of all nodes C_i, such that there exists a path from the root to C_i which does not contain arcs of $\mathcal{A}$, is a sub-MC of $\mathcal{M}$.*

Two special cases of MC-fronts for a Multi-Complex $\mathcal{M}$ are the one formed by the arcs entering the drain, whose associated sub-MC is $\mathcal{M}$ itself, and whose front complex is front($\mathcal{M}$) (i.e., the complex at the maximum resolution available in $\mathcal{M}$), and the one formed by the arcs outgoing from the root, whose associated sub-MC contains just the root C_0, and whose front complex is $\Pi_2(C_0)$ (i.e., the complex at the minimum available resolution).

4.4 Combined Cell Complexes from a Multi-Complex

Intuitively, a *combined cell complex* of an MC $\mathcal{M}$ is any complex formed by cells of $\mathcal{M}$ and representing a portion of the object modeled in $\mathcal{M}$. This idea is formalized as follows:

Definition 4.19 *A* combined cell complex *associated with a k-dimensional Multi-Complex $\mathcal{M}$ is a regular k-dimensional cell complex Γ such that:*

1. *every cell of Γ belongs to $\mathcal{M}$;*
2. *Γ does not contain any pair of k-cells γ_1 and γ_2, such that $\gamma_1 \in \Pi_1(C_1)$ and $\gamma_2 \in \Pi_2(C_2)$, with $C_1 \equiv C_2$ or C_1 an ancestor of C_2.*

On the enriched MC, Condition 2 is equivalent to saying that γ_1 and γ_2 must not be in the labels of two distinct arcs lying on the same directed path. Condition 2 tells that complexes

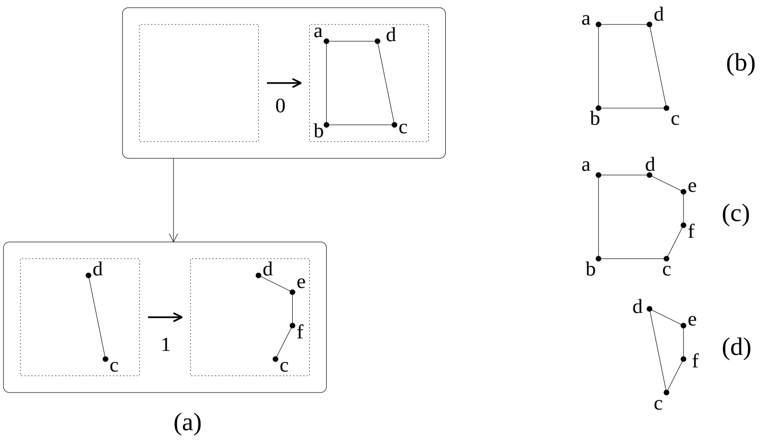


Figure 4.17: (a) A one-dimensional Multi-Complex in $\mathbb{E}^2$; and three cell complexes made of cells from it: (b) and (c) are combined cell complexes, (d) is not a combined cell complex because it contains cells from $\Pi_1(C)$ and $\Pi_2(C)$ for the same node C.

obtained by glueing (parts of) alternative representations of the same portion of the target object are not combined cell complexes. For instance, in Figure 4.17, (d) is not a combined complex because it is obtained by glueing two representations, at two different resolution, of the same line portion. Let cell γ_1 be segment dc and cell γ_2 be segment de: we have that $\gamma_1 \in \Pi_1(C)$ and $\gamma_2 \in \Pi_2(C)$ for the same MC-node C.

The resolution of a combined complex depends on that of its cells, and, in general, it can be variable over the complex.

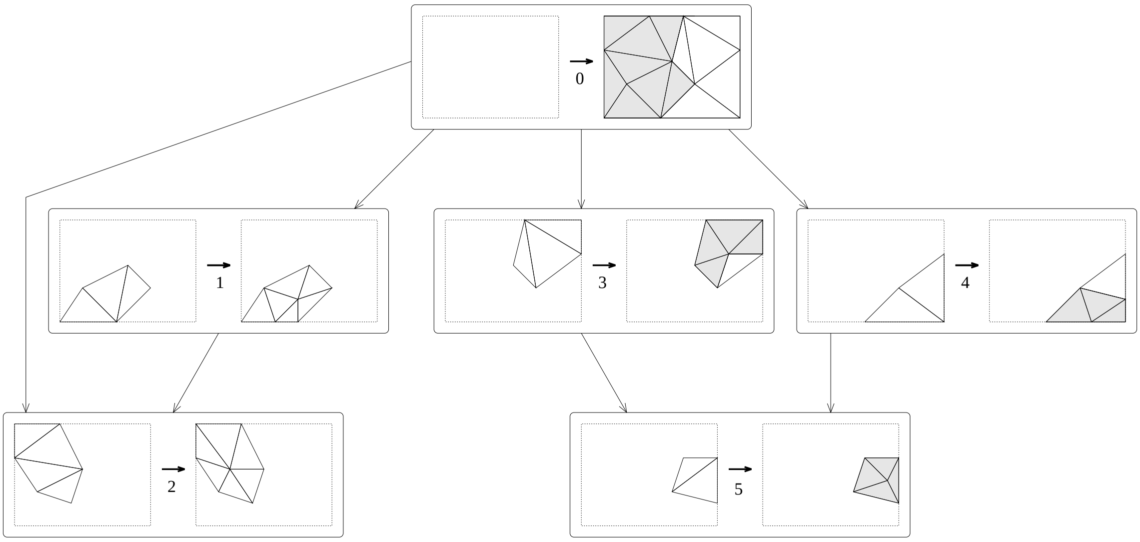
A combined cell complex that represents the whole object is called *global*:

Definition 4.20 *A combined cell complex* Γ *is* global *if there exists no other combined complex* Γ' *such that* $\Gamma \subset \Gamma'$ *(as sets of cells).*

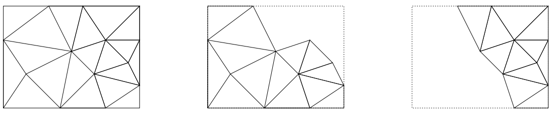
A combined cell complex which is not global is called *partial*. Figure 4.18 shows a global and two partial combined cell complexes in the two-dimensional case.

Note that any combined complex is necessarily contained into a global one. Global combined cell complexes of an MC are representations of the whole target geometric object. Partial combined cell complexes represent just a part of the whole object, but they are not the only complexes that can be obtained by combining cells from of an MC. In fact, some cell complexes may be obtained by glueing parts of alternative representations of the same object portion (see Figure 4.17 (d)). However, such representations may be topologically and geometrically consistent, but are not guaranteed to "make sense" with respect to the original object.

In the following we provide a characterization of *global combined cell complexes* of an MC. A partial combined cell complex is simply a subset of a global one.



(a)



(b) (c) (d)

Figure 4.18: (a) An MC; and three combined cell complexes, where (b) is global, and (c), (d) are partial. The triangles of (b) are shown shaded in (a). Note that (c) and (d) are proper subsets of (b).

4.4.1 Global Combined Cell Complexes and Sub-MCs

The following two lemmas show that the global combined cell complexes of a Multi-Complex $\mathcal{M}$ are in one-to-one correspondence with the sub-MCs of $\mathcal{M}$, or, equivalently, with the MC-fronts of the associated enriched MC. Figure 4.16 shows a sub-MC, its MC-front, and the corresponding cell complex.

Lemma 4.3 *For every sub-MC $\mathcal{M}'$ of a k-dimensional Multi-Complex $\mathcal{M}$, the front cell complex Γ of $\mathcal{M}'$ is a global combined cell complex of $\mathcal{M}$.*

Proof Trivially, Γ is made of cells of $\mathcal{M}'$, and, thus, of cells of $\mathcal{M}$.

Two cells γ_1 and γ_2 in Γ cannot violate Condition 2 in Definition 4.19, since in that case the arcs of the enriched MC containing γ_1 and γ_2, respectively, in their labels would lie on the same root-to-drain path in the enriched $\mathcal{M}$: this is a contradiction because the front arcs of $\mathcal{M}'$ are an MC-front. Thus, Γ is a combined cell complex of $\mathcal{M}$.

Now, we show that Γ is global. Let Γ' be a cell complex formed by adding to Γ a k-cell γ of $\mathcal{M}$, which is not already in Γ. Then, in the enriched MC the arc having γ in its label would be on the same directed path as one of the arcs of the MC-front defining $\mathcal{M}'$; thus, Γ' would not be a combined cell complex. $\square$

Lemma 4.4 *Let $\mathcal{M}$ be a k-dimensional Multi-Complex in $\mathbb{E}^d$, and Γ be a global combined cell complex of $\mathcal{M}$. Then, there exists a sub-MC $\mathcal{M}'$ of $\mathcal{M}$ such that the front cell complex of $\mathcal{M}'$ is the same as Γ.*

Proof Since two k-cells of Γ cannot belong to the labels of two different arcs on the same root-to-drain path in $\mathcal{M}'$, it follows that the set $\mathcal{A}$ of arcs containing the k-cells of Γ in their labels must be a subset of an MC-front of $\mathcal{M}$.

Now, we show that $\mathcal{A}$ must be an MC-front (i.e., no arc is missing), and that every k-cell labelling an arc of $\mathcal{A}$ must be in Γ (i.e., no cell is missing).

Let $\mathcal{A}'$ be an MC-front of $\mathcal{M}$ containing $\mathcal{A}$. Then, for Theorem 4.3, $\mathcal{A}'$ provides a global combined complex Γ', and, trivially, $\Gamma' \supseteq \Gamma$. For absurd, let $\Gamma' \setminus \Gamma$ be not empty. Since both Γ' and Γ are regular, there exists at least a k-cell $\gamma \in (\Gamma' \setminus \Gamma)$. Therefore, Γ is not global, which is a contradiction. Thus, $\Gamma' = \Gamma$. $\square$

Puppo [Pup96, Pup98] has shown a stronger result for a special class of two-dimensional simplicial MCs in $\mathbb{E}^2$. He proved that (under some additional hypothesis) every triangulation formed by triangles of an MC $\mathcal{M}$ and covering the whole carrier is the front complex of a sub-MC of $\mathcal{M}$, i.e., it is a global combined complex of $\mathcal{M}$.

In the following subsection, we report this result. Since the theorem of [Pup96, Pup98] extends in a straight-forward way to d-dimensions, we formulate it directly in $\mathbb{E}^d$.

4.4.2 Conservative Multi-Complexes

Conservative Multi-Complexes correspond to the subclass of d-dimensional MCs in $\mathbb{E}^d$ (i.e., models of the same dimension as the space embedding them) considered in [Pup96, Pup98] for $d = 2$ and simplicial cells.

Such class of MCs has a special interest because the result shown in [Pup96, Pup98] applies to it; moreover, we will see in Section 5.5.2, that it supports more efficient query algorithms.

Definition 4.21 *A* conservative Multi-Complex *is a d-dimensional Multi-Complex $\mathcal{M}$ in $\mathbb{E}^d$ such that, for every MC-node C of $\mathcal{M}$ different from the root, $\Delta(\Pi_1(C)) = \Delta(\Pi_2(C))$.*

Intuitively, in a conservative MC modification patterns do not change the shape of the carrier. For short, we denote with $\Delta(C)$ the carrier of $\Pi_1(C)$ and $\Pi_2(C)$ for an MC-node C. It also implies that the carrier of every global combined cell complex is the same; in particular, it is the same as the carrier of the front cell complex of $\mathcal{M}$. For short, we denote it with $\Delta(\mathcal{M})$.

In the following we list a number of interesting properties that hold in a conservative MC:

Property 4.7 *For every MC-node C of a conservative Multi-Complex, both $\Pi_1(C)$ and $\Pi_2(C)$ consist of one connected component.*

This follows from the fact that $\Pi_1(C)$ and $\Pi_2(C)$ cover the same carrier: if the carrier is made of two connected components, then C can be split in two independent modification patterns, contradicting the fact that C is minimal.

Property 4.8 *Let C be a node of a conservative Multi-Complex $\mathcal{M}$. Let us denote with $\partial'(\Pi_i)$ the subset of $\partial(\Pi_i)$ obtained by eliminating the cells that are contained in the boundary of $\Delta(\mathcal{M})$. For $i = 1, 2$, we have that $\partial'(\Pi_1(C)) = \partial'(\Pi_2(C))$.*

This follows from the fact that, in an MC, the boundary cells of $\Pi_1(C)$ which are internal to the carrier must belong to $\Pi_2(C)$; moreover, in a conservative MC the carrier and, thus, its boundary, is fixed. For short, we denote with $\partial'(C)$ the set $\partial'(\Pi_1(C)) = \partial'(\Pi_2(C))$.

In a conservative MC, dependency relations between nodes have a connection with overlapping of their carriers:

Property 4.9 *Let $\mathcal{M}$ be a conservative Multi-Complex, and C_1, C_2 be two nodes of $\mathcal{M}$:*

1. *if $\Delta(C_1) \cap \Delta(C_2) \neq \Delta(\partial'(C_1) \cap \partial'(C_2))$, then either C_1 is an ancestor of C_2, or vice versa;*
2. *the converse of 1 is not necessarily true;*
3. *if C_1 is a parent of C_2, then $\Delta(C_1) \cap \Delta(C_2) \neq \Delta(\partial'(C_1) \cap \partial'(C_2))$.*

Fact 1 of Property 4.9 follows from the fact that $\Pi_i(C_1)$ and $\Pi_j(C_2)$ (for all $i, j \in \{1, 2\}$) cannot coexist in a combined cell complex, thus C_1 and C_2 must be on the same directed path. Fact 2 can be easily seen through the counter-example shown in Figure 4.19. Fact 3 is straight-forward since in this case $\Delta(C_1) \cap \Delta(C_2) = \Delta(\Pi_2(C_1) \cap \Pi_1(C_2))$ (which contains at least a top-cell by definition).

Now, we formulate an extension of the main theorem of [Pup96, Pup98] for conservative Multi-Complexes in d-dimensions. We require the following additional property for a conservative MC:

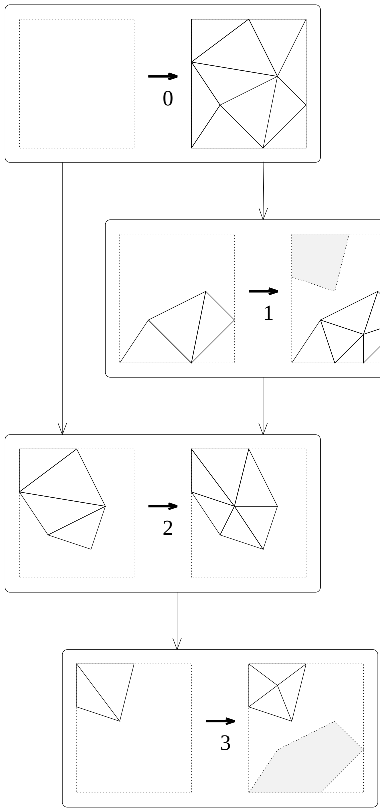


Figure 4.19: MC-Node 3 depends on MC-node 1, but their domains are disjoint (the domain of each of the two nodes is shown shaded in the other node).

Definition 4.22 *A k-dimensional Multi-Complex $\mathcal{M}$ satisfies the* permanence of $(k-1)$-cells *if, for every MC-node C of $\mathcal{M}$, for every $(k-1)$-cell γ' of $\Pi_2(C)$, if γ' is in $\Pi_2(C_i)$ for some ancestor C_i of C, then γ' is in $\Pi_1(C)$ as well.*

Intuitively, the permanence property means that $(k-1)$-cells removed by applying a modification patterns C_i of $\mathcal{M}$ are never recreated by applying another modification pattern following C_i in the partial order.

Theorem 4.5 *Let $\mathcal{M}$ be a d-dimensional conservative MC satisfying the permanence of $(d-1)$-cells, and Γ' be a cell complex made of cells of $\mathcal{M}$, and such that $\Delta(\Gamma') = \Delta(\text{front}(\mathcal{M}))$. Then, there exists a sub-MC $\mathcal{M}'$ of $\mathcal{M}$ such that $\Gamma' = \text{front}(\mathcal{M}'))$.*

The proof of Theorem 4.5 is a direct extension of the one provided in [Pup96, Pup98], and it is not reported here.

4.5 Adjacency Relations in a Multi-Complex

From Section 2.3.1.3, we recall that two k-cells in a complex are called $(k-1)$-*adjacent*, or simply *adjacent*, if they share a common $(k-1)$-face. If γ' is the shared $(k-1)$-face, we say that the two k-cells are *adjacent at* γ'. In a manifold k-cell complex, no more than two k-cells are adjacent at a given $(k-1)$-cell.

Although all cell complexes that can be obtained from an MC are manifolds, an arbitrary number of k-cells in an MC may share the same $(k-1)$-face; however, no more than two at a time can exist in a combined cell complex.

The k-cells γ_2 adjacent to a given k-cell γ_1 at a given $(k-1)$-face γ' are distinct in two types which, intuitively, correspond to the case when γ_1 and γ_2 can co-exist in a combined cell complex (i.e., their union represents a part of the target object), and the case when this does not hold. The two types of adjacency are defined below:

Definition 4.23 *In a k-dimensional Multi-Complex $\mathcal{M}$, two k-cells γ_1 and γ_2 are:*

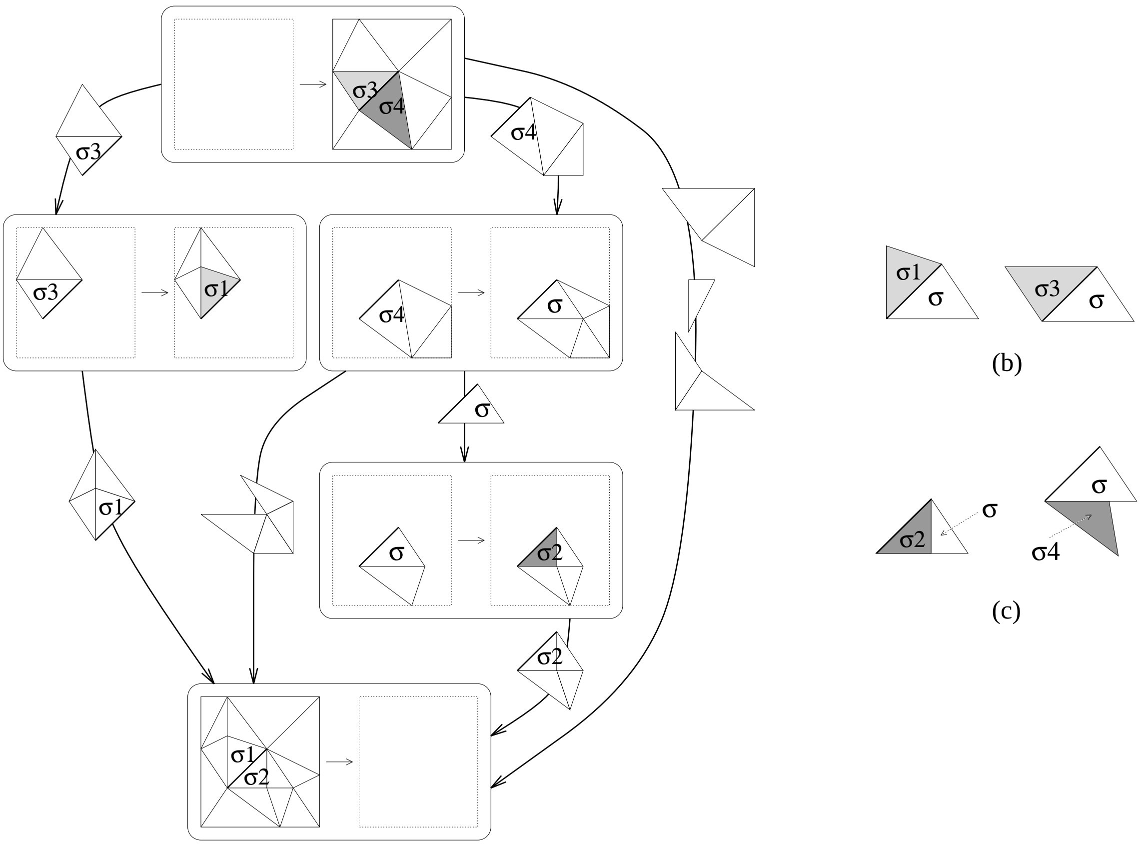
- co-adjacent *at a $(k-1)$-face γ', if γ_1 and γ_2 share γ', and γ_1 and γ_2 are in the label of either the same arc, or of two arcs which do not lie on the same directed path in the enriched version of $\mathcal{M}$;*
- counter-adjacent *at a $(k-1)$-face γ', if γ_1 and γ_2 share γ', and γ_1 and γ_2 belong to the label of two distinct arcs lying on a directed path in the enriched version of $\mathcal{M}$.*

Intuitively, co-adjacent cells may co-exist in a combined cell complex, counter-adjacent cells may not. See Figure 4.20 for an example. Note that all k-cells that are co-adjacent to a given k-cell at a given $(k-1)$-face are mutually counter-adjacent, and they all belong to the labels of arcs lying on the same directed path in the enriched MC.

A k-cell γ may have an arbitrary number of co-adjacent and counter-adjacent cells at each of its $(k-1)$-faces. In the following, we define four adjacency relations that provide at most one cell for each $(k-1)$-face of a k-cell γ, and are sufficient to retrieve all the co–adjacent k-cells of a given k-cell in an MC. Such relations will be useful in the design of data structures and methods to reconstruct the connectivity structure of combined cell complexes extracted from an MC (see Chapter 6).

Definition 4.24 *Given a k-dimensional MC, and two k-cells γ_1 and γ_2, sharing a $(k-1)$-dimensional face γ':*

1. *γ_1 and γ_2 are* co-adjacent$_1$ *at γ' if $\gamma_1, \gamma_2 \in \Pi_1(C)$ for the same node C in the enriched MC;*



(a)

Figure 4.20: Adjacency relations in a two-dimensional simplicial MC. Triangle σ is co-adjacent to σ_1 and σ_3 at the thick edge, and is counter-adjacent to σ_2 and σ_4. (a) The enriched MC; (b) σ can co-exist with σ_1 or σ_3; (c) σ cannot co-exist with σ_2 or σ_4 since they partially overlap.

2. *γ_1 and γ_2 are* co-adjacent$_2$ *at γ' if $\gamma_1, \gamma_2 \in \Pi_2(C)$ for the same node C in the enriched MC;*

3. *γ_2 is* counter-adjacent$_{1,2}$ *to γ_1 at γ' if, denoted with C the node of the enriched MC such that $\gamma_1 \in \Pi_1(C)$, we have that $\gamma_2 \in \Pi_2(C)$, and $\gamma' \in \partial(\Pi_1(C)) \cap \partial(\Pi_2(C))$;*

4. *γ_2 is* counter-adjacent$_{2,1}$ *to γ_1 at γ' if, denoted with C the node of the enriched MC such that $\gamma_1 \in \Pi_2(C)$, we have that $\gamma_2 \in \Pi_1(C)$, and $\gamma' \in \partial(\Pi_1(C)) \cap \partial(\Pi_2(C))$;*

In Figure 4.20, the counter-adjacent$_{2,1}$ triangle of σ at the thick edge is σ_4 and its counter-adjacent$_{1,2}$ is σ_2; triangles σ_3 and σ_4 are mutually co-adjacent$_2$ at the thick edge, and σ_1 and σ_2 are mutually co-adjacent$_1$ at the thick edge.

Relations co-adjacent$_1$ and co-adjacent$_2$ link k-cells that are removed and created, respectively, in the application of the same modification pattern. Clearly, if two cells are in relation co-adjacent$_1$, or co-adjacent$_2$, then they are also co-adjacent, but the converse is in general not true.

Relation counter-adjacent$_{1,2}$ links a k-cell γ to cells that are created by applying the same modification pattern that removes γ; conversely, counter-adjacent$_{2,1}$ links γ to cells that are removed by applying the same modification pattern that creates γ. If two cells are in relation counter-adjacent$_{1,2}$, or counter-adjacent$_{2,1}$, then they are also counter-adjacent, but the converse is not true in general.

Note that co-adjacent$_1$ and co-adjacent$_2$ are symmetric relations. Relations counter-adjacent$_{1,2}$ and counter-adjacent$_{2,1}$ are anti-symmetric, and inverse of each other.

The following property will be exploited in the design of data structures:

Property 4.10 *Relations co-adjacent$_1$ and counter-adjacent$_{1,2}$ are mutually exclusive: a k-cell cannot have both a co-adjacent$_1$ and a counter-adjacent$_{1,2}$ cell at the same $(k-1)$-face. The same property holds for relations co-adjacent$_2$ and counter-adjacent$_{2,1}$.*

We claim that, by combining the four adjacency relations defined above, it is possible to obtain all the k-cells that are co-adjacent to any k-cell in a Multi-Complex. Thus, they permit the reconstruction of adjacency relations in any combined cell complex.

In order to formalize and prove our claim, we need the following auxiliary definition of a transitive and reflexive closure of relations counter-adjacent$_{1,2}$ and counter-adjacent$_{2,1}$:

Definition 4.25 *Let p be either 1 or 2 (i.e., relation counter-adjacent$_{p,3-p}$ be either counter-adjacent$_{1,2}$ or counter-adjacent$_{2,1}$). Let γ_A and γ_B be two k-cells and γ' be a $(k-1)$-cell of a k-dimensional MC $\mathcal{M}$.*

Cell γ_A is counter-adjacent$^*_{p,3-p}$ to γ_B at γ' if either $\gamma_A = \gamma_B$, or there exists a sequence of k-cells $\gamma_A = \gamma_1, \ldots, \gamma_m = \gamma_B$ such that, for every $1 \leq i \leq k-1$, γ_i is counter-adjacent$_{p,3-p}$ to γ_{i+1} at γ'.

Now, we are able to state and prove our claim that it is possible to express relation co-adjacent by combining relations co-adjacent$_1$, co-adjacent$_2$, counter-adjacent$_{1,2}$ and counter-adjacent$_{2,1}$:

Theorem 4.6 *Two k-cells γ_A and γ_B are co-adjacent at a $(k-1)$-cell γ' if and only if one of the following two equivalent conditions holds:*

1. *there exist two k-cells γ_C and γ_D such that γ_C and γ_D are co-adjacent$_2$ at γ', γ_A is counter-adjacent$^*_{1,2}$ to γ_C at γ', γ_B is counter-adjacent$^*_{1,2}$ to γ_D at γ' (see Figure 4.21).*
2. *there exist two k-cells γ_C and γ_D such that γ_C and γ_D are co-adjacent$_1$ at γ', γ_A is counter-adjacent$^*_{2,1}$ to γ_C at γ', γ_B is counter-adjacent$^*_{2,1}$ to γ_D at γ' (see Figure 4.22).*

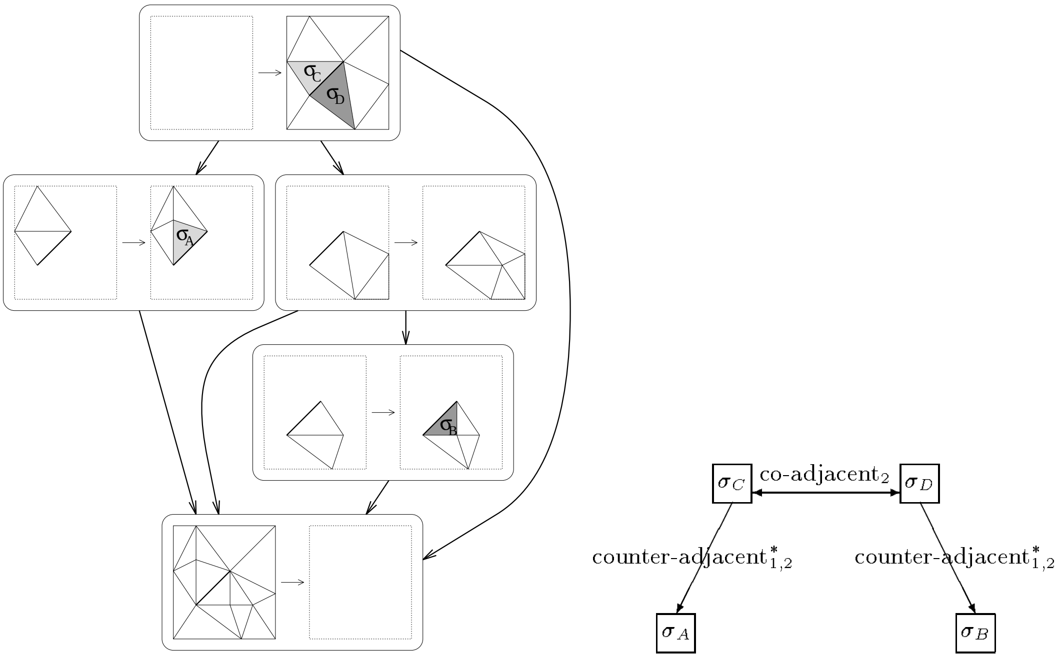


Figure 4.21: Retrieving the relation of co-adjacency between triangles σ_A and σ_B by using relation co-adjacent$_2$.

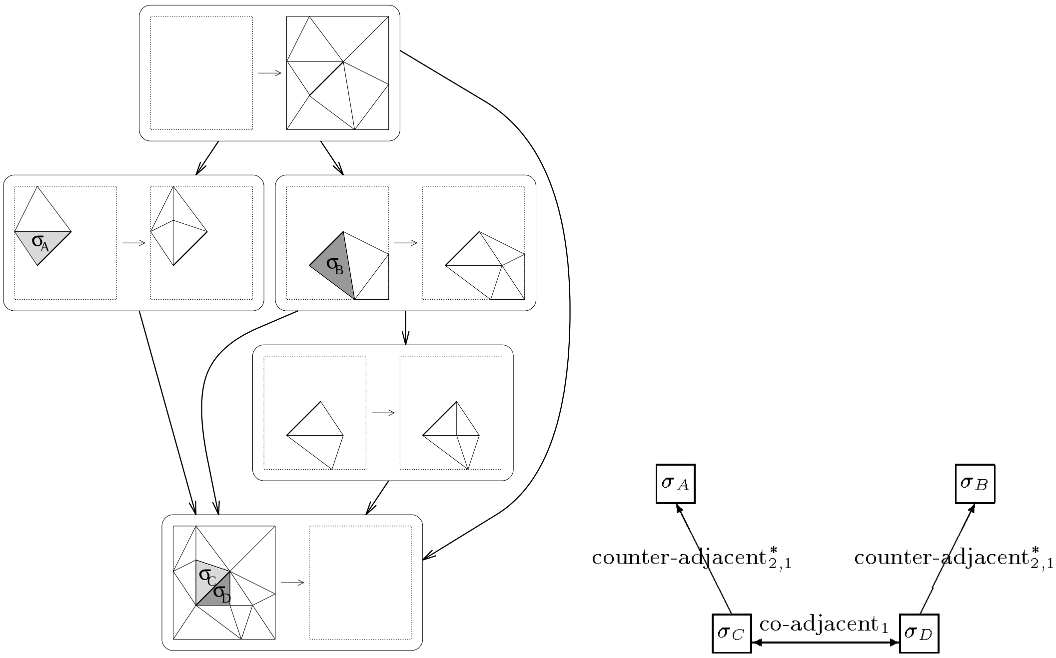


Figure 4.22: Retrieving the relation of co-adjacency between triangles σ_A and σ_B by using relation co-adjacent$_1$.

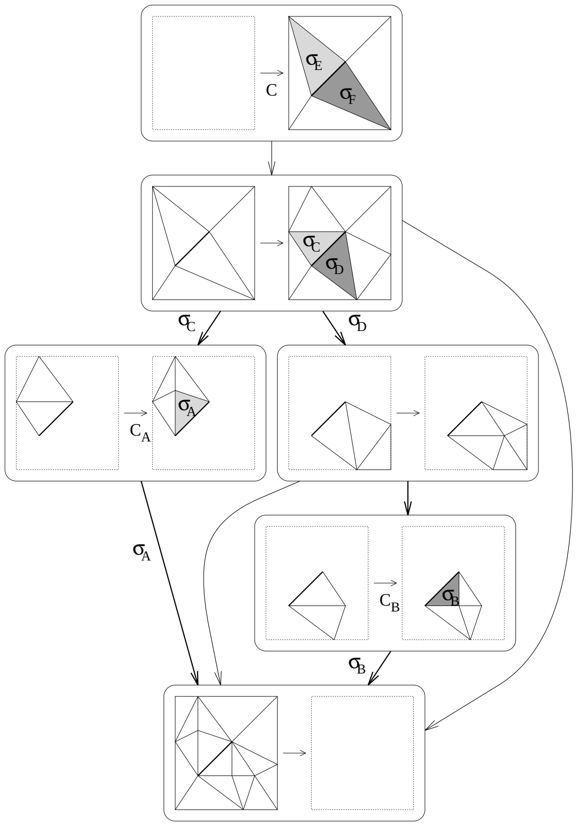


Figure 4.23: The min-length assumption in Theorem 4.6: σ_C and σ_D are the ones indicated, although σ_E and σ_F satisfy the hypothesis of the theorem as well.

Proof We give the proof just for Condition 1. The proof for Condition 2 is a straight-forward variant.

First, we show that Condition 1 implies that γ_A and γ_B are co-adjacent.

If $\gamma_A = \gamma_C$ and $\gamma_B = \gamma_D$ then γ_A and γ_B are co-adjacent$_2$, and, thus, co-adjacent.

Otherwise, without loss of generality we consider the pair of cells γ_C and γ_D such that Condition 1 holds with a minimum number of steps of the transitive closure (see Figure 4.23). We call this the *min-length assumption* in the remainder of the proof.

Let $\gamma_A \in \Pi_2(C_A)$, $\gamma_B \in \Pi_2(C_B)$, and $\gamma_C, \gamma_D \in \Pi_2(C)$. Cells γ_A and γ_C label arcs on the same path in the enriched MC, and they share γ'; a similar property holds for γ_B and γ_D. In order to prove that γ_A and γ_B are co-adjacent, it remains to show that the path containing γ_A and that containing γ_B are distinct.

Since γ_C and γ_D are co-adjacent, we have to consider two cases:

- γ_C and γ_D label the same arc of the enriched MC, thus, γ_C, γ_D are in $\Pi_1(C')$ for the same node C'.

  If γ' survives in $\Pi_2(C')$, then there are another two co-adjacent$_2$ cells at γ' in $\Pi_2(C')$, against the min-length assumption.

  Otherwise, γ' disappears in $\Pi_2(C')$, thus γ' must be recreated in some descendant C'' of C'. For the validity, γ' will be recreated as an internal cell of $\Pi_2(C'')$, thus there are another two co-adjacent$_2$ cells at γ' in $\Pi_2(C'')$, against the min-length assumption.

- γ_C and γ_D label two distinct arcs of the enriched MC, i.e., they start two different path from C.

  Such paths can only join at a node C' such that γ' is internal to $\Pi_1(C')$, thus either γ' survives in $\Pi_2(C')$ or it disappears, but in both cases this violates the min-length assumption.

Thus, the two paths containing γ_C and γ_D must split in C and never join.

Now, we show that, if γ_A and γ_B are co-adjacent, then Condition 1 is true.

Let $\gamma_A \in \Pi_2(C_A)$, $\gamma_B \in \Pi_2(C_B)$.

If $C_A = C_B$, Condition 1 is trivially true with $\gamma_A = \gamma_C$ and $\gamma_B = \gamma_D$.

Otherwise, we know that C_A and C_B must lie on distinct paths. Let us consider the nearest common ancestor C to of C_A and C_B.

The proof is based on the following facts:

1. Cell γ' is on the boundary of $\Pi_2(C_A)$, $\Pi_1(C_A)$, $\Pi_2(C_B)$, $\Pi_1(C_B)$.

   Let us consider the smallest sub-MC $\mathcal{M}'$ containing C_A, and C_B. C_A and C_B are leaves in $\mathcal{M}'$; front($\mathcal{M}'$) contains at least two cells incident in γ', i.e., γ_A and γ_B.

   γ' cannot be internal to $\Pi_2(C_A)$, otherwise two cells of $\Pi_2(C_A)$ would be incident at γ', and front($\mathcal{M}'$) would have three cells incident at γ', contradicting the fact that front($\mathcal{M}'$) is manifold. For the same reason, γ' cannot be internal to $\Pi_2(C_B)$. Thus, γ' is on the boundary of $\Pi_2(C_A)$ and $\Pi_2(C_B)$.

   Let us consider the sub-MC $\mathcal{M}''$ obtained from $\mathcal{M}'$ by removing C_A and its incoming arcs. γ' must be in $\partial(\Pi_1(C_A))$ because C_A is a valid modification pattern for front($\mathcal{M}''$): if $\gamma' \notin \partial(\Pi_1(C_A))$, then γ' would be on the boundary of front($\mathcal{M}''$), and replacing $\Pi_2(C_A)$ to $\Pi_1(C_A)$ would sew separate parts of the boundary, against the condition of validity. For the same reason, γ' must be in $\partial(\Pi_1(C_B))$.

2. From Fact 1 above, it follows that the predecessor of C_A in the path from C to C_A contains a cell γ'_A such that γ_A is counter-adjacent$_{1,2}$ to γ'_A at γ', and, symmetrically,

the predecessor of C_B in the path from C to C_B contains a cell γ'_B such that γ_B is counter-adjacent$_{1,2}$ to γ'_B at γ'.

The same argument can be repeated for each node of the two paths, thus concluding that γ_A is counter-adjacent$^*_{1,2}$ to a cell γ_C of C, and γ_B is counter-adjacent$^*_{1,2}$ to a cell γ_D of C at face γ'.

γ_C and γ_D are distinct since they label different arcs of two different paths outgoing from C.

3. γ_C and γ_D both belong to $\Pi_2(C)$, and share face γ', thus, they are co-adjacent$_2$ at γ'.

This concludes the proof. $\square$

4.6 Evaluating the Quality of a Multi-Complex

In this section, we first define some parameters related to the size and "shape" of the DAG describing a Multi-Complex. Next, we consider values of such parameters, which determine a good behavior of the MC with respect to the efficiency of query operations.

4.6.1 Shape Parameters of an Multi-Complex

Definition 4.26 *We define the* height *of a Multi-Complex as the maximum number of arcs on a path from the root to a leaf.*

For instance, the height of the MC in Figure 4.13 is two. A small height implies little work to reach the highest resolution in a top-down traversal of the DAG.

Definition 4.27 *We define the* width *of a node C as the number of top cells in $\Pi_2(C)$, and the* width *of a Multi-Complex as the maximum width of its nodes.*

A small width implies little work for testing geometric properties of the modification pattern represented by a node.

Definition 4.28 *The* size *of a Multi-Complex is the total number of top cells in it.*

The ratio of the size of an MC over the size of its front cell complex provides a measure of the storage overhead introduced by the multiresolution representation with respect to storing a single cell complex at the highest resolution. The following definition of *growth* generalizes this concept to encompass the storage overhead introduced by any sub-MC with respect to its front cell complex. Such overhead also affects the performance of query algorithms that find a combined cell complex by traversing the corresponding sub-MC.

Definition 4.29 *The* growth *of an MC is the maximum, over all sub-MCs $\mathcal{M}'$ of $\mathcal{M}$, of the the ratio between the size of $\mathcal{M}'$ and the size of the front cell complex of $\mathcal{M}'$.*

4.6.2 Desirable Properties for a Multi-Complex

In this subsection, we discuss some properties of an MC which are relevant to the efficiency of the encoding data structures, and/or of the algorithms which traverse the DAG of a Multi-Complex, as we will see in Chapter 6. Such properties are determined by the algorithm used to produce the initial cell complex and the sequence of modification patterns from which a Multi-Complex is built, as we will see in Chapter 7.

Let $\mathcal{U}$ be a class of Multi-Complexes. For instance, $\mathcal{U}$ may contain all the MCs built from sequences of modification patterns of a certain type (e.g., MCs built through iterative vertex insertion in a Delaunay triangulation, MCs built through iterative vertex split, etc.).

Definition 4.30 *A class $\mathcal{U}$ of Multi-Complexes has a* bounded width *if there exists a positive real number b such that, for every MC $\mathcal{M}$ in $\mathcal{U}$, the width of $\mathcal{M}$ is less or equal to b.*

Definition 4.31 *A class $\mathcal{U}$ of Multi-Complexes has a* logarithmic height *if there exists a positive real number b such that, for every MC $\mathcal{M}$ in $\mathcal{U}$, the ratio between the height of $\mathcal{M}$ and the logarithm of the number of its nodes is less or equal to b.*

In Section 7.5, we will see that bounded width and logarithmic height are important for achieving an optimal time complexity of some traversal operations on an MC, such as point location.

Definition 4.32 *A class $\mathcal{U}$ of Multi-Complexes has a* linear growth *if there exists a positive real number b such that, for every MC $\mathcal{M}$ in $\mathcal{U}$, the growth of $\mathcal{M}$ is less or equal to b.*

Note that, both for bounded width and linear growth, the constant b is necessarily ≥ 1, and $b = 1$ is possible only for MCs consisting of just one node.

If an MC has linear growth, then the size of any sub-MC is linear in the size of its front complex. In particular, the size of the MC itself is linear in the size of its front cell complex. This means that the multiresolution structure introduces only a linear storage overhead with respect to a simple cell complex at the maximum resolution.

In Sections 6.3.1.6 and 6.3.3.6, we will see that linear growth is fundamental to achieve an optimal time complexity (i.e., linear in the output size) for algorithms extracting a cell complex at a given resolution from an MC. These algorithms perform a top-down traversal of the DAG in order to find the MC-front providing the cell complex that is the answer to the query; linear growth guarantees that the size of the traversed sub-MC is linear in the output size.

The following two lemmas state that a bounded width implies a linear growth.

Lemma 4.5 *Let $\mathcal{U}$ be a class of MCs. If there exists a real number $c > 1$ such that, for every node C in an MC $\mathcal{M} \in U$:*

$$\frac{\text{size}(\Pi_2(C))}{\text{size}(\Pi_1(C))} > c$$

then $\mathcal{U}$ has a linear growth.

Proof Let $\mathcal{M}'$ be a sub-MC of $\mathcal{M}$, and $\{C_i \mid i = 0, \ldots, n\}$ be the set of its nodes. For any node C_i, let m_i and m'_i be the number of top cells in $\Pi_1(C_i)$, and in $\Pi_2(C_i)$, respectively.

The total number of top cells in $\mathcal{M}'$ is $N = \sum_{i=0}^{n}(m'_i)$ and the number of top cells in front$(\mathcal{M}')$ is $N' = \sum_{i=0}^{n}(m'_i - m_i)$. Since $\frac{m'_i}{m_i} > c$, $N' > \sum_{i=0}^{n}(m'_i(1 - \frac{1}{c})) = \frac{c-1}{c}\sum_{i=0}^{n}(m'_i)$. Thus, $\frac{N}{N'} \leq \frac{c}{c-1}$, which is a positive real number for any $c > 1$. □

Lemma 4.6 *Let $\mathcal{U}$ be a class of MCs. If $\mathcal{U}$ has a bounded width, then it has a linear growth.*

Proof If all MCs in $\mathcal{U}$ are made of just one node, the lemma is trivially true.

Otherwise, let $b > 1$ be the bound for the width of MCs in $\mathcal{U}$. We show that $\mathcal{U}$ satisfies the hypothesis of Lemma 4.5. Let m_i and m'_i be, as in the proof of Lemma 4.5, the number of top cells in $\Pi_1(C_i)$, and in $\Pi_2(C_i)$, respectively, for any MC-node C_i.

Since $\mathcal{U}$ has a bounded width, and all modification patterns in an MC are refinement patterns, then $b \geq m'_i \geq m_i + 1$, i.e., $m_i \leq m'_i - 1 \leq b - 1$. Thus, $\frac{m'_i}{m_i} \geq \frac{m_i+1}{m_i} = 1 + \frac{1}{m_i} \geq 1 + \frac{1}{b-1} = \frac{b}{b-1}$, which is a real number greater than 1. Thus, the hypothesis of Lemma 4.5 is satisfied, and $\mathcal{U}$ has linear growth. □

4.7 Existing Multiresolution Models as Multi-Complexes

In this section, we show how multiresolution models reviewed in Section 3.3 can be seen as special cases of Multi-Complexes.

Each of these models gives a class of MCs characterized by certain properties of the modification patterns, and, consequently, by certain properties in terms of width, height and growth. Moreover, each existing model provides an implementation of an MC with a specific data structure. Usually, such an implementation is optimized to take advantage of the specific properties of the modification patterns, and/or of information dependent on the application for which the model has been designed. However, often some information present in the MC is either lost or incorrectly represented in such a data structure. Examples of lost information are found in models which forget the DAG and replace it with a linear sequence [Hop96, KS97]; we will see examples of incorrectly represented information, for instance, in nested models. Such an aspect will be examined for existing models in Sections 4.7.1, 4.7.2, and 4.7.3. Viewing a multiresolution model as an MC helps understanding its properties better.

For the sake of simplicity and brevity, we explain the interpretation of just two-dimensional multiresolution models as two-dimensional MCs. Higher-dimensional models can be interpreted in a similar way.

Also, since the Multi-Complex has been defined as a model for manifold objects, we cannot encompass existing multiresolution models dealing with non-manifold entities (e.g., [LE97, PH97]).

4.7.1 Layered Models

A layered multiresolution model, consisting of a sequence $\Gamma_0, \Gamma_1, \ldots, \Gamma_h$ of cell complexes, can be interpreted as a Multi-Complex $\mathcal{M}$ where the DAG has the structure of a list:

- the root C_0 of $\mathcal{M}$ represents the modification pattern $(\emptyset, \Gamma_0)$;
- for every $0 < i \leq h$, $\mathcal{M}$ contains a node C_i, representing the modification pattern (Γ_{i-1}, Γ_i), and an arc from C_{i-1} to C_i.

It follows from the structure of the DAG that the only possible sub-MCs of $\mathcal{M}$ are those corresponding to prefixes of the list, and thus the only possible global combined cell complexes are $\Gamma_0, \Gamma_1, \ldots, \Gamma_h$.

Indeed, a modification pattern (Γ_{i-1}, Γ_i) may not be minimal. In this case, we can split it into two or more minimal modification patterns, thus creating two or more MC-nodes. This results in a DAG which is not a list, and thus the MC interpreting the layered model may have a larger expressive power (see Figure 4.24). Note that this additional expressive power is only visible when interpreting the layered model as a Multi-Complex, and it is not exploited in the layered model.

4.7.2 Nested Models

A nested model is usually represented as a tree, where each node corresponds to a complex whose carrier coincides with one of the cells of its parent node.

In a k-dimensional nested model, a node represents the refinement of a top-cell γ of its parent node into a complex Γ'. The corresponding modification pattern is $C = (\mathrm{sub}(\{\gamma\}, \Gamma), \Gamma')$, where Γ is the complex contained in the parent node. In general (and in most existing nested models), C is not a valid modification pattern for Γ, since Γ' may split some of the $(k-1)$-faces of γ: the

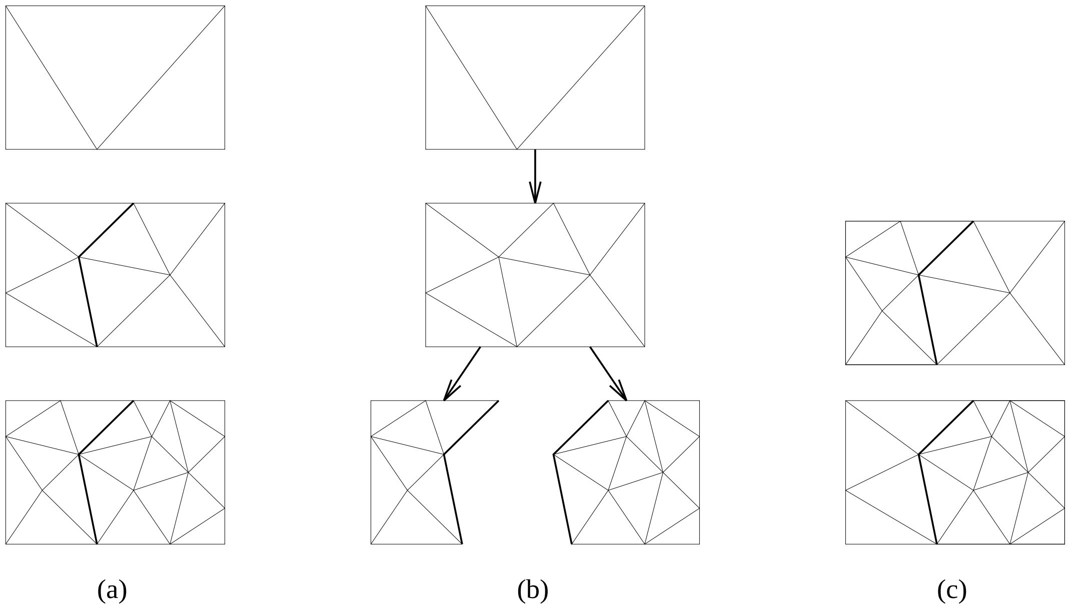


Figure 4.24: (a) A layered model with three layers, where the transition from the second to the third layer corresponds to a non-minimal modification; (b) the structure of the MC interpreting the model; (c) complexes that can be obtained from the MC but not from the layered model.

cells of $\Gamma \setminus \{\gamma\}$ and those of Γ' have intersections that do not coincide with the common cells of their boundary complexes, thus violating Definition 4.3. Figures 4.25 (a) and (b) illustrate this problem.

If Γ' splits a $(k-1)$-face γ' of γ, this means that γ can be replaced consistently with Γ' only if also the cell γ_1, adjacent to γ along γ' in Γ, is replaced by a complex Γ'_1 splitting face γ' in the same way. Thus, the tree nodes representing the refinement of γ and γ' must become one node of the Multi-Complex. The result of such node merging is shown in Figure 4.25 (c).

A Multi-Complex representing a k-dimensional nested model is obtained as follows:

- the MC-nodes are obtained from the nodes of the tree by iteratively joining pairs of nodes corresponding to complexes in which two adjacent k-cells have been refined by decomposing their common $(k-1)$-face; this defines a (not necessarily injective) mapping of nodes of the tree to MC-nodes.
- the MC arcs are obtained from the tree arcs according to the following rule: for every arc (N_1, N_2) of the tree, if N_1 maps to an MC-node C_1 and N_2 maps to an MC-node C_2, then arc (C_1, C_2) represents (N_1, N_2) in the MC; this defines a (not necessarily injective) mapping of the arcs of the tree to arcs of the MC.

Looking at the Multi-Complex for a given nested model enables us to understand what cell

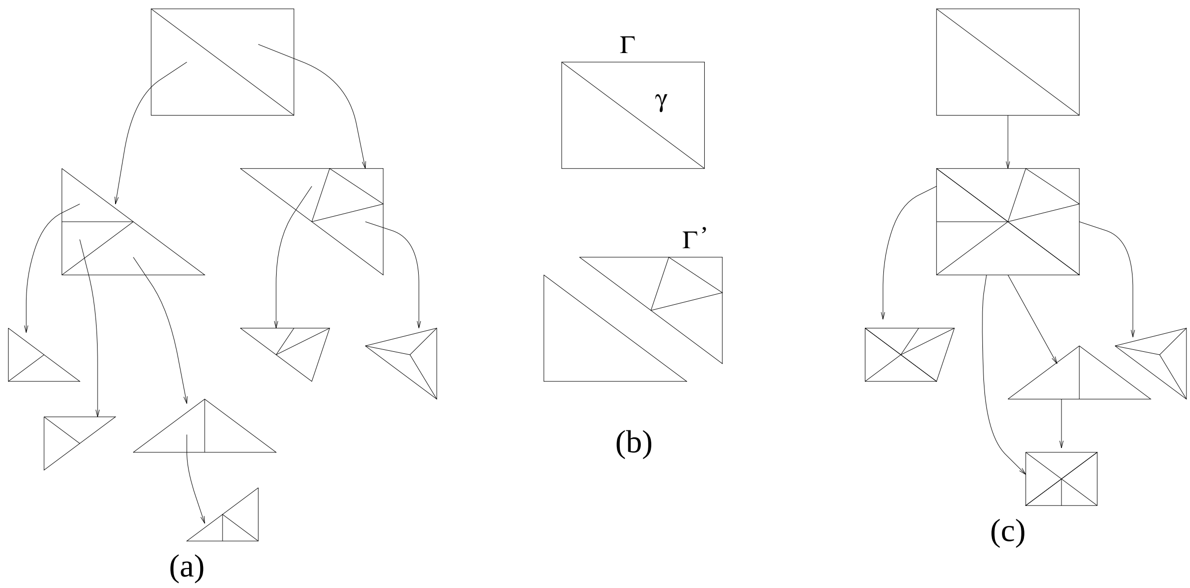


Figure 4.25: (a) The tree representing a nested model. (b) The complex Γ' refining a top-cell γ does not define a valid modification pattern for the complex Γ containing γ; in fact the boundary of Γ' does not match with that of the adjacent cell to γ in Γ. (c) Clusters of tree nodes that become MC-nodes when interpreting the nested model as an MC.

complexes can be obtained from it (i.e., the domain decompositions that avoid cracks in the graph of a scalar or vector field represented by the nested model, see Section 3.3.2). The set of cell complexes available from the model is in one-to-one correspondence with the MC-fronts of the corresponding Multi-Complex. The set of domain decompositions (possibly not complexes) obtained from a nested model is in one-to-one correspondence with the cuts of the tree. Since several nodes of the tree are merged into one MC-node, it follows that some cuts of the tree are not captured by the MC: these are exactly the cuts leading to inconsistent representations.

Since no knowledge about different tree nodes that must be clustered to form an MC-node is explicitly encoded in the tree, algorithms for extracting generic (i.e., variable resolution) complexes from nested models are more involved.

Nested models based on the recursive subdivision of a domain according to a fixed pattern that splits *every* $(k-1)$-face of the current k-cell (e.g., quadtrees and their extensions [Sam90a], quaternary triangulations [GG79]), correspond to Multi-Complexes where the DAG reduces to a list. In fact, all the nodes in a level of the tree must be merged together into one MC-node. Thus, the set of cell complexes that can be obtained from such models reduces to one for every tree level, and the expressive power is similar to that of layered models.

Nested models that leave some $(k-1)$-face of a k-cell γ unrefined when expanding γ into a complex (e.g., hierarchical triangulations [SP92, DFP95a], hierarchies of right triangles [DWS$^+$97, EKT97, HK95, LKR$^+$96]) may have a better expressive power. In general, if $(k-1)$-

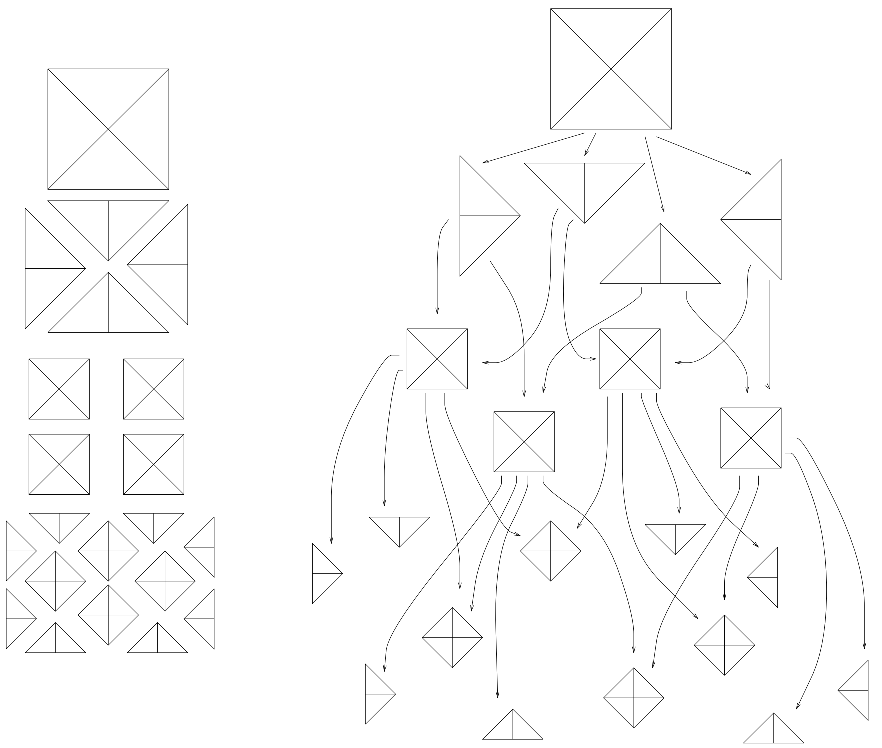


Figure 4.26: (a) A hierarchy of right rectangles, with meshes shown level by level, and (b) the clusters of tree nodes corresponding the nodes of the Multi-Complex.

cells are split too often, then the nodes of the tree tend to be clustered into large MC-nodes, hence allowing only a reduced number of MC-fronts, and a low expressive power. On the other hand, the persistence of $(k-1)$-cells across the levels of the tree tends to produce slivery cells, which are not desirable. A limit example is provided by the ternary triangulation [DFFPN84], a model based on the recursive splitting of a triangle into three triangles by inserting a new vertex inside it: for such model the tree and the MC have the same structure; nevertheless, the model was soon abandoned for the bad quality of meshes it provided. The best compromise seems to have been achieved with hierarchies of right triangles, where, at each level, one edge of each triangle is split and two are retained to be split at the next level.

In the Multi-Complex interpreting a hierarchy of right rectangles (shown in Figure 4.26), for each MC-node C, $\Pi_1(C)$ is formed by two right triangles and $\Pi_2(C)$ by four right triangles (except for MC-nodes lying at the boundary of the domain). The domains of all such meshes have either the shape of a square or of a diamond (a right triangle for border MC-nodes). The model of Duchaineau et at. [DWS+97] for a hierarchy of right triangles resembles the corresponding MC very closely; however, the concepts of modification pattern and partial order are not so explicitly remarked.

It is easy to verify that the MC corresponding to a hierarchy of right triangles has linear growth, bounded width, and logarithmic height.

4.7.3 Evolutionary Models

Evolutionary models track the evolution of an initial cell complex Γ_0 through a sequence of local modifications, which is assumed to be a valid sequence for Γ_0.

A Multi-Complex for such models represents the local modifications of the sequence in its nodes, and encodes in its arcs their relations of direct dependency: there is an arc from a node $C_i = (\Gamma_i, \Gamma'_i)$ to a node $C_j = (\Gamma_j, \Gamma'_j)$ if and only if the intersection of Γ'_i and Γ_j (as sets of cells) contains at least one top-cell. The validity and non-redundancy of the sequence ensures that the resulting graph is a DAG.

In most existing models, the local modifications are performed by applying a specific operator, for instance:

- In [Hop96, XV96, XESV97, Hop97, MMS97, GTLH98, KCVS98] the refinement pattern is a vertex split, which expands a vertex p into an edge, two edges incident at p into two triangles, and deforms the surrounding triangles accordingly.
- In [DF89, CDFM+94, dBD95, BDFM95b, KS97] the refinement pattern consists of inserting a new vertex p, and replacing a subset of triangles in the neighborhood of p with a set of triangles incident at p (determined according to various criteria). Note that a vertex split where the edge created by splitting p has p itself as one of its endpoints can be regarded as a special case of vertex insertion.

Finally, some models do not rely on any specific type of modification patterns [Pup96, Pup98, BDFM95b, CCMS97, CPS97, CMPS97, CMRS97].

In all models, the local modifications guarantee the non-redundancy and the minimality of the modifications, except in Delaunay pyramids [DF89, BDFM95b, dBD95] and in [GHJ+97].

Evolutionary models proposed in the literature represent the local modifications and/or the cells involved within them by using different structures, ranging from sequential lists, to DAGs. In the following we illustrate the relationship between those structures and the corresponding Multi-Complex.

4.7.3.1 Sequential Models

Sequential models [KS97, Hop96] keep the local modifications in a totally ordered list which is consistent with the partial order represented in the Multi-Complex. Usually, such total order is either the same or the reverse of the order in which the local modifications have been performed during the construction of the model. Dependencies between local modifications are not explicitly stored, but they are reconstructed on-line when extracting a complex at variable resolution. Usually, the algorithms for such task are rather involved. Thus, sequential models can be seen as simplified versions of an MC, containing less information.

In terms of the corresponding MC, the models of [KS97] and [Hop96] do not guarantee bounded width, logarithmic height or linear growth.

4.7.3.2 Delaunay Pyramids

The modifications in a Delaunay pyramid [BDFM95b, dBD95] consist of updating a current Delaunay triangulation by inserting a set of vertices. Such modifications are, in general, not minimal, but each of them can be split into minimal modifications which are independent and conflict-free.

The Delaunay pyramid of [BDFM95b] stores the triangles appearing in the various levels as well as interference links between triangles which appear in two consecutive levels, and whose interiors have a non-empty intersection. In this representation, a node of the Multi-Complex is a maximal subset of triangles belonging to consecutive levels, that are reachable from each other by following interference links upwards and downwards (see Figure 4.27). In fact, for an MC-node $C = (\Gamma_{old}, \Gamma_{new})$, Γ_{old} and Γ_{new} are two mutually exclusive choices for describing the same portion of an object: for two-dimensional MCs in $\mathbb{E}^2$, this property implies that the domains of Γ_{old} and Γ_{new} overlap. In [BDFM95b], the overlapping relation is represented between individual pairs of triangles of Γ_{old} and Γ_{new}. This leads to an explosion of the size of the structure: in the worst case, a quadratic number of arcs in the size of Γ_{old} and Γ_{new} may be necessary.

The model of de Berg and Dobrindt [dBD95] also stores the triangles appearing in the various levels, but it also represents each local modification (insertion of a vertex p) explicitly, by linking p to the triangles present in the mesh immediately before, and to those present immediately after its insertion (see Figure 4.28 (b)). This concept directly corresponds to a modification pattern as a pair of cell complexes. In fact, the DAG used in [dBD95] reflects the structure of the corresponding Multi-Complex. For instance, the three sibling nodes corresponding to points p_1, p_2, p_3 in Figure 4.28 (b) become three sibling MC-nodes in Figure 4.28 (c).

The MC for [BDFM95b] might not have linear growth, bounded width, and logarithmic height. In [dBD95], each level is obtained from the previous by eliminating a maximal set of independent vertices of bounded degree. Since every vertex is eliminated independently, it generates an

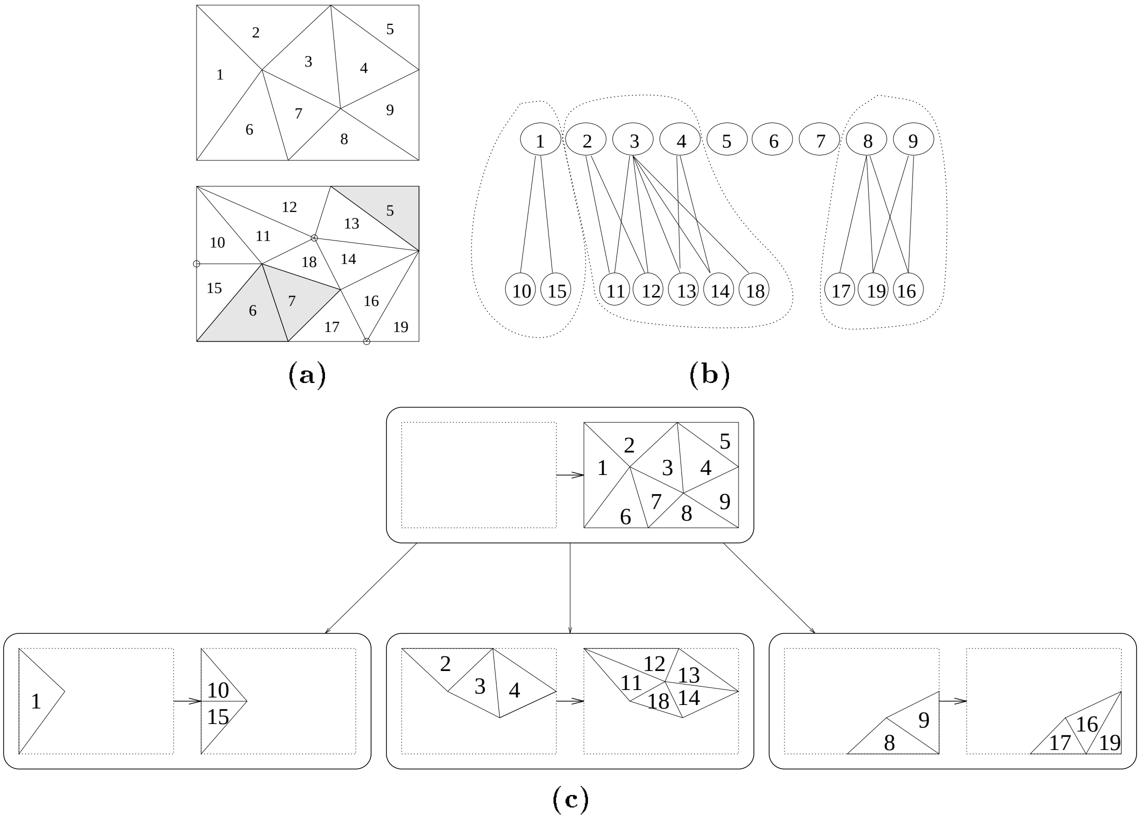


Figure 4.27: (a) A Delaunay pyramid with two levels; (b) the graph stored in [BDFM95b] and, circled with dotted lines, the groups of its nodes forming the same MC-node; (c) the corresponding MC.

individual MC-node. The corresponding MC has linear growth, bounded width, and logarithmic height. The reasons will be explained in more details in Section 7.5.2.

4.7.3.3 Hierarchies of Vertices

Hierarchies of vertices identify a local modification (i.e., a vertex split) with the vertex v being split. Thus, dependencies between modification patterns become dependencies between vertices. Most models (e.g., [Hop97, XV96, XESV97]) distinguish between two cases of dependency:

1. if a vertex v is an endpoint of the edge created by splitting w, then v depends on w;
2. v also depends on (some of) the vertices that are incident in v when v is split.

Dependencies of type 1 are stored as a binary tree of vertices (every vertex split creates two new vertices, which become its children, see Figure 4.29 (b)); dependencies of type 2 are stored

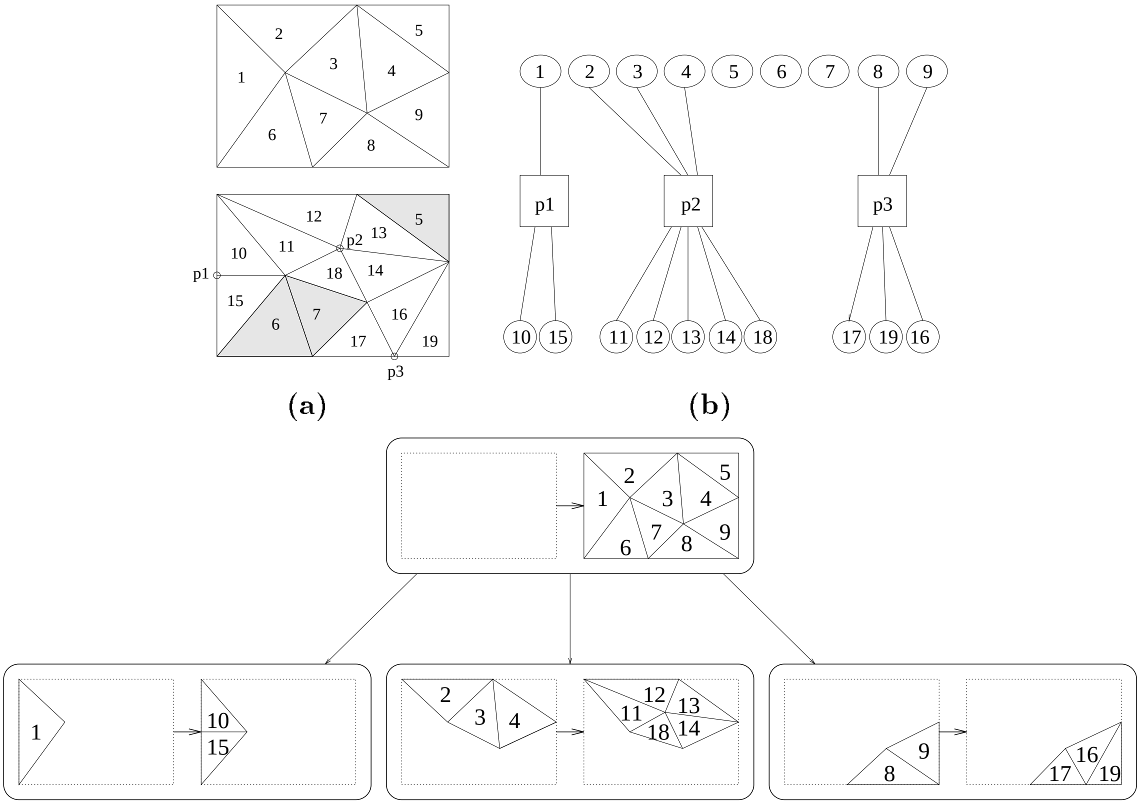


Figure 4.28: (a) A Delaunay pyramid with two levels, the upper one obtained from the lower one by removing vertices p_1, p_2, p_3; (b) the graph stored in [dBD95]; (c) the corresponding MC.

separately. The various authors define dependencies of type 2 in different ways.

Xia and Varshney [XV96, XESV97] consider a vertex v as dependent of all the vertices that are adjacent to v when v is split. With respect to the Multi-Complex, the resulting DAG contains redundant arcs which correspond to paths in the MC (see Figure 4.30 (b)).

Hoppe [Hop97] considers a vertex v as dependent just on the vertices of the four faces incident at the two edges that are expanded into triangles when v is split (see Figure 3.18). However, this does not capture all the existing dependencies and there are still redundant arcs (see Figure 4.30 (c)).

Progressive TINs [MMS97] consider a vertex p as dependent from its parent in the binary tree, and from the other endpoints of the two edges that are expanded into triangles when splitting p. Again, some dependencies are missing, and some redundant arcs are present (see Figure 4.30 (d)).

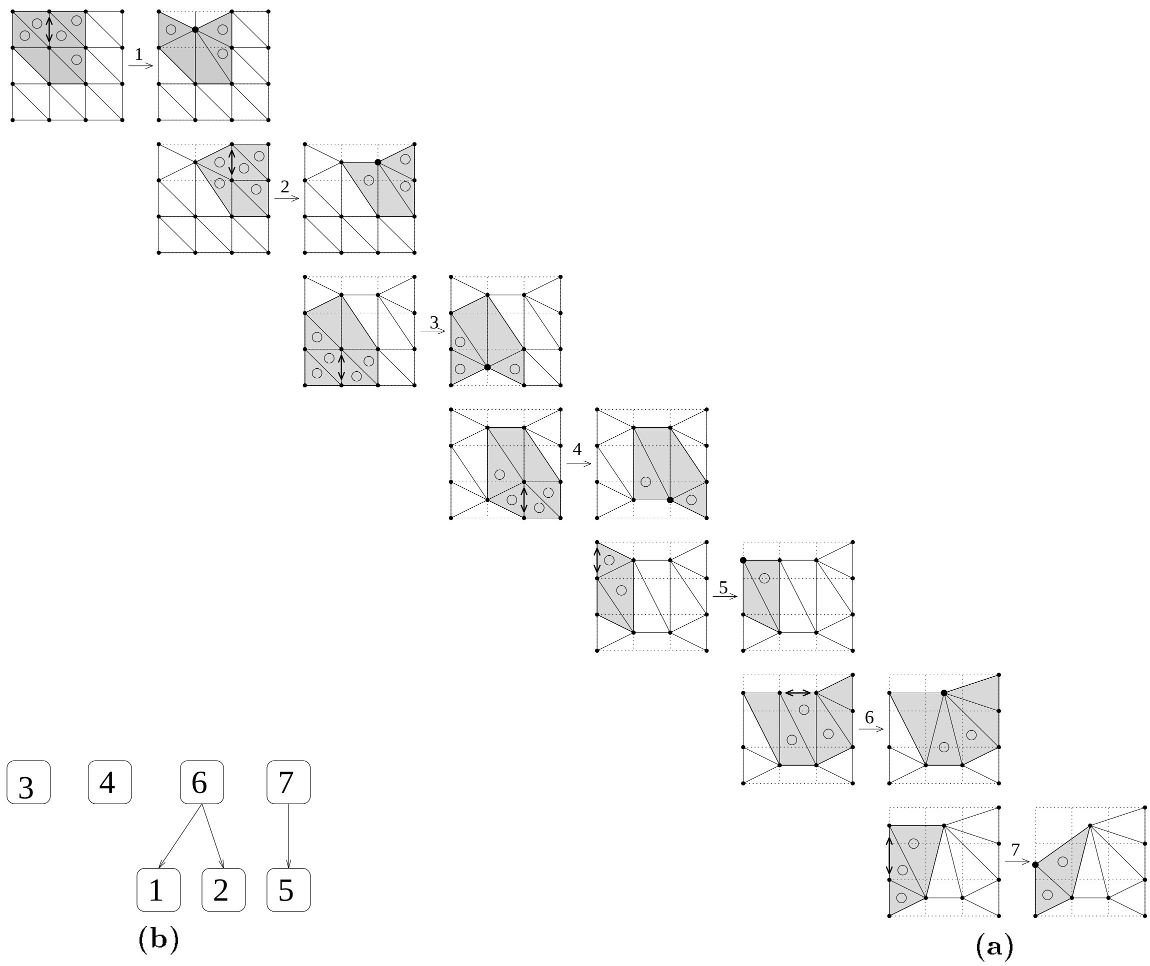
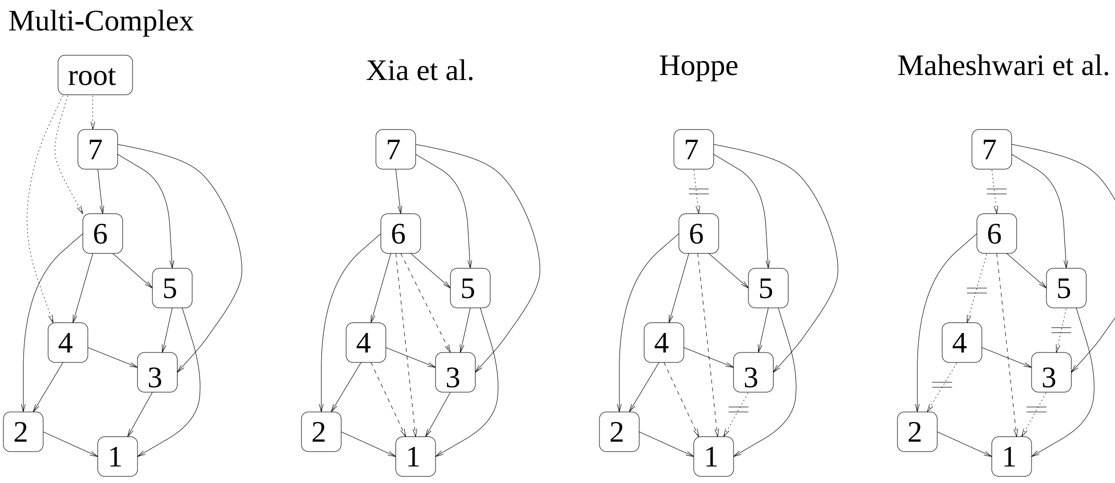


Figure 4.29: (a) A sequence of edge collapses, to be read reversed as a sequence of vertex splits. Shaded triangles correspond to the two meshes forming a modification pattern in the MC. Triangles marked with a circle are those considered in [Hop97] for determining the vertex dependencies. Shaded triangles are also those considered for defining vertex dependencies in [XV96, XESV97]. (b) The corresponding binary forest of vertices.

The model by Luebke and Erikson [LE97] is based on modifications that are "generalized" vertex splits: a vertex may be split into two vertices not necessarily connected by a segment (such an operation is the inverse of not only edge collapse but also vertex-pair collapse, when the two collapsed vertices are not adjacent in the mesh). This model keeps just the binary tree of vertices and ignores all other existing dependencies.

Guéziec et at. [GTLH98] store vertex dependencies explicitly by means of a DAG, but they consider dependencies as in [XV96, XESV97], thus their DAG contains some redundant arcs.

The problem of redundant arcs is intrinsic of the binary tree of vertices: a vertex split may depend only indirectly on its parent, since other modifications may have altered the configuration



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

Figure 4.30: (a) The Multi-Complex corresponding to the sequence of edge splits in Figure 4.29; the arcs emanating from the root are shown dashed since the root does not represent a vertex splits. (b,c,d) The DAG representing vertex dependencies according to [XV96, XESV97], to [Hop97] and to [MMS97], respectively. Redundant arcs are shown dashed, missing arcs are dotted and marked with '='.

of triangles between the parent and the child vertex splits. For instance, the binary tree shown in Figure 4.29 contains arc $(6,1)$ which is not in the corresponding MC shown in Figure 4.30.

Extra arcs increase the space requirements of data structures but do not compromise the integrity of the model. On the contrary, missing arcs are a problem because DAG traversal algorithms working on such models may extract inconsistent representations.

Models affected by missing dependency links [Hop97, LE97, MMS97] guarantee that any extracted cell complex is consistent topologically, but it may contain intersecting triangles and triangles which were not present in the original sequence of cell complexes that were generated during the construction of the model. Since the models of [Hop97, LE97] are designed to support real-time visualization, the problem of "extra" triangles is not so relevant, but it would be relevant in other applications where it is necessary to know the approximation error of any triangle with respect to the target object; in fact, the approximation error of extra triangles is not known.

The models of [Hop97, GTLH98, MMS97] do not guarantee bounded width, logarithmic height or linear growth. The Multi-Complex corresponding to the model of [XV96, XESV97] has linear growth, and logarithmic height, but it does not necessarily have bounded width.

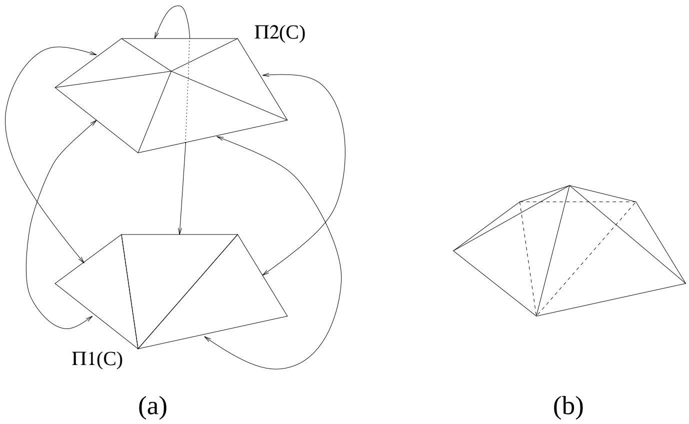


Figure 4.31: (a) A modification pattern C with arrows between $\Pi_1(C)$ and $\Pi_2(C)$ representing counter-adjacent$_{1,2}$ links (the upward direction of the arrow) and counter-adjacent$_{2,1}$ links (the downward direction of the arrow); (b) its higher-dimensional embedding.

4.7.3.4 The Hyper-Triangulation

The Hyper-Triangulation [CPS97, Cig97] assumes that, for any modification pattern in an MC, the boundary complexes of $\Pi_1(C)$ and $\Pi_2(C)$ are coincident (as sets of cells). It defines dependencies between local modifications in the same way as they are defined for the Multi-Complex, but represents them through adjacency relations rather than using a DAG. The Hyper-Triangulation can be seen as a special implementation of the MC that represents the DAG by embedding the local modifications of an MT into a higher dimensional space and providing the four adjacency relations co-adjacent$_1$, co-adjacent$_2$, counter-adjacent$_{1,2}$, and counter-adjacent$_{2,1}$, defined in Section 4.5 (see Figure 4.31). A data structure for MCs based on the same principle as the Hyper-Triangulation is described in Section 6.3.3.

4.8 Summary

In this chapter, we have defined the Multi-Complex (MC), a general model for multiresolution cell complexes. An MC consists of a DAG that represents a partially ordered set of modifications refining an initial cell complex; the partial order is induced by the relation of direct dependency, where a modification that deletes some top-cells from the current complex directly depends on those modifications that have previously introduced such top-cells.

Cell complexes at variable resolution can be obtained from an MC by selecting a subset of modifications consistent with the partial order. We have provided an explicit characterization of the set of cell complexes available from an MC, by showing a one-to-one correspondence between them and special cuts, called MC-fronts, of an enriched DAG associated with an MC.

We have shown that the Multi-Complex encompasses most existing multiresolution models and helps understanding the range of cell complexes at variable resolution that a model can provide. Thus, the MC acts as a unifying framework that is useful for comparing the expressive power of different approaches. As a side effect, it follows that the class of dependency-based multiresolution models, which contains the MC, is the most general one. However, the MC is limited to manifold cell complexes. The extension of the MC to non-manifold complexes will be object of future work.

Chapter 5

Querying a Multiresolution Cell Complex

5.1 Introduction

In this chapter, we address the problem of determining the cell complexes, among all those available from an MC, which adapt best to the requirements of a given application task. The ideal aim for an application task is to work, at each instant in time, with a cell complex whose approximation error and/or size are maintained within some given bound.

We define two general query operations on an MC: a *global query*, which allows the extraction of a cell complex representing the whole target object according to certain resolution requirements, and a *local query*, which allows the selection of a subcomplex representing an area of interest according to certain resolution requirements. Such fundamental operations are the basis for defining operations typical of each application; examples of application-dependent operations, implemented on top of the query operations defined here, can be found in Chapter 8.

For global queries, we provide algorithms that find an optimal solution based on a top-down traversal of the DAG, and dynamic algorithms that find a sub-optimal solution by updating the result of a previous query. For local queries, we provide a heuristic approach based on a top-down traversal in which some branches of the DAG are cut away.

This chapter is organized as follows:

- In Section 5.2, we introduce the entities that will be used as parameters for querying an MC: resolution filters (which specify the resolution at which a query must be answered) and focus sets, which play a role in local queries.
- In Section 5.3, we define the two basic *query operations* on a Multi-Complex: the global and the local query; for each query we define two versions: one that optimizes the size of

the answer for a given resolution filter, and one that optimizes the closeness to a resolution filter within a given upper bound on the size.

- In Section 5.4, we present algorithms for the two versions of the global query and prove their correctness.

- In Section 5.5, we propose a sub-optimal algorithm for local queries. Such algorithm does not consider all dependency links present in the partial order of an MC, but only those that involve portions of the complex interfering with a focus set. Such an approach outputs a cell complex only in the case of conservative Multi-Complexes (see Section 4.4.2), but it can be extended to a heuristic to other MCs.

- In Section 5.6, we propose *dynamic algorithms* for global queries on a Multi-Complex, that, unlike the ones proposed in Section 5.4, compute the solution of a new query by updating that of a previous query. Dynamic algorithms are correct with respect to a slightly weaker requirements than those formulated in Section 5.3.

- In Section 5.7, we provide an experimental comparison of the three query algorithms proposed in Sections 5.4, 5.5 and 5.6.

- In Section 5.8, we present a specification of the operations acting on a Multi-Complex, from high-level query operations (corresponding to the user-interface of an MC) to low-level kernel operations (corresponding to the interface of the data structure used to represent an MC).

5.2 Preliminaries

Since a multiresolution model for a geometric object $\mathcal{O}$ provides several descriptions of $\mathcal{O}$, a basic query operation consists of extracting a cell complex, selected among those available in the multiresolution model, which represents $\mathcal{O}$ according to some user-defined requirements. The user requirememts define the *desired resolution* (either directly, or by specifying a desired accuracy which, in turn, determines a desired resolution), and/or the *acceptable size* of the returned cell complex. A *global query* on an MC corresponds to such an idea.

Queries can also be defined on a multiresolution model which extend usual queries performed on single-resolution cell complexes by adding the possibility of selecting the resolution, the accuracy and/or the size of the cell complex on which the query is answered. *Local queries* on an MC correspond to such an idea.

Before defining the queries (see Section 5.3), here we formally introduce their input paramenters.

5.2.1 Resolution Filters

We express resolution requirements for a (global or local) query on a Multi-Complex by means of a *resolution filter*. A k-dimensional *resolution filter* in $\mathbb{E}^d$ is a function $R : \Gamma_d^{(k)} \longrightarrow \mathbb{R}$.

A resolution filter assigns to each k-cell γ a real value $R(\gamma)$ that depends on the geometry of γ (e.g., on its size) and/or on its approximation error. A resolution filter measures the "signed difference" between the resolution or accuracy of a cell and that needed by an application. A value of $R(\gamma) > 0$ indicates that the resolution of γ is not sufficient; $R(\gamma) < 0$ means that the resolution of γ is higher than necessary. A value less than, or equal to, zero identifies cells that satisfy the application requirements. Such cells are called *feasible*:

Definition 5.1 *Given a resolution filter $R : \Gamma_d^{(k)} \longrightarrow \mathbb{R}$, a k-cell γ is* feasible *with respect to R if $R(\gamma) \leq 0$.*

In the following, we give some examples of resolution filters that are of interest for applications.

As a first example, let us consider an application that requires a *uniform resolution*: the resolution $\rho(\gamma)$ of a cell γ must be at least r, where $r \in \mathbb{R}^+$. The corresponding resolution filter is defined as:

$$R_r(\gamma) = \rho(\gamma) - r$$

Similar resolution filters are used in surface visualization, where instead of considering the size of a cell (recall from Section 2.3.7 that the resolution of a cell is the inverse of its size), the size of its projection onto the viewplane is considered. The basic idea here is that it is worthless to render individual cells whose image is smaller than a few pixels.

Still in surface visualization, some authors [Hop97, LE97] use resolution filters that impose more strict constraints on cells where the local surface normal is perpendicular to the view direction. Intuitively, such cells contribute to the silhouette of the surface, which is an important feature for shape recognition from an image.

An interesting class of resolution filters is that of filters *induced by error threshold functions*. Intuitively, the resolution filter accepts a cell γ if its approximation error $\varepsilon(\gamma)$ is not larger than the threshold function. Given an error threshold function τ, the resolution filter R_τ associated with τ is defined as:

$$R_\tau(\gamma) = \varepsilon(\gamma) - \min\{\tau(p) \mid p \in \gamma\}$$

A resolution filter can be *restricted to an area of interest*. Let $I \subseteq \mathbb{E}^d$ be a spatial entity defining an area of interest (e.g., I may be an axis-parallel box in d-dimensions). Let R be an arbitrary resolution filter. A resolution filter $R'_{I,R}$ can be defined with restricts the influence of filter R to the area of interest represented by I:

$$R'_{I,R}(\gamma) = \begin{cases} R(\gamma) & \text{if and only if } \gamma \cap I \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

For instance, if R is induced by a constant error threshold function equal to 0, then $R'_{I,R}$ requires a null approximation error on cells interesecting the area of interest I and accepts any error on cells disjoint from I.

In the reminder of the thesis, sometimes an area of interest will be called a focus set, for analogy with focus sets used in interference query (see Section 5.2.2).

5.2.2 Interference Queries

Local queries on an MC extend standard queries on a cell complex at single resolution. Thus, they share the same input parameters.

A generic query on a k-dimensional cell complex Γ consists of *selecting* a sub-complex of Γ formed by the cells of Γ that have certain properties. In this thesis, we consider *geometric queries*, i.e., queries based on geometric properties only. Geometric queries on a cell complex Γ are concerned with:

- *set-theoretic properties*, i.e., selecting the cells of Γ that coincide, contain, are contained in, properly intersect, are tangent to, are disjoint from, etc., a given geometric entity;
- *metric properties*, i.e., selecting the cells of Γ that are at, within, out of a certain distance d from a given geometric entity;
- *directional properties*, i.e., selecting the cells of Γ that lie in a certain direction (e.g., to the right of, to the left of, in front of, behind of above, or below) a given entity with respect to a reference system.

Among the three classes above, *set-theoretic queries* are the most general ones.

Metric queries reduce to set-theoretic queries with respect to the given entity enlarged by distance d: a cell lies at a distance equal to, less than, or larger than d from the original entity if it is tangent to, intersects, or is disjoint from the enlarged entity, respectively (see Figure 5.1 (a)). Directional queries reduce to set-theoretic queries by considering the intersection with the locus of half-lines emanating from each point of the given entity and directed in the given direction (see Figure 5.1 (b)).

In this thesis, we restrict our attention to a special case of set-theoretic queries, namely those based on *interference relations*. Two geometric entities in $\mathbb{E}^d$ *interfere* if they have a non-empty intersection.

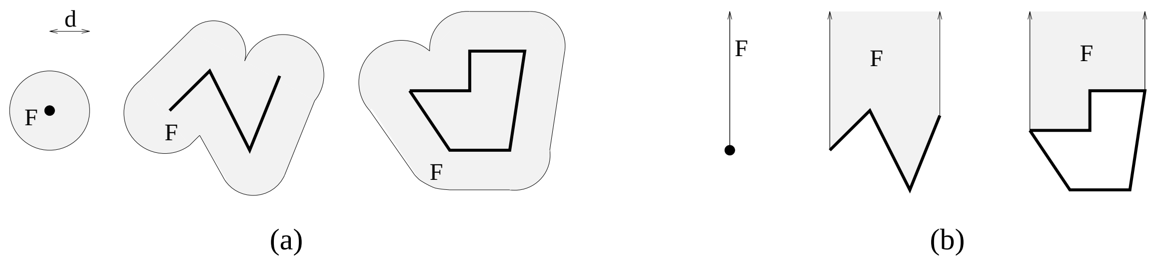


Figure 5.1: (a) Reducing metric queries with respect to a geometric entity to interference queries with respect to the enlarged entity; (b) reducing directional queries to interference queries with respect to the locus of half-lines emanating from the entity in the given direction (in this case, north). The new query entity F to be used in the interference query is shown in grey.

Interference queries represent the most general case of set-theoretic queries. All set-theoretic queries involving a non-empty intersection with two geometric entities can be viewed as special cases of interference. The relation of disjointness can be expressed by considering the interference of one of the two objects involved in the relation with the complement in $\mathbb{E}^d$ of the other one.

Interference queries on a cell complex Γ in $\mathbb{E}^d$ can be expressed parametrically based on a given entity F in $\mathbb{E}^d$, that we call a *focus set*. For a given focus set F, a query selects the cells of Γ which have a non-empty intersection with F. For convenience, we will call such cells *active*:

Definition 5.2 *A cell γ in $\mathbb{E}^d$ is* active *with respect to a focus set $F \subseteq \mathbb{E}^d$ if $\gamma \cap F \neq \emptyset$.*

The result of an interference query on a k-dimensional cell complex Γ with a focus set F is the cell complex $\Gamma' \subseteq \Gamma$ formed by all the cells $\gamma \in \Gamma$ (of any dimension) which interfere with F and by all their faces in Γ. Note that, since cells are closed sets, the resulting complex Γ' is always a regular cell complex.

5.3 Query Operations on Multiresolution Complexes

In this section, we formally define global and local queries on an MC.

The basic parameter necessary for querying an MC (both in the global and in the local case) is a resolution filter. In addition, an upper bound on the size of the returned cell complex may be given. These parameters determine the resolution at which the query will be answered. There are two opposite issues to trade off:

- answering a query on a representation having a *sufficient resolution* to satisfy certain requirements (expressed either directly in terms of resolution or in terms of accuracy);

- maintaining the *size* of the cell complex reported as the answer to the query within a given upper bound.

These two issues are usually in conflict and cannot, in general, be fullfilled at the same time. So, two variants of the queries can be considered (which were formulated in [CVM+96] in the context of mesh simplification):

1. finding the answer of *minimum size* which satisfies a certain resolution filter, or
2. finding the answer which *best approximates* the requirements of a resolution filter and has a size less or equal to a given upper bound.

Variant 1 is useful in those applications where guaranteeing the desired accuracy of representation is of major importance. Variant 2 is useful in applications where efficiency is a major concern (for instance, an application task may be able to process, e.g., render, only cell complexes up to a certain size). In the reminder, variants 1 and 2 will be called the *min-size* and the *best-resolution* variant, respectively. The min-size and the best-resolution variant apply both to the global and to the local query. The local query has an additional parameter consisting of a focus set.

5.3.1 Global Queries on a Multi-Complex

Global queries on a Multi-Complex solve the problem of extracting a cell complex representing the whole target object according to some requirements in terms of resolution and size. The input parameter for a global query on a k-dimensional Multi-Complex $\mathcal{M}$ in $\mathbb{E}^d$ are:

- a *k-dimensional resolution filter* $R : \mathbb{\Gamma}_d^{(k)} \longrightarrow [0, +\infty)$, which establishes whether the resolution of a cell is sufficient for the application performing the query;
- in the best-resolution variant only, an upper bound $b \in \mathbb{N}$, $b > 0$ on the size of the extracted cell complex.

Informally, the result of the global query is a cell complex Γ_{glob} representing the whole target object of $\mathcal{M}$ and such that:

- in the min-size variant, Γ_{glob} has the *minimum size* necessary and sufficient to guarantee that all its cells are feasible with respect to the given resolution filter R.
- in the best-resolution variant, Γ_{glob} gives the best approximation of the resolution constraints set by R, which can be achieved with a number of cells less than or equal to the given bound b.

The search space for the solution of a global query is the set of *global combined cell complexes* associated with $\mathcal{M}$, since these are all possible consistent representations of the whole target object which can be obtained from $\mathcal{M}$ (see Section 4.4).

Definition 5.3 *Given a k-dimensional Multi-Complex $\mathcal{M}$, and a k-dimensional resolution filter R, the result of the* minimum-size variant *of the* global query *is a cell complex Γ_{glob} such that:*

1. *Γ_{glob} is a global combined cell complex of $\mathcal{M}$;*
2. *all k-cells of Γ_{glob} are feasible with respect to the resolution filter R;*
3. *the size of Γ_{glob} is minimum among all the cell complexes that satisfy conditions 1 and 2 above.*

Definition 5.4 *Given a k-dimensional Multi-Complex $\mathcal{M}$, an integer number $b > 0$, and a k-dimensional resolution filter R, the result of the* best-resolution variant *of the* global query *is a cell complex Γ_{glob} such that:*

1. *Γ_{glob} is a global combined cell complex of $\mathcal{M}$;*
2. *the number of k-cells in Γ_{glob} is less than or equal to b;*
3. $\max\{R(\gamma) \mid \gamma \in \Gamma^{(k)}_{glob}\}$ *is minimum among all the cell complexes that satisfy conditions 1 and 2 above.*

Condition 3 corresponds to the intuition of being as close as possible to the requirements imposed by the resolution filter R. In fact, since R assigns larger values to cells that are more distant from the requirements, then minimizing the maximum of R means minimizing the distance from the desired resolution (in case it cannot be achieved because of the size constraint).

5.3.2 Local Queries on a Multi-Complex

Local queries are concerned with the problem of selecting from an MC a cell complex *interfering with a focus set* according to some requirements in terms of resolution and size. The input parameters for a local query on a k-dimensional Multi-Complex $\mathcal{M}$ in $\mathbb{E}^d$ are:

- a focus set $F \subseteq \mathbb{E}^d$;
- a *k-dimensional resolution filter* $R : \mathbb{\Gamma}^{(k)}_d \longrightarrow [0, +\infty)$;

- in the best-resolution variant only, an upper bound $b \in \mathbb{N}$, $b > 0$ on the size of the output cell complex.

Informally, the result of a local query is a cell complex Γ_{loc} made of cells of $\mathcal{M}$ which are active with respect to F (i.e., which describe the portion of the target object interfering with F), and:

- in the min-size variant, Γ_{loc} has the *minimum size* necessary and sufficient to guarantee that all its cells are feasible with respect to the given resolution filter.
- in the best-resolution variant, Γ_{loc} gives the best possible approximation of the given resolution constraints on its cells which can be achieved with a size less than or equal to the given bound.

The search space for the solution of the query is the set of (partial) *combined cell complexes* Γ_{loc} of $\mathcal{M}$ which have the following properties:

1. all top cells of Γ_{loc} are active;
2. Γ_{loc} can be completed to a global combined cell complex by adding just non-active cells.

Condition 1 expresses the fact that Γ_{loc} does not represent portions of the object other than those interfering with the focus set; Condition 2 means that Γ_{loc} covers the portion interfering with the focus set completely.

Definition 5.5 *Given a k-dimensional Multi-Complex $\mathcal{M}$, a k-dimensional resolution filter R, and a focus set $F \subseteq \mathbb{E}^d$, the result of the* minimum-size variant *of the* local query *is a cell complex Γ_{loc} such that:*

1. *Γ_{loc} is a combined cell complex of $\mathcal{M}$;*
2. all k-cells *of Γ_{loc} are active with respect to F;*
3. *Γ_{loc} can be completed to a global combined cell complex by adding just non-active cells;*
4. all k-cells *of Γ_{loc} are feasible with respect to the resolution filter R;*
5. *the size of Γ_{loc} is minimum among all the cell complexes that satisfy conditions 1,2,3,4 above.*

Definition 5.6 *Given a k-dimensional Multi-Complex $\mathcal{M}$, an integer number $b > 0$, a k-dimensional resolution filter R, and a focus set $F \subseteq \mathbb{E}^d$, the result of the* best-resolution variant *of the* local query *is a cell complex Γ_{loc} such that:*

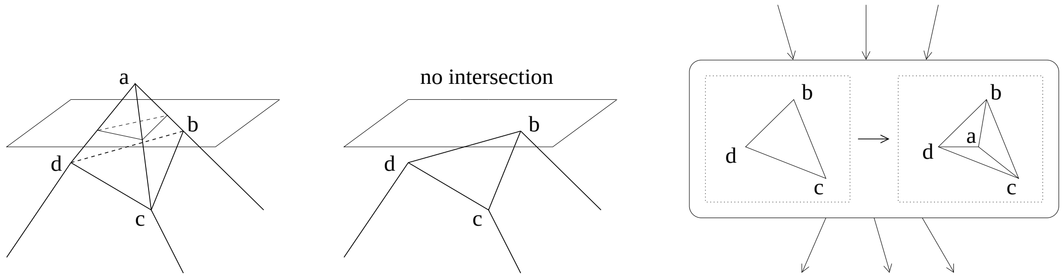


Figure 5.2: The focus set is a plane. The MC contains a modification pattern replacing cell *bcd* (not active) with *abc*, *acd*, *adb* (active). The complex at the left is over-refined according with the definition of the result of a local query. The complex on the right is not over-refined and satisfies the requirements, but the intersection is lost.

1. *Γ_{loc} is a combined cell complex of $\mathcal{M}$;*
2. *all k-cells of Γ_{loc} are active with respect to F;*
3. *Γ_{loc} can be completed to a global combined cell complex by adding just non-active cells;*
4. *the number of k-cells in Γ_{loc} is less than or equal to b;*
5. *$\max\{R(\gamma) \mid \gamma \in {\Gamma_{loc}}^{(k)}\}$ is minimum among all the cell complexes that satisfy conditions 1,2,3,4 above.*

5.3.3 Strict and Loose Interference

Let Γ be a cell complex at *single resolution* representing a geometric object $\mathcal{O}$; the interference of a cell $\gamma \in \Gamma$ with a subset S of $\mathbb{E}^d$ means that (based on the knowledge available in the model) the portion of $\mathcal{O}$ represented by γ intersects S.

In a Multi-Complex $\mathcal{M}$, the knowledge contained in the model includes an approximation error associated with each cell γ; if $\varepsilon(\gamma) > 0$, a more accurate representation of the portion $\mathcal{O}_\gamma$ of $\mathcal{O}$ represented by γ may be available in $\mathcal{M}$. We define two notions of interference in a Multi-Complex:

1. *(strict) interference*: γ intersects S;
2. *loose interference*: γ may or may not intersect S but it lies at a distance $\leq \varepsilon(\gamma)$ from it; thus, a more accurate representation of $\mathcal{O}_\gamma$ may be available in $\mathcal{M}$ which intersects S.

We recall that the approximation error $\varepsilon(\gamma)$ provides an upper bound on the distance of γ from the portion of $\mathcal{O}$ it represents (see Section 2.4.3). The distance used for defining the loose interference is measured in the same (application-dependent way) used to measure the approximation error of γ.

The notion of strict interference is the common notion of interference. Loose interference means that the portion of target object approximated by γ *may* intersect S, but the accuracy at γ is not sufficient to decide whether it does or not.

Local queries can be defined on a Multi-Complex by considering a cell as active based either on a strict or on a loose notion of interference.

Definition 5.7 *Let $\mathcal{M}$ be a k-dimensional Multi-Complex in $\mathbb{E}^d$, and $F \subseteq \mathbb{E}^d$ be a focus set. A k-cell γ of $\mathcal{M}$ is:*

- (strongly) active*, if $\gamma \cap F \neq \emptyset$;*
- loosely active*, if the distance between γ and F is less than or equal to $\varepsilon(\gamma)$.*

Strongly active cells are the same as active cells of Definition 5.2. If a cell is (strongly) active, then it is also loosely active, but the converse is not necessarily true.

Local queries defined in Section 5.3.2 can be reformulated by simply adding the term "strictly" or "loosely" before each occurrence of term "active".

A strict local query may return an empty cell complex because coarse representations of the target object available in $\mathcal{M}$ do not intersect the focus set. In this case, an empty complex satisfies the requirements of Definition 5.5 or 5.6: it is still possible that representations at higher resolutions intersect the focus set, but they are not of minimum size according to Definition 5.5. For the best-resolution version, such representations may satisfy Definition 5.6, thus the solution in this case is not unique: a criterion that chooses the acceptable solution of minimum size (which seems reasonable in general) will accept the empty solution.

An example is shown in Figure 5.2 for a two-dimensional MC representing a surface in $\mathbb{E}^3$. In the complex at low resolution, all active cells are feasible (in this case, there are no active cells), and thus it is the result of the local query: the target surface intersects the focus set, but such intersection is not reported.

In some applications (e.g., collision detection, extraction of isosurfaces or isolines), it is important to consider in any case a level of resolution sufficient to find an intersection, if there exists one. For this aim, use a loose local query. In Figure 5.2, triangle $\sigma \equiv bcd$ may or may not be loosely active depending on its approximation error. Assuming that the target surface passes through point a, then $\varepsilon(\sigma)$ is not smaller than its distance from a; thus, σ is loosely active and a non-empty result is generated.

Loose local queries guarantee that, if the target object intersects the focus set, then a non-empty result is returned. In other words, no intersections are lost.

The same notion of strict and loose interference can be used in restricting resolution filters to areas of interest.

Let R be an arbitrary resolution filter, and let $I \subseteq \mathbb{E}^d$ be an area of interest. Then, a resolution filter $R'_{I,R}$ which restricts R to I can be defined in a strict way as:

$$R'_{I,R}(\gamma) = \begin{cases} R(\gamma) & \text{if and only if } \gamma \cap I \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

or in a loose way as:

$$R'_{I,R}(\gamma) = \begin{cases} R(\gamma) & \text{if and only if the distance between} \gamma \text{ and } I \text{ is } \leq \varepsilon(\gamma) \\ 0 & \text{otherwise} \end{cases}$$

In a global query, using a resultion filter loosely or strictly restricted to an area of interest may lead to different results: in general, in the first case the portion of the target object which is refined according to the original resilution filter R is larger.

For conservative Multi-Complexes in $\mathbb{E}^d$ (see Section 4.4.2), the notion of strict and loose interference coincide:

Property 5.1 *Let $\mathcal{M}$ be a conservative (thus, d-dimensional) Multi-Complex in $\mathbb{E}^d$; then, for every k-cells γ of $\mathcal{M}$ and for every geometric entity $S \subseteq \mathbb{E}^d$, if γ loosely interferes with S then γ strictly interferes with S.*

In a conservative Multi-Complex in $\mathbb{E}^d$ (see Definition 4.21), the warping error, measured in $\mathbb{E}^d$, is always null since all representations at all resolutions cover the same carrier. Thus, for each spatial entity $S \subseteq \mathbb{E}^d$, the result of a strict and that of a loose interference test are the same.

Conservative Multi-Complexes are typically used for representing d-dimensional scalar or vector fields. In this case, a conservative MC $\mathcal{M}$ covers the domain of the field, and the field values define an embedding of $\mathcal{M}$ into the higher-dimensional space $\mathbb{E}^{d+m}$, containing the graph of the field. The d-dimensional MC in $\mathbb{E}^{d+m}$ resulting from the embedding is not conservative (see Figure 5.3).

5.4 Algorithms for Global Queries

In this section, we propose algorithms for the global query in the two variants: the min-size and the best-resolution variant.

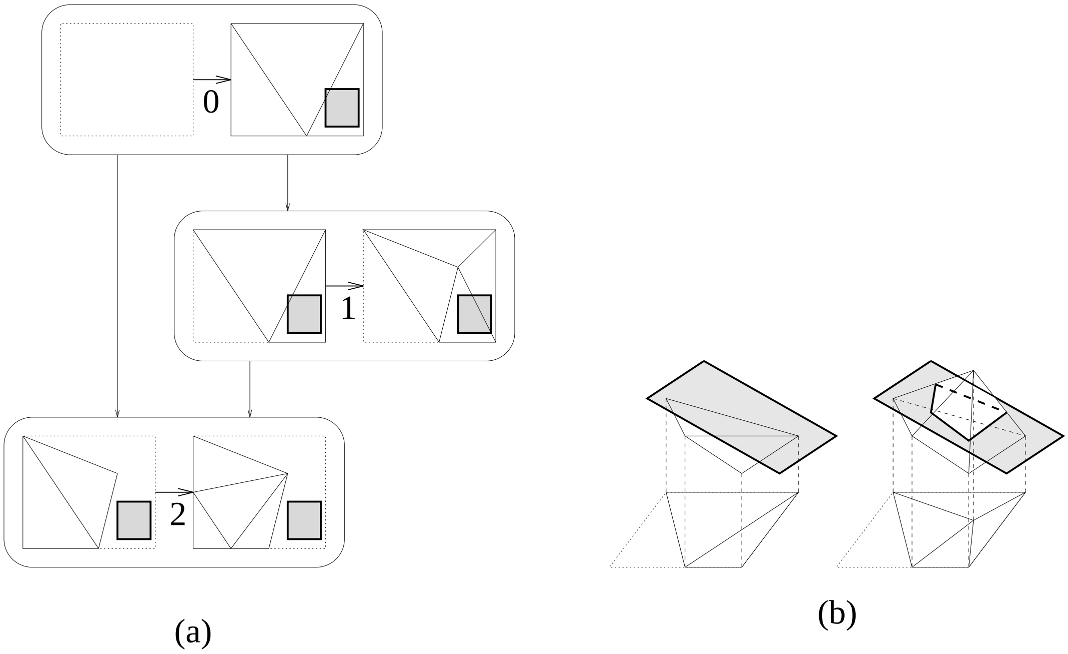


Figure 5.3: (a) A conservative two-dimensional MC $\mathcal{M}$ and a focus set F (the grey box). Note that, for every node C (except the root), F intersects $\Pi_1(C)$ if and only if F intersects $\Pi_2(C)$. (b) If $\mathcal{M}$ represents a field, then its three-dimensional graph is not conservative: a focus set in $\mathbb{E}^3$ is shown which intersects $\Pi_1(1)$ and not $\Pi_2(1)$.

5.4.1 An Algorithm for the Min-Size Global Query

From Lemmas 4.3 and 4.4, we know that the global combined cell complexes of an MC $\mathcal{M}$ are the same as the front complexes of the sub-MCs of $\mathcal{M}$, or, equivalently, they are the same as the complexes associated with the MC-fronts of $\mathcal{M}$. Thus, the cell complex that is the result of the query admits a characterization based on the sub-MC which generates it.

Since every node of an MC represents a refinement pattern, adding a node to a sub-MC increases the size of its front cell complex. Thus, requiring a minimum number of k-cells in a global combined cell complex is equivalent to requiring a minimum number of nodes in the corresponding sub-MC:

Lemma 5.1 *Given a Multi-Complex $\mathcal{M}$ and a resolution filter R, the answer to the* min-size global query *is a cell complex Γ such that:*

1. *Γ is a global combined cell complex of $\mathcal{M}$, i.e., the front cell complex of a sub-MC $\mathcal{M}'$ of $\mathcal{M}$;*

2. *all k-cells of Γ are feasible with respect to the resolution filter R;*

3. *there exists no other sub-MC $\mathcal{M}''$ of $\mathcal{M}$, such that the front cell complex of $\mathcal{M}''$ satisfies condition 2, and the node set of $\mathcal{M}''$ is completely contained in that of $\mathcal{M}'$.*

The proof of the lemma is straight-forward, based on the fact that all modification patterns in an MC are refinement patterns.

The sub-MC $\mathcal{M}'$ providing the solution of the min-size global query can be computed by traversing the DAG in a top-down fashion. The algorithm starts from the sub-MC containing only the root, and iteratively expands it by including nodes as long as the corresponding front cell complex contains some unfeasible k-cell. In other words, an MC-front is progressively advanced from the root towards the drain of the enriched MC associated with $\mathcal{M}$, following a greedy strategy that tries to refine the current complex by applying, at each step, a modification pattern which replaces some unfeasible k-cell.

The basic operation consists of *including a node* C into the current sub-MC $\mathcal{M}'$, or, equivalently, of sweeping the current MC-front after C. Two cases may arise:

1. If all ancestors of C are already in $\mathcal{M}'$ (i.e., all arcs entering C are in the front MC-front of $\mathcal{M}'$), then it is sufficient to add C and its incoming arcs to the set of nodes and to the set of arcs, respectively, of $\mathcal{M}'$ (on the enriched MC, this is equivalent to replacing the arcs entering C with those leaving C in the current MC-front).

2. Otherwise, it is necessary first to include any ancestor of C that is not already in $\mathcal{M}'$, and finally include C into $\mathcal{M}'$ by proceeding as in case 1.

The above idea gives rise to the following recursive algorithm:

Algorithm 5.1
Input: a sub-MC $\mathcal{M}'$ of a Multi-Complex $\mathcal{M}$, and a node C of $\mathcal{M}$ (the algorithm will be always called with C being the destination of a front arc of $\mathcal{M}'$).
Output: $\mathcal{M}'$ after modifications.

1. For each parent C' of C which is not in $\mathcal{M}'$, recursively execute Algorithm 5.1 on C';

2. (now, all the parents of C are in $\mathcal{M}'$) add C and its incoming arcs to $\mathcal{M}'$.

It is easy to verify that Algorithm 5.1 guarantees that, if the input is a sub-MC, then the output is a sub-MC as well:

Lemma 5.2 *Let $\mathcal{M}'$ be a sub-MC of a Multi-Complex $\mathcal{M}$, and C be a node of $\mathcal{M}$, destination of a front arc of $\mathcal{M}'$. Then, after having executed Algorithm 5.1 on $\mathcal{M}'$ and C, $\mathcal{M}'$ is still a sub-MC.*

The proof of Lemma 5.2 is omitted.

The algorithm for extracting a cell complex based on query a resolution filter R from a Multi-Complex $\mathcal{M}$ is outlined below:

Algorithm 5.2
Input: a Multi-Complex $\mathcal{M}$, and a resolution filter R.
Output: a cell complex Γ.

1. [Initialization] Initialize the current sub-MC $\mathcal{M}'$ as the one containing just the root node (equivalently, initialize the current MC-front as the one formed by the arcs outgoing from the root).
2. [Expansion] While the front cell complex of $\mathcal{M}'$ (equivalently, the union of the arc labels of the current MC-front) contains some unfeasible k-cell:
   a. choose a node C such that $\Pi_1(C)$ contains a k-cell γ of the current front complex which is not feasible;
   b. include C into the current sub-MC $\mathcal{M}'$ along with all its ancestors that are not already in $\mathcal{M}'$ (equivalently, sweep the current MC-front after C) by applying Algorithm 5.1.
3. [Output] Return the front cell complex Γ of $\mathcal{M}'$ (equivalently, the union of the arc labels of the current MC-front).

Figure 5.4 shows Algorithm 5.2 at work. The resolution filter is induced by an area of interest consisting of the grey square and requires the maximum resolution inside the square and any resolution outside it. At the first iteration of the expansion loop, the destination node (node 3) of cell γ_1 is included. At the second iteration, the destination (node 5) of cell γ_2 is included along with its parent (node 4). The resulting triangulation has no more unfeasible cells, and thus it is the result of the query.

5.4.1.1 Correctness of the Algorithm

The mechanism used to choose the node C to be included in Step 2.a of Algorithm 5.2 is left to the implementation. It can be shown that the result is unique in the case of many unfeasible k-cells π, regardless of the order in which their destinations are chosen (as a corollary to Theorem 5.1).

We assume that the resolution filter R is identical to zero on the cells of the front complex of any MC (i.e., cells than cannot be refined further must be accepted as they are). The correctness of Algorithm 5.2 is established by the following theorem:

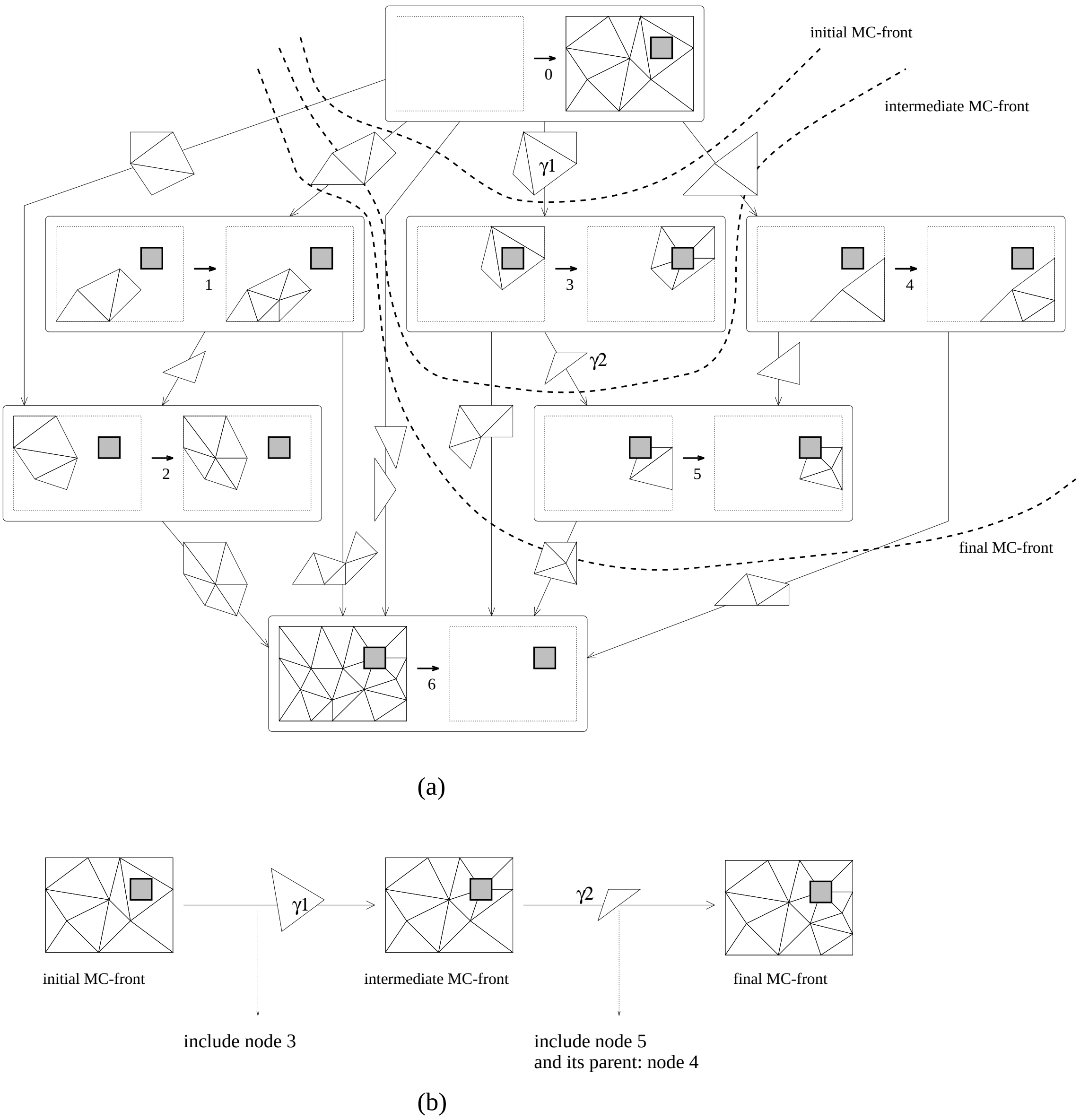


Figure 5.4: A working example of Algorithm 5.2. (a) The enriched MC. The resolution filter requires the maximum resolution inside the grey square, and accepts any resolution outside it. The three successive MC-fronts considered in the expansion stage are shown. (b) The evolution of the current triangulation during expansion. For each step, we show the cell whose destination is included.

Theorem 5.1 *The result* Γ *of Algorithm 5.2 is the global combined cell complex having minimum size among those where all* k*-cells are feasible.*

Proof We have to prove three facts:

1. Γ is a global combined cell complex;
2. every k-cell in Γ is feasible;
3. every other cell complex satisfying conditions 1 and 2 above has more k-cells than Γ.

Fact 1 holds because Γ is the front complex of a sub-MC (from Lemma 5.2).

Fact 2 holds because the algorithm stops only when all k-cells in the current front complex are feasible.

We prove Fact 3 by showing that the sub-MC $\mathcal{M}'$ generating Γ is the one containing the minimum set of nodes necessary to achieve property 2 on its front cell complex.

Clearly, such a minimum sub-MC must contain the root. Starting from the root, each step of the algorithm selects a node C and includes it into the current sub-MC together with its ancestors. We show that every node selected to be included is necessary in order to achieve condition 2. We prove this by induction on the number m of steps performed before the current one.

- Case $m = 0$.

  The current sub-MC $\mathcal{M}'$ contains just the root, thus it is contained in the minimum sub-MC. The selected node C replaces at least one k-cell γ of the front complex which is not feasible. Since every k-cell in an MC has a unique destination, in no other way γ can disappear from the current front complex unless C is included, thus C must belong to the minimum sub-MC. The only ancestor of C is the root, which is already in $\mathcal{M}'$, thus no other node different from C is included.

- Case $m > 1$.

  We know that the current sub-MC $\mathcal{M}'$ is contained in the minimum sub-MC. Let C be the node selected for inclusion. Reasoning as in the previous case, we conclude that C must belong to the minimum sub-MC, since it replaces at least one k-cell of the front complex which is not feasible. All ancestors of C must also belong to the minimum sub-MC, because otherwise the definition of sub-MC would be violated.

This concludes the proof. $\square$

Algorithm 5.2 maintains a consistent sub-MC at each step. Thus, it may be *interrupted* at any time, and the cell complex provided by the current sub-MC is a consistent representation of the target object. In particular, intermediate results are always less refined than necessary to satisfy the resolution filter on all cells, and have a smaller size.

5.4.1.2 Algorithm Complexity

We can estimate the computational complexity of Algorithm 5.2 parametrically on the costs of auxiliary operations, which depend on their implementation and on the data structure used to encode the MC. The complexity of various implementations of the algorithm on different data structures will be discussed in Chapter 6.

We denote with $\mathcal{T}_i^{(check)}$ the time needed to check the control condition at the i-th iteration of the loop in step 2, and with $\mathcal{T}_i^{(find)}$ the cost of finding the node in step 2.a. We denote with $\mathcal{T}^{(incl)}(C)$ the cost of including a node C in the current sub-MC (except for the recursive inclusion of its parents).

The loop in step 2 is executed at most as many times as the number the nodes in the final sub-MC, since at each iteration the current sub-MC gets at least a new node, and no node is ever removed from it. Thus, the total cost of step 2 is in

$$O(\sum_{i=1}^{m}(\mathcal{T}_i^{(check)} + \mathcal{T}_i^{(find)}) + \sum_{C \in \mathcal{M}'}(\mathcal{T}^{(incl)}(C)))$$

where m is the number of iterations performed; note that m is smaller than, or equal to, the number of nodes in the final sub-MC $\mathcal{M}'$.

5.4.2 An Algorithm for the Best-Resolution Global Query

Since Algorithm 5.2 maintains a cell complex whose size increases at each iteration, the basic idea is interrupting Algorithm 5.2 before the size of current complex becomes too large. Note that, if Algorithm 5.2 is interrupted, the current MC-front may not be refined uniformly with respect to the given resolution filter. In other words, the current front cell complex may satisfy the filter already in some areas, while it can be far from it elsewhere.

In order to obtain a more uniform refinement, it necessary to modify step 2.a (selection of the next node to be included). A good strategy consists of always choosing the destination C of the k-cell γ corresponding to the largest value of the resolution filter R. Unfortunately, it is not guaranteed that including the destination of γ in the current sub-MC does not introduce in the new front complex cells having a larger value of R than γ. We are only sure that, after including C and a suitable subset of its children, the maximum value of R over the corresponding front complex will be less than $R(\gamma)$. Thus, the idea is that we try to include C; if the largest value of R of the current complex increases, then we recursively try to include the destination of all the new cells γ' introduced in the current complex where $R(\gamma') > R(\gamma)$; if the complex size exceeds the given bound during this operation, then we backtrack to restore the situation before the inclusion of C, stop the algorithm, and output the current front complex.

In the new approach, Algorithm 5.2 is modified as follows:

Algorithm 5.3
Input: a Multi-Complex $\mathcal{M}$, a resolution filter R, and a bound b for the size of the extracted cell complex.
Output: a cell complex Γ.

1. [Initialization]: as in Algorithm 5.2.
2. [Expansion] While front($\mathcal{M}'$) contains some k-cell that is not feasible, and the size of front($\mathcal{M}'$) is less than b:
   a. find the k-cell γ that maximizes the value of $R(\gamma)$; let C be the node such that $\gamma \in \Pi_1(C)$;
   b. include C into the current sub-MC $\mathcal{M}'$ along with all its ancestors that are not already in $\mathcal{M}'$ (as in Algorithm 5.2);
   c. while the maximum value of the resolution filter on front($\mathcal{M}'$) is larger than $R(\gamma)$: for every k-cell $\gamma' \in$ front($\mathcal{M}'$) such that $R(\gamma') > R(\gamma)$, include in $\mathcal{M}'$ the destination of γ' and all its ancestors;
   d. if, during step 2.b or 2.c, the size of front($\mathcal{M}'$) gets larger than b, then restore the situation before step 2.a, and go to step 3.
3. [Output]: generate Γ as in Algorithm 5.2.

5.4.2.1 Correctness of the Algorithm

The correctness of Algorithm 5.3 is established in the following theorem:

Theorem 5.2 *The result Γ of Algorithm 5.3 is the global combined cell complex which minimizes the value of* $\max\{R(\gamma)|\gamma \in \Gamma^{(k)}\}$ *among the ones having size less or equal to b.*

Proof We have to prove three facts:

1. Γ is a global combined cell complex;
2. the number of k-cells in Γ is $\leq b$;
3. for every other cell complex Γ' satisfying properties 1 and 2 above, there is at least one k-cell $\gamma' \in \Gamma'$ such that $R(\gamma') > \max\{R(\gamma)\,|\,\gamma \in \Gamma^{(k)}\}$.

We assume that the values of $R(\gamma)$ for the k-cells of $\mathcal{M}$ are all distinct.

Fact 1 holds because Γ is the front complex of a sub-MC (as for Algorithm 5.2).

Fact 2 holds because the main loop of step 2 exits if the current complex has a size $\geq b$, and the condition in step 2.d guarantees that a complex of size $> b$ cannot be generated.

We prove Fact 3 by showing that, at each iteration of loop 2, the action of including C decreases the maximum value of R over the cells of the current cell complex, and it is the only possible choice in order to do that.

The maximum value of R decreases because we continue including descendants of C until the value of R gets smaller than $R(\gamma)$. Moreover, including any other node different from C leaves cell γ in the current front complex, and thus the maximum value of R is unchanged.

Thus, the size of the current complex increases (i.e., nodes are included) only if the maximum value of R decreases [1], and therefore the current cell complex is the smallest one having that maximum value of R. The current complex is returned as the answer if and only if it is not possible to reduce the maximum value of R without exceeding the size bound. $\square$

5.4.2.2 Algorithm Complexity

The parametric evaluation of the time complexity is unchanged with respect to Algorithm 5.2, but in this variant the cost $\mathcal{T}^{(find)}$ of step 2.a is likely to be higher, because here we must select a specific MC-node, rather than an arbitrary one, among the ones that "remove" some unfeasible cell from the current front complex: this will involve a searching cost. Moreover, we must take into account the cost of possible work that is done in steps 2.a, 2.b and 2.c and then undone in step 2.d because the size of the resulting complex exceeds b (however, this may happen at most once during each execution of the algorithm). See Section 6.3 for an implementation of both algorithms and a comparison of their time complexity.

5.4.2.3 Variants

As an alternative to imposing an upper bound on the size of the extracted cell complex, we can impose an upper bound on the execution time, and interrupt the algorithm as soon as the available time has been exhausted. This variant guarantees that the output complex minimizes R among the ones obtainable within the given time (the proof of Theorem 5.2 can be easily adapted).

5.5 Algorithms for Local Queries

In this Section, we propose three different approaches for the solution of local queries on an MC. For simplicity, we address just min-size queries. All algorithms do not find an optimal

[1] Note that the assumption that the values of R are all distinct is fundamental here.

solution since they fail to satisfy the requirements of minimality of Definition 5.5. In particular, we propose a heuristic for conservative Multi-Complexes which does not guarantee a combined cell complex but simply a complex made of cells from the MC, and is especially efficient (as the experimental results of Section 5.7 show).

In describing the algorithms, we refer to the notion of *active* cells; a variant using *loosely active* cells can be designed in a similar way.

5.5.1 Two-Step Approaches

In this subsection, we present two approaches that solve local queries by exploiting algorithms developed for global ones. The idea is working in two steps:

1. Perform a global query on the given cell complex $\mathcal{M}$ in such a way to return a global combined cell complex Γ (which may contain both active and non-active cells) satisfying R at least on its active cells; the resolution filter to be used here will be discussed later.
2. Use the focus set F to perform a simple interference query on Γ (as in single-resolution cell complexes); this selects the subcomplex Γ' made up of the active cells of Γ.

It is easy to see that the final result Γ' satisfies conditions 1,2,3,4 of Definition 5.5: it is a combined cell complex, it is made of active cells, it can be completed to a global one by adding just non-active cells, and satisfies R on every cell.

Condition 5 (the minimum-size condition) is not necessarily satisfied because, while the global query returns a minimum-size global combined complex, it does not provide the global combined complex containing the active subcomplex of minimum size. Adding an MC-node C to the current sub-MC increases the total number of cells in the current complex, but the number of active cells may decrease. This happens when $\Pi_1(C)$ contains more active cells than $\Pi_2(C)$: the restriction of C to active cells may not be a refinement pattern.

We present two approaches which differ in the resolution filter used in Step 1. The first approach uses the same resolution filter R given for the local query; the resulting complex Γ satisfies R on all of its cells. We note that, in this way, a lot of work is spent in ensuring that the resolution filter is satisfied everywhere, including on non-active cells.

The basic idea of the second approach is that the resolution filter only needs to be satisfied on active cells, while the resolution of non-active cells is irrelevant and may be as coarse as possible. Thus, in Step 1 we use a resolution filter restricted to the focus set F viewed as an area of interest. The new resolution filter R' is defined as equal to R on all active cells, and zero on non-active ones:

$$R'(\gamma) = \begin{cases} R(\gamma) & \text{if and only if } \gamma \text{ is active with respect to } F \\ 0 & \text{otherwise} \end{cases}$$

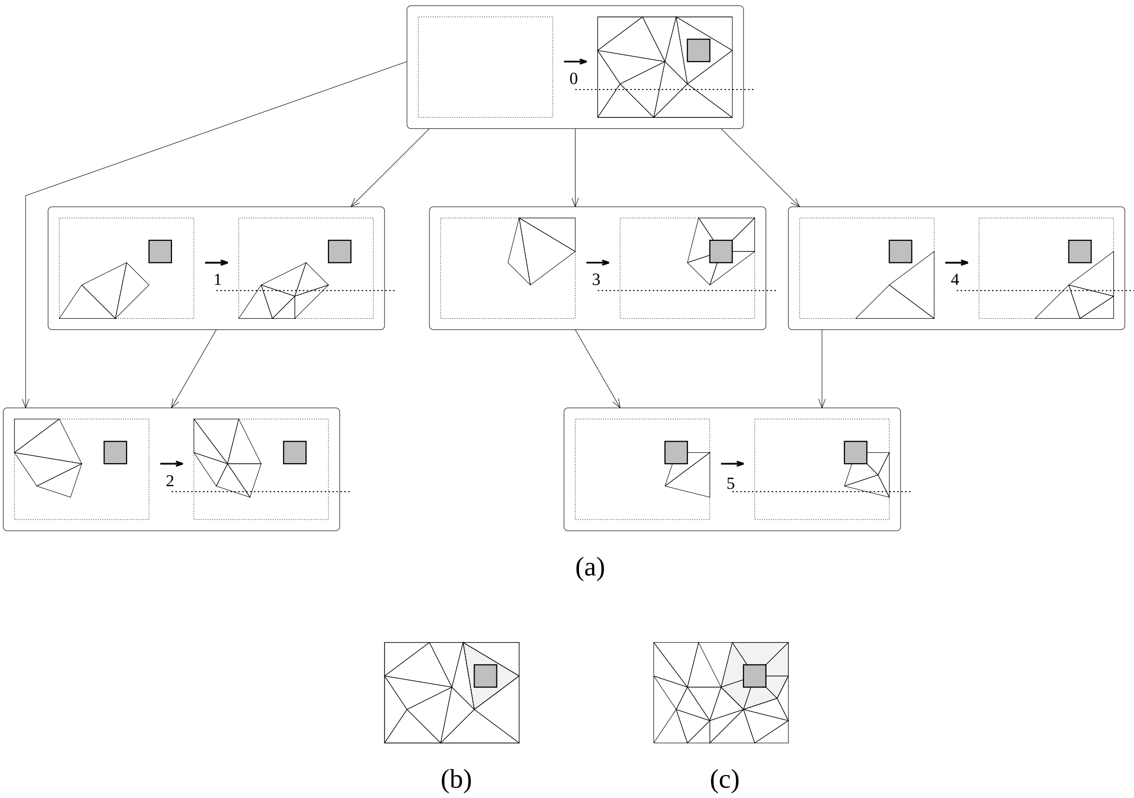


Figure 5.5: Results of answering a local query through the integrated and the pure two-step approach. (a) The Multi-Complex. The focus set is the grey square, the resolution filter requires the maximum resolution for the cells either intersecting or lying below the dotted line, and accepts any resolution for those above the line. (b) The result of the integrated two-step algorithm is the root mesh since the triangle containing the square is feasible. (c) The result of the pure two-step approach has to detail the part below the line to the maximum resolution, thus it returns the front cell complex. Note that the active parts of (b) and (c) (shown shaded) are different.

We call this approach an *integrated two-step approach* since it integrates the focus set in the first step as well; the previous approach will be called the *pure two-step approach*.

For the same resolution filter R and an arbitrary focus set F, in general the two global combined cell complexes Γ_{pure} and Γ_{intg} resulting from Step 1 of the pure and of the integrated two-step approach, respectively, are different, and Γ_{pure} contains more cells than Γ_{intg} (see Figure 5.5). The two corresponding results of Step 2 (the selection of active cells) Γ'_{pure} and Γ'_{intg} may be different as well. It is not necessarily true that Γ_{pure} has a larger size than Γ'_{intg}, but Γ'_{intg} has been obtained with less work on the DAG.

5.5.2 A Local Query Algorithm

Two-step approaches outlined in Section 5.5.1 always build a cell complex covering the whole target object, although the final result of the query covers just a portion of it. For instance, in the case of a point location query, the final result consists of just one cell, while the first step extracts a complex covering the whole object. It is true that, with an integrated two-step approach, the resolution of the intermediate complex Γ outside the focus set is low, but Γ may still contain many cells if the object represented in the Multi-Complex is very large.

Since a local query is only interested in the "active part" of an MC-front, the idea is sweeping a "local" MC-front. Ideally, we would like to use a local MC-front which is the subset of an MC-front $\mathcal{A}$ formed by the arcs of $\mathcal{A}$ that contain some active cells in their labels. In practice, this condition cannot be achieved since it is not possible to test whether two arcs are on the same directed path if not actually searching for a path, which is too expensive.

Our "local" MC-front is not a subset of an MC-front: it is allowed to contain more than one arc lying on the same path, provided that the sets of active cells labelling them can coexist in a cell complex. Thus, we call it a *pseudo-MC-front*. The corresponding set of nodes traversed by the algorithm does not form a sub-MC, because some dependency links have been skipped while constructing it, and, thus, not all ancestors of a node are in it. We call such structure a *pseudo-sub-MC*. The pseudo-MC-front associated with a pseudo-sub-MC $\mathcal{M}'$ of a Multi-Complex $\mathcal{M}$ is the set of arcs A of the corresponding enriched MC, where the source of A is in $\mathcal{M}'$ and its destination is not in $\mathcal{M}'$. Figure 5.6 (a) shows a pseudo-MC-front, a pseudo-sub-MC, and the corresponding cell complex.

The union of the labels of arcs in a pseudo-MC-front $\mathcal{A}$ may not be a cell complex (for instance, because some cells intersect in their interiors), but the union of the active cells in the labels of arcs in $\mathcal{A}$ is guaranteed to be a cell complex (see Figures 5.6 (b) and (c)).

For describing the algorithm, we assume to have a *safety test* which permits to decide whether some branches of the DAG need to be examined or not.

Intuitively, a safety test is a Boolean function that takes an MC-node C, a parent C' of C and a focus set: if the test returns true, then we can safely forget the dependency link defined by arc (C', C) and we are guaranteed that the collection of active cells in the current pseudo-MC-front remains a cell complex. The safety test may be more strict than necessary, i.e., it may return false in some cases which are "safe". The important point is that it always returns false in "unsafe" cases.

For convenience, given a focus set F we denote with active(Γ, F) the cell complex formed by the active top-cells of a complex Γ and their lower-dimensional faces.

Definition 5.8 *Given an MC-node C, a parent C' of C, and a focus set F, a* safety test *is a Boolean predicate that returns true only if, for every MC-node C'' that is an ancestor of C' and*

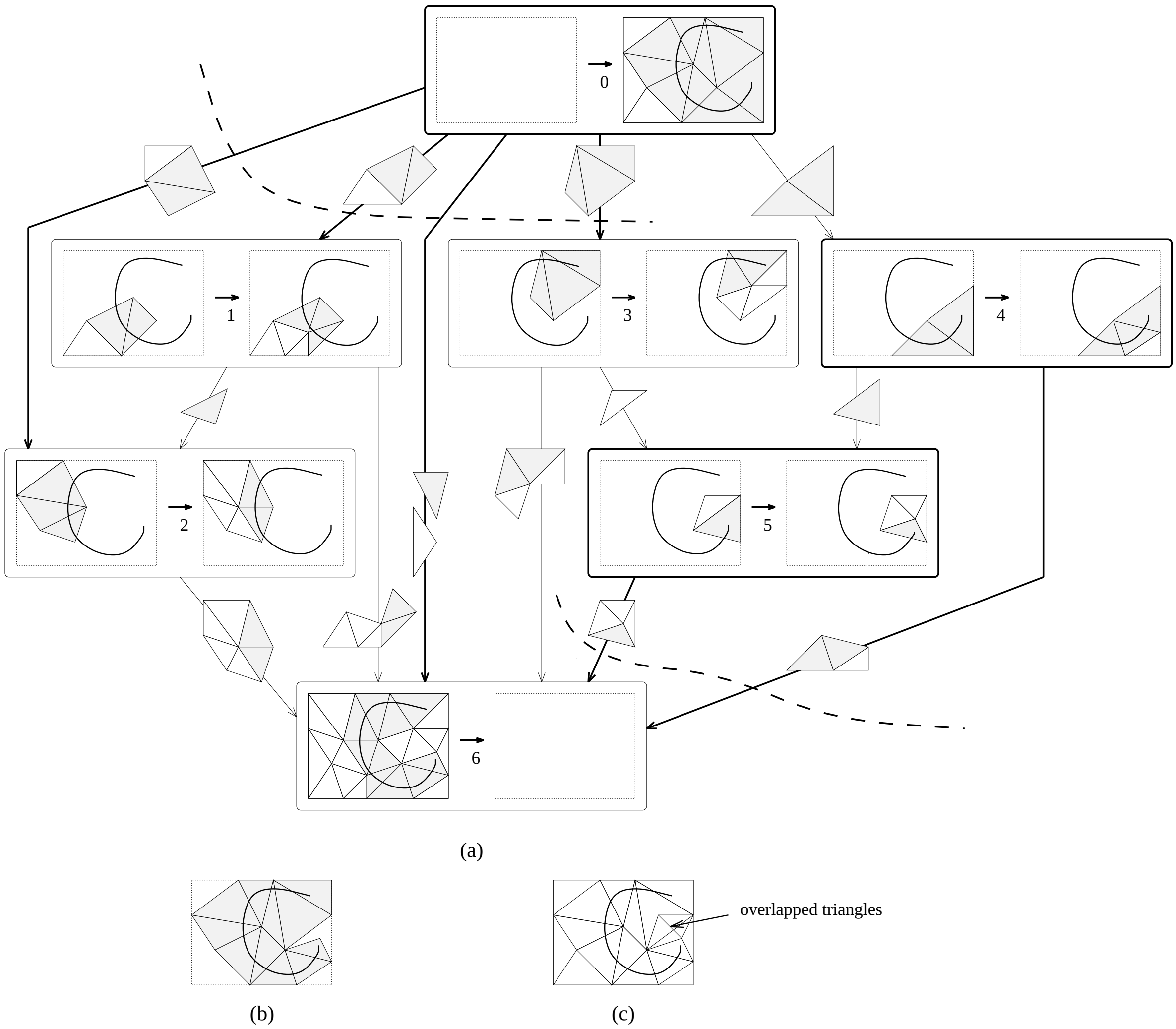


Figure 5.6: (a) A conservative enriched two-dimensional MC, and a focus set (the curve line), active triangles are shaded; a pseudo-MC-front (containing the arcs cut by the thick dashed line) and the corresponding pseudo-sub-MC (containing nodes 0, 4, and 5) are shown. (b) The triangulation formed by collecting the active triangles from the arcs of the pseudo-MC-front. (c) The set (not a complex) of all the triangles labelling the arcs of the pseudo-MC-front.

not an ancestor of any other parent of C*, the union of* $\text{active}(\Pi_2(C''), F)$ *and* $\text{active}(\Pi_2(C), F)$ *is a cell complex, i.e.,*

$$\Delta(\text{active}(\Pi_2(C'')) \cap \Delta(\text{active}(\Pi_2(C)) = \Delta(\partial(\text{active}(\Pi_2(C''))) \cap \partial(\text{active}(\Pi_2(C)))).$$

Definition 5.8 corresponds to the idea that it is possible to identify, on a local basis, cases in which $\text{active}(\Pi_2(C''), F)$ and $\text{active}(\Pi_2(C), F)$ can coexist in a larger cell complex.

Intuitively, if a safety test returns true on C, C' and F, then a pseudo-MC-front containing arcs outgoing from C'' and C (which are on the same path) is guaranteed to define a cell complex when restricted to its active cells.

Assuming to have a safety test, we define an algorithm for the local query and prove that its result satisfies Conditions 2,3,4 of Definition 5.5, and a weaker version of Condition 1, i.e., it ensures that the output is a cell complex, although not a combined cell complex.

We will provide a safety test for *conservative* d-dimensional cell complexes in $\mathbb{E}^d$ (see Section 4.4.2). Finally, we will extend the same approach as a *heuristic* for generic MCs, where we do not have a safety test.

The local algorithm uses a similar approach as the global one, i.e., a top-down traversal of the DAG. It modifies subcomponent Algorithm 5.1 in the following way:

Algorithm 5.4
Input: a pseudo-sub-MC $\mathcal{M}'$ of a Multi-Complex $\mathcal{M}$, and a node C of $\mathcal{M}$ (the algorithm will be always called with C being the destination of an arc of the pseudo-MC-front of $\mathcal{M}'$).
Output: $\mathcal{M}'$ after modifications.

1. for each parent C' of C such that $C' \notin \mathcal{M}'$ and the safety test returns false on C, C' and F, recursively execute Algorithm 5.4 on C';
2. update $\mathcal{M}'$ by adding C and its incoming arcs whose source is in $\mathcal{M}'$.

The difference with respect to the global case is that, when including a node C into the current pseudo-sub-MC, the local algorithms does not consider arcs (C', C) on which the safety test returns true.

Algorithm 5.4 preserves the property that the union of the collection of active cells in the labels of the arcs of the pseudo-MC-front of $\mathcal{M}'$ is a cell complex, as established by the following theorem:

Theorem 5.3 *Let $\mathcal{M}'$ be a pseudo-sub-MC of a conservative Multi-Complex $\mathcal{M}$, and C be a node of $\mathcal{M}$, destination of an arc of the pseudo-MC-front $\mathcal{A}$ of $\mathcal{M}'$.*

We denote with $\Gamma_{\mathcal{A}}$ the union of the labels of arcs in $\mathcal{A}$. If set active$(\Gamma_{\mathcal{A}}, F)$ *is a cell complex, then, after calling Algorithm 5.4 on $\mathcal{M}'$ and C,* active$(\Gamma_{\mathcal{A}}, F)$ *(where $\mathcal{A}$ now denotes the new pseudo-MC-front) is still a cell complex.*

Proof The proof reduces to showing that, for every arc A' outgoing from C that is added to $\mathcal{A}$, and for every arc A possibly present in $\mathcal{A}$ that lies on the same directed path as A', the active cells of A and A' can coexist in a cell complex.

The source of arc A must be a node C_1 such that C_1 is an ancestor of C and, for every descendant C_2 of C_1 which is a parent of C, the safety test returns true on C, C_2 and F.

Then, all active cells of C_2 can coexist with the active cells of C, and in particular those labelling the two arcs A and A'. $\square$

The local query algorithm on a Multi-Complex $\mathcal{M}$ behaves in a similar way as Algorithm 5.2, with the difference that all calls to Algorithm 5.1 are replaced with calls to Algorithm 5.4, and all arcs which do contain any active k-cells are purged from the current front:

Algorithm 5.5
Input: a Multi-Complex $\mathcal{M}$, a resolution filter R, and a focus set F.
Output: a cell complex Γ.

1. [Initialization] Initialize the current pseudo-sub-MC $\mathcal{M}'$ as the one containing just the root node, and its front MC-front as the set of arcs outgoing from the root, and containing at least one active k-cell.
2. [Expansion] While the front cell set of $\mathcal{M}'$ (equivalently, the union of the arc labels of the current pseudo-MC-front) contains some active k-cell that is not feasible:
   a. choose a node C such that $\Pi_1(C)$ contains one of the above k-cells;
   b. include C into $\mathcal{M}'$ by applying Algorithm 5.4.
3. [Output] Return the cell complex Γ formed by active k-cells belonging to the labels of the arcs of the pseudo-MC-front of $\mathcal{M}'$.

5.5.2.1 Correctness of the Algorithm

We prove that Algorithm 5.5 is correct with respect to the following requirements:

Theorem 5.4 *Given a Multi-Complex, a focus set F, and a resolution filter R, Algorithm 5.5 outputs a set of cells Γ_{loc} such that:*

1. *Γ_{loc} is a cell complex made of cells from $\mathcal{M}$;*
2. *all cells of Γ_{loc} are active;*
3. *let $\mathcal{A}$ be the pseudo-MC-front providing Γ_{loc}; each directed path in the MC which does not contain an arc of $\mathcal{A}$ contains at least one arc labelled with non-active cells only; moreover, for every arc $A \in \mathcal{A}$, all active cells of A are in Γ_{loc};*
4. *all active cells of Γ are feasible;*

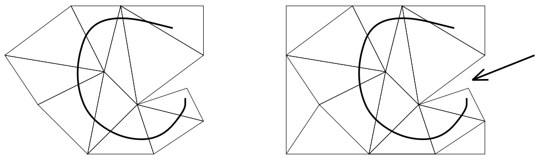


Figure 5.7: The result of the local query shown in Figure 5.6 cannot be completed to cover the whole domain: The arrows indicates a gap that cannot be filled with MC triangles.

Condition 3 replaces the requirement of Definition 5.5 that Γ_{loc} must be completable to a global combined cell complex by adding just non-active cells; it is equivalent to such requirement when Γ_{loc} is a combined complex.

Note that Γ_{loc} cannot in general be completed to a cell complex covering the whole target object: the case shown in Figure 5.6 is a counter-example (see also Figure 5.7). In fact, complexes generated by the local algorithm are not combined cell complexes.

Proof of Theorem 5.4 It is straight-forward to see that Γ_{loc} is made of cells of $\mathcal{M}$ and that all its cells are active and feasible (similar to the proof of Theorem 5.1).

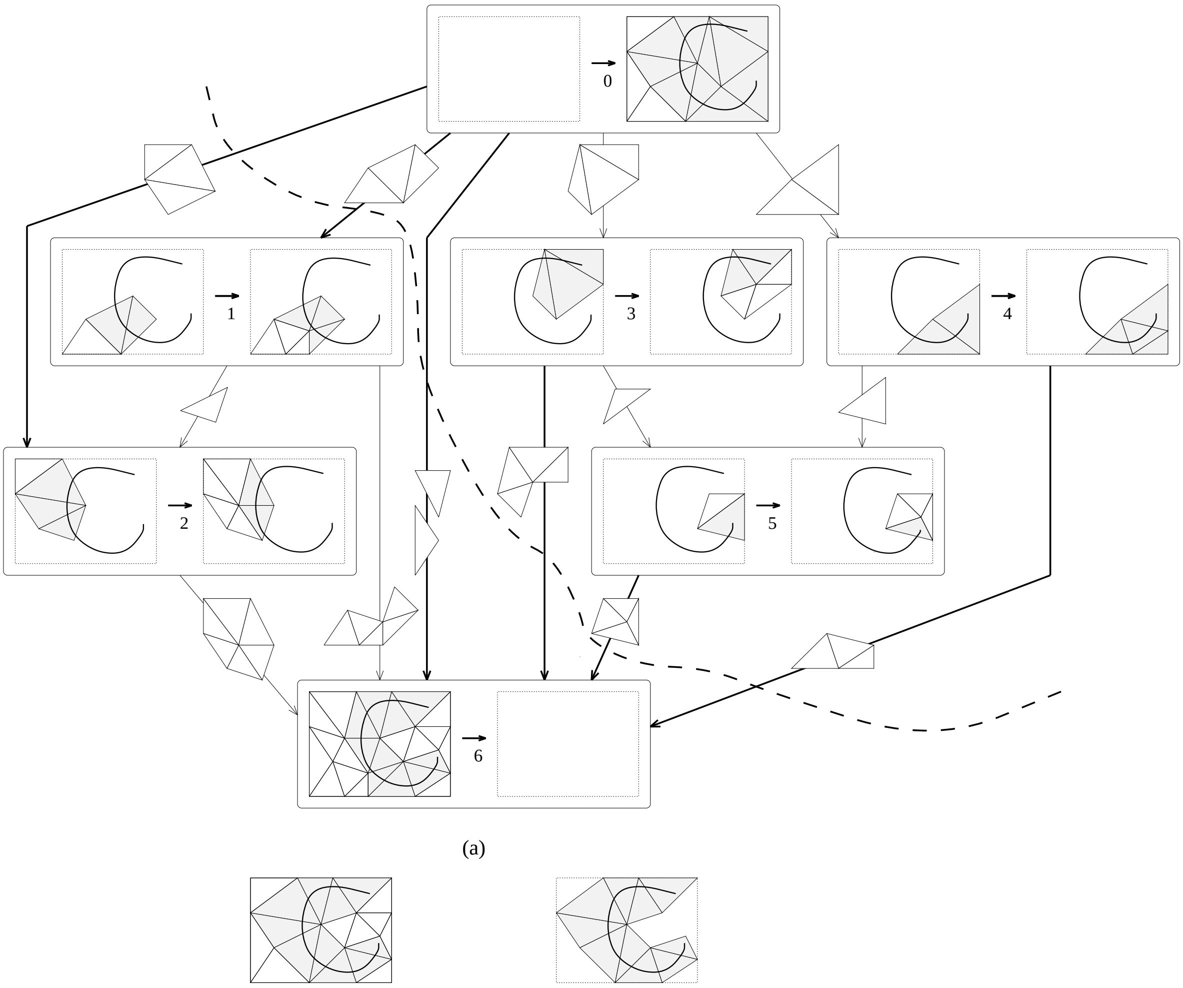
We show that Γ_{loc} is a cell complex by observing that, just after initialization, the current pseudo-MC-front provides a cell complex (since it is a subset of an MC-front and, thus, a combined complex) and, because of Theorem 5.3, each application of Algorithm 5.4 preserves this property.

All paths that are discarded by Algorithm 5.5 start with an arc that does not contain active cells in its label. Moreover, all active cells labelling each arc of the pseudo-MC-front are inserted in Γ_{loc}. Thus, Fact 2 follows. $\square$

As in the two-step algorithms, the result may not be of minimum size.

If we compare the pseudo-sub-MC $\mathcal{M}'_{loc}$ generated by Algorithm 5.5 and the sub-MC $\mathcal{M}'_{intg}$ generated by Step 1 of the integrated two-step method, we note that all the nodes included in $\mathcal{M}'_{loc}$ are necessarily included in $\mathcal{M}'_{intg}$ as well. This means that the cell complex extracted by the local algorithm is less refined than (or as refined as) the active part of the cell complex extracted by the integrated two-step algorithm; hopefully, it will also have a smaller size. Figures 5.6 and 5.8 show an example in which the complexes produced by the two algorithms are different. Moreover, having traversed fewer nodes, intuitively Algorithm 5.5 has been less expensive.

A trivial safety test is one that returns true on every input; in this case, Algorithm 5.5 produces the same output as the integrated two-step method since every parent of every included node is included as well.



(a)

(b)

(c)

Figure 5.8: (a) The same MT and focus set used for the local query in Figure 5.6. As in Figure 5.6, the focus set is the curve line, active triangles are shaded. the MC-front resulting from a global query with a resolution filter restricted to the given focus is shown (the thick dashed line cuts the arcs of the MC-front). (b) The corresponding global combined cell complex, corresponding to the result of Step 1 of the integrated two-step approach to local queries. (c) The active triangles of (b), corresponding to the result of Step 2 of the integrated two-step approach.

5.5.2.2 Algorithm Complexity

The time complexity of the algorithm can be evaluated parametrically on the costs of the subcomponents, in a similar way as done for Algorithm 5.2. In $\mathcal{T}_i^{(check)}$ we have the additional cost of evaluating the safety test on every parent of a node being included. However, note that in general the total number of nodes traversed is less than in the global algorithm with a

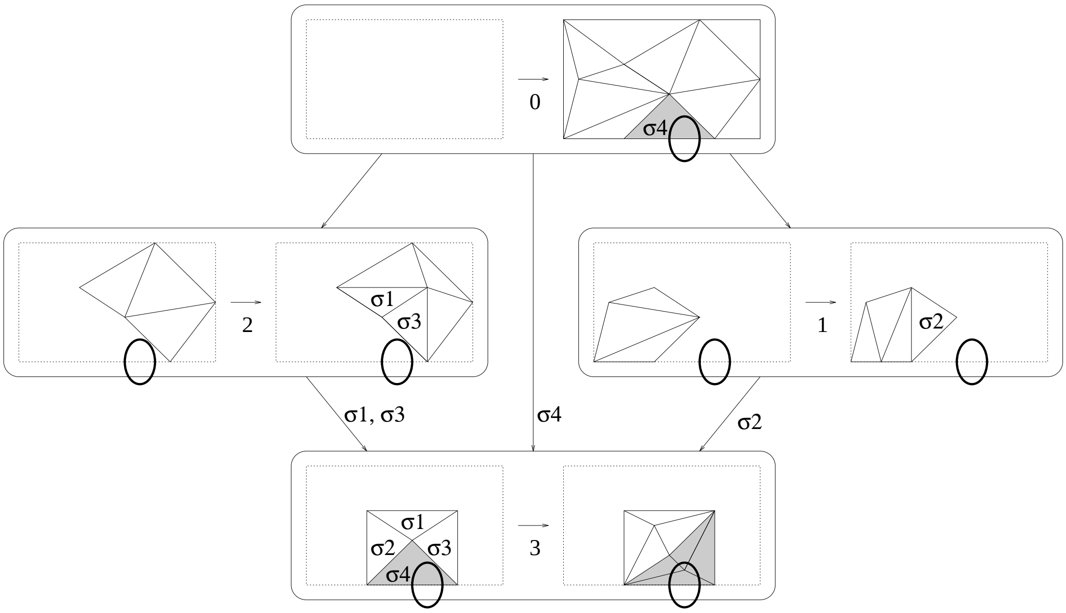


Figure 5.9: Relevant and non-relevant parents for node 3, with respect to a focus set consisting of the thick circle (active triangles are shaded). Node 0 is a relevant parent since triangle σ_4 is active; node 1 is not a relevant parent since triangle σ_2 is not active and does not intersect the active part of $\Pi_2(3)$; node 2 is a relevant parent since triangle σ_3 (not active) intersects the active part of $\Pi_2(3)$.

restricted resolution filter, and thus the time complexity is expected to be better (as confirmed by experiments reported in Section 5.7).

We have defined a local algorithm for the min-size version of the local query; a local algorithm for the best-resolution version can be defined in a totally similar way as done in Section 5.4.2 for the global query.

5.5.3 A Safety Test for Conservative MCs

For conservative MCs, we define the following notion of a *relevant* parent for an MC-node:

Definition 5.9 *Let (C_1, C_2) be an arc of a conservative Multi-Complex $\mathcal{M}$ in $\mathbb{E}^d$, and $F \subseteq \mathbb{E}^d$ be a focus set. We say that C_1 is a* relevant parent *of C_2 with respect to F if, denoted with Γ' the cell complex* $\mathrm{sub}(\Pi_2(C_1) \cap \Pi_1(C_2), \Pi_2(C_1))$*:*

1. *either Γ' contains some active k-cell, or*
2. $\Delta(\Gamma') \cap \Delta(\mathrm{active}(\Pi_2(C_2), F)) \neq \Delta(\partial'(\Gamma') \cap \partial'(\mathrm{active}(\Pi_2(C_2), F))$.

See Figure 5.9 for an illustration of the possible cases.

Γ' is the same as the cell complex labelling arc (C_1, C_2) in the enriched MC. Intuitively, Γ' is the set of cells that cause C_2 to depend on C_1. Definition 5.9 tells that a parent of C_2 is not relevant if the corresponding arc does not contain active cells and does not interfere with the active part of $\Pi_2(C_2)$. This implies that active$(\Pi_2(C_1), F)$ and either $\Pi_1(C_2)$ or $\Pi_2(C_2)$ may coexist in a larger complex.

Note that condition 2 implies that $\Delta(C_1) \cap \Delta(\text{active}(\Pi_2(C_2), F)) \neq$ $\Delta(\partial'(C_1) \cap \partial'(\text{active}(\Pi_2(C_2), F))$, because $\Delta(\Gamma')$ is the only portion of the carrier of C_1 that intersects the carrier of C_2; thus, it corresponds to the intuitive notion given before.

Our *safety test for conservative MCs* returns true on C, C' and F if and only if C' is not a relevant parent for C with respect to F.

Now, we prove that such a definition satisfies the property required for a safety test:

Theorem 5.5 *Let $\mathcal{M}$ be a conservative Multi-Complex, F a focus set, C and C' two nodes of $\mathcal{M}$ where C is a child of C'. If C' is not a relevant parent for C with respect to F, then for every MC-node C'' that is an ancestor of C' and not an ancestor of any other parent of C, the union of* active$(\Pi_2(C''), F)$ *and* active$(\Pi_2(C), F)$ *defines a cell complex, i.e.,* $\Delta(\text{active}(\Pi_2(C'')) \cap \Delta(\text{active}(\Pi_2(C)) = \Delta(\partial'(\text{active}(\Pi_2(C''))) \cap \partial'(\text{active}(\Pi_2(C))))$.

The proof of Theorem 5.5 uses the following lemma:

Lemma 5.3 *In a conservative d-dimensional Multi-Complex, if C_1 is an ancestor of C_2, then* $\Delta(C_1) \cap \Delta(C_2) \subseteq \cup\{\Delta(C_i) \,|\, C_i \textit{ parent of } C_2 \textit{ and descendant of } C_1\}$.

Proof of Lemma Since $\Pi_1(C_2)$ is the union of the labels of its incoming arcs, then its carrier $\Delta(C_2)$ is contained in the union of the carriers of its parents. Let C_i be a parent of C_2. If C_i is not a descendant of C_1, then $\Delta(C_1) \cap \Delta(C_i)$ is less than d-dimensional, and thus does not contribute to cover $\Delta(C_1)$. $\square$

Proof of Theorem 5.5 The proof reduces to showing that, for every ancestor C'' such that all the parents of C that are also descendants of C'' are not relevant, we have that, for $i = 1, 2$, active$(\Pi_i(C''), F)$ can coexist with active$(\Pi_2(C), F)$.

More precisely, we show a slightly stronger fact, i.e., that $\Delta(C'') \cap \Delta(\text{active}(\Pi_2(C), F)) =$ $\Delta(\partial'(C'') \cap \partial'(\text{active}(\Pi_2(C), F)))$. This means that $\Pi_i(C'')$ as a whole (and, in particular, its active part) can coexist with active$(\Pi_2(C), F)$. The "$\supseteq$" inclusion is trivial; we prove the "$\subseteq$" one.

Without loss of generality, we assume that there is only one parent C' of C that is a descendant of C''.

We note that $\Delta(C'') \cap \Delta(\text{active}(\Pi_2(C), F)) \subseteq \Delta(C'') \cap \Delta(\Pi_2(C))$, and for Lemma 5.3, this is $\subseteq \Delta(C')$. But, since C' is not relevant, then $\Delta(C') \cap \Delta(\text{active}(\Pi_2(C), F)) = \Delta(\partial'(C') \cap \partial'(\text{active}(\Pi_2(C), F))$.

Since $\Delta(\partial'(C') \cap \partial'(\text{active}(\Pi_2(C), F))$ is less than d-dimensional, this means that $\Delta(C'') \cap \Delta(\text{active}(\Pi_2(C), F))$ is less than d-dimensional, thus the two domain do not intersect in their interiors, but may touch along the boundaries. Note that this result holds for C'' and for all MC-nodes on the directed path from C'' to C.

It remains to show that cells along the shared boundary portions of $\Delta(C'')$ and $\Delta(\text{active}(\Pi_2(C), F))$ are pairwise matching, i.e., any two $(d-1)$-cells $\gamma \in \partial'(\text{active}(\Pi_2(C), F))$ and $\gamma' \in \partial'(C'')$ do not intersect in their interiors.

By contraddiction, assume that γ and γ' intersect in their interiors. Since γ' is internal to $\Delta(\mathcal{M})$, then there must be a node C_q along the path from C'' to C such that $\gamma' \in \Pi_1(C_q)$ and $\gamma' \notin \Pi_2(C_q)$. γ' must be internal to $\Delta(C_q)$ since in a conservative MC boundary cells of a node are preserved. Thus, $\Delta(C_q)$ extends on both sides of γ'. Since $\gamma \in \partial'(\text{active}(\Pi_2(C), F))$, then $\Delta(\text{active}(\Pi_2(C), F))$ extends on one side of γ; since γ and γ' properly intersect, then $\Delta(C_q) \cap \Delta(\text{active}(\Pi_2(C), F))$ is d-dimensional. But C_q is in the same conditions as C'', and thus $\Delta(C_q) \cap \Delta(\text{active}(\Pi_2(C), F))$ must be less than d-dimensional. Therefore, we have a contradiction. $\square$.

From Theorem 5.4 it follows that, for conservative MCs, Algorithm 5.5 is guaranteed to return a cell complex.

5.5.4 A Heuristic for Arbitrary MCs

The local query algorithm presented in Section 5.5.2 can be extended as a heuristic strategy for arbitrary k-dimensional cell complexes in $\mathbb{E}^d$. Unfortunately, in this case we are not able to provide a safety test.

We have experimented with an algorithm that uses a weaker test that is not sufficient to guarantee Theorem 5.5: the test returns true on C, C' and F if and only if the label of arc (C', C) does not contain any active k-cell.

The resulting algorithm does not satisfy condition 1 of Theorem 5.4 because its output may be a set of cells that is not a complex (i.e., there may be proper intersections among the cells of the extracted set). This is illustrated for a conservative cell complex in Figure 5.10. But in all of our experiments the output of the local algorithm was a cell complex, and the execution time was much shorter than with the integrated two-step algorithm.

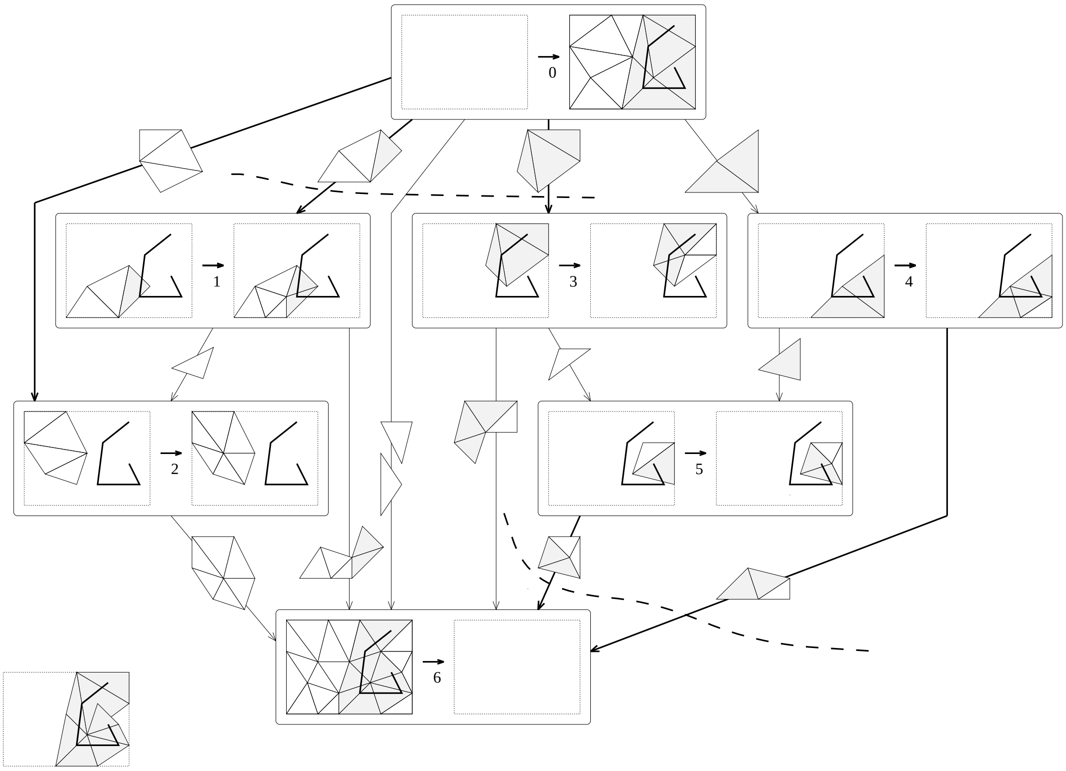


Figure 5.10: The generalized local query algorithm may produce sets of cells which are not complexes. Here, two triangles in the resulting mesh intersect each other. The focus set is a polygonal line; the original MC is shown, with active triangles in grey.

5.6 A Dynamic Approach to Global Queries

In this subsection, we define dynamic algorithms for global queries which find the resulting cell complex by transforming the result of a previous query, instead of recomputing it from scratch as Algorithms 5.2 and 5.3 do. This possibility may be important in a context where the resolution filter changes very gradually between to consecutive queries, and thus the extracted cell complex for the current query is likely to change only slightly with respect to the one extracted for the previous query.

The algorithms presented in this section will be called *dynamic*, in contrast with Algorithms 5.2 and 5.3, that will be called *static* in the reminder. The main feature of static algorithms is that they traverse an MC in a top-down fashion starting from its root. Dynamic algorithms start from an intermediate MC-front and move both forward and backward in the DAG. An additional input parameter is needed, i.e., a global combined complex (result of a previous query) which must be updated. Since global combined complexes are in one-to-one correspondence with sub-MCs, this additional input parameter can be equivalently specified as a sub-MC. Thus, the input parameters of the dynamic algorithm on a Multi-Complex $\mathcal{M}$ are a resolution filter R and a sub-MC $\mathcal{M}'$ of $\mathcal{M}$.

The intuitive idea is performing the minimum possible amount of work necessary to transform $\mathcal{M}'$ into the sub-MC $\mathcal{M}''$ providing the solution for the query new parameters. The "amount of work" is quantified in terms of the number of nodes that must be included into, or eliminated from $\mathcal{M}'$ in order to transform $\mathcal{M}'$ into $\mathcal{M}''$ (or, in other words, the number of modifications which must be done, or undone, on the front cell complex of $\mathcal{M}'$ to transform it into the solution of new query).

The elementary operations here are the inclusion of a node into the current sub-MC (already explained in Section 5.4.1), and the elimination of a node from it. A node C can be eliminated from a sub-MC $\mathcal{M}'$ only if C is a leaf in $\mathcal{M}'$ (or, equivalently, if the MC-front of $\mathcal{M}'$ contains all the arcs leaving C).

In the reminder, we first present a dynamic algorithm for the min-size query, and then one for best-resolution one.

5.6.1 A Dynamic Algorithm for the Min-Size Global Query

Let $\mathcal{M}'$ be the current sub-MC. Given a node C which is the destination of a front arc of $\mathcal{M}'$, we call the inclusion of C into $\mathcal{M}'$ *necessary* if $\Pi_1(C) \cap \mathrm{front}(\mathcal{M}')$ contains some k-cell which is not feasible. Given a node C such that all arcs entering C belong to the MC-front of $\mathcal{M}'$, we say that the elimination of C from $\mathcal{M}'$ is *allowed* if all k-cells in $\Pi_1(C)$ are feasible.

Intuitively, necessary inclusions of nodes are aimed at refining the current front complex where it is too coarse with respect to the new parameters; allowed eliminations of nodes are aimed at coarsening the current front complex where this can be done without violating the given requirements.

The dynamic algorithm returns the front cell complex of a sub-MC $\mathcal{M}''$ obtained from the given one $\mathcal{M}'$ by performing a sequence of elementary operations, where each elementary operation is either a necessary inclusion or an allowed elimination of a node.

Indeed, the output of a dynamic algorithm depends on the order in which elementary operations (i.e., inclusions and eliminations) are performed. We consider three basic approaches:

1. perform all allowed eliminations first, then all necessary inclusions;
2. perform all necessary inclusions first, then all allowed eliminations;
3. interleave necessary inclusions and allowed eliminations.

Approach 1 guarantees the smallest possible sub-MC which can be built by applying a sequence of elementary operations. However, it may happen that some eliminated node is re-included afterwards, since it is an ancestor of a node object of a necessary inclusion. A working example of this approach is shown in Figure 5.11.

Approach 2 may produce a sub-MC larger than approach 1, since it prevents the elimination of those leaves C of the input sub-MC, such that all k-cells of $\Pi_1(C)$ are feasible, but some k-cell of $\Pi_2(C)$ is not feasible. In approach 1, the unfeasible cells of $\Pi_2(C)$ are removed from the current front complex while excluding C. In approach 2, they are eliminated by including some of the children of C; the elimination of such children of C is not allowed, thus they will remain in the output. However, in approach 2, an included node cannot be eliminated afterwards since its elimination is not allowed. A working example of approach 2 is shown in Figure 5.12; in the figures, node C is node 1: it is eliminated in Figure 5.11, while in Figure 5.12 its child node (node 2) is included.

Approach 3 has all the drawbacks of both approaches 1 and 2 (i.e., it may first eliminate and then re-include the same node, and may produce a sub-MC larger than necessary), and none of the advantages.

Since the idea is minimizing the amount of work, we choose approach 2 (we reject approach 1 because it "wastes" some work in doing and undoing the same operations).

Thus, starting from the input sub-MC, the dynamic algorithm determines the sub-MC providing the solution of the query by performing two phases:

1. *Expansion*: include nodes into the current sub-MC until all k-cells in its front complex are feasible (looking at the enriched MC, in this phase the current MC-front is advanced towards the drain).

2. *Contraction*: eliminate nodes from current sub-MC as long as this can be done without getting some unfeasible k-cell (looking at the enriched MC, in this phase the current MC-front is moved back towards the root).

The expansion phase performs a maximal sequence of necessary inclusions; it works as the main loop of the corresponding static algorithm (see step 2 of Algorithm 5.2).

The contraction phase iteratively removes one node from the current sub-MC (equivalently, it sweeps the current MC-front behind that node), as long as this can be done without getting some unfeasible k-cell in its front complex (i.e, it performs a maximal sequence of allowed eliminations).

More precisely, the dynamic algorithm for the min-size global query works as follows:

Algorithm 5.6
Input: a Multi-Complex $\mathcal{M}$, a resolution filter R, and a sub-MC $\mathcal{M}'$ of $\mathcal{M}$.
Output: a cell complex Γ.

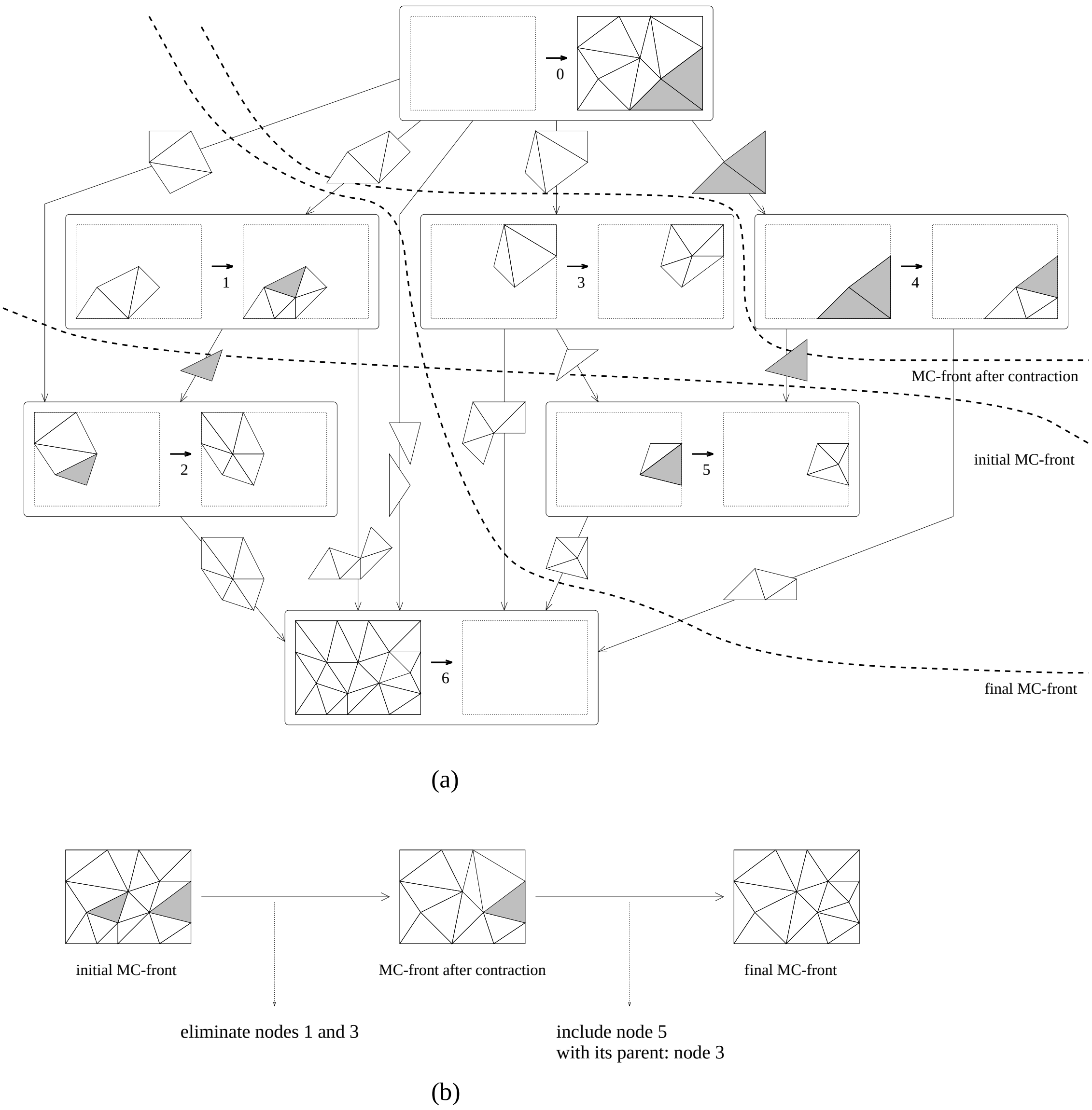
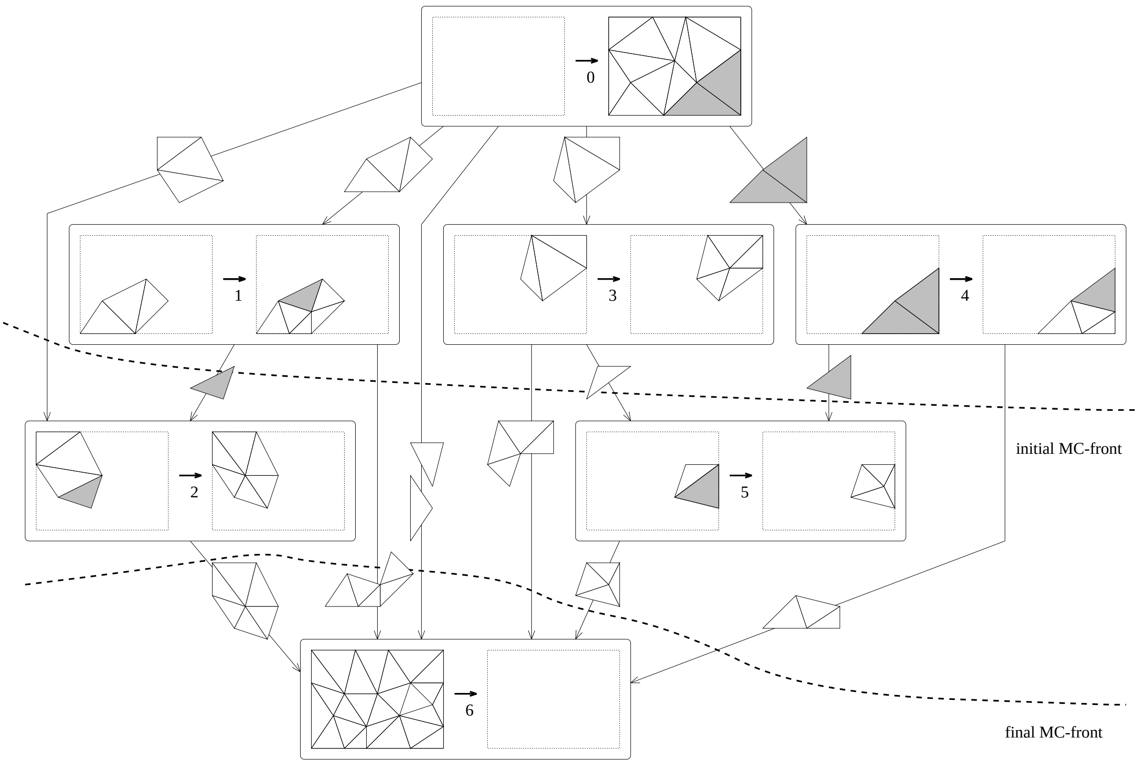
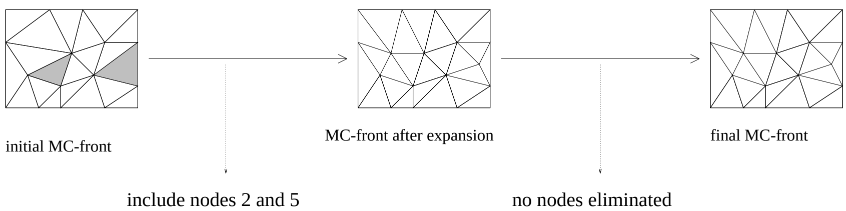


Figure 5.11: The dynamic approach performing contraction first, then expansion. (a) The enriched MC. The resolution filter is > 0 on grey triangles. The initial MC-front, the MC-front after contraction, and the final one are shown. (b) The evolution of the current triangulation during contraction and expansion.

1. [Initialization] Initialize the current sub-MC $\mathcal{M}''$ with $\mathcal{M}'$.
2. [Expansion] While the front cell complex of $\mathcal{M}''$ contains some k-cell that is not feasible:
   a. choose a node C such that $\Pi_1(C)$ contains a k-cell γ of the current front complex which is not feasible;



(a)



(b)

Figure 5.12: The dynamic approach performing expansion first, then contraction, on the same input as Figure 5.11. (a) The enriched MC. The resolution filter is > 0 on grey triangles. The initial and the final MC-fronts are shown. (b) The evolution of the current triangulation during expansion and contraction.

b. include C and all its ancestors into the current sub-MC $\mathcal{M}''$, along with all its ancestors that are not already in $\mathcal{M}''$, by applying Algorithm 5.1.

3. [Contraction] While a leaf C of $\mathcal{M}''$ exists such that all k-cells in $\Pi_1(C)$ are feasible:

a. choose one of the leaves C as above;

b. eliminate C from the current sub-MC $\mathcal{M}''$ (equivalently, sweep the current MC-front behind C).

4. [Output] Return the front cell complex Γ of $\mathcal{M}''$.

Figure 5.12 illustrates a working example of Algorithm 5.6.

Step 2 is the same as the corresponding step 2 in Algorithm 5.2. Thus, it adds the minimum possible number of nodes to the input sub-MC necessary to achieve that all top-cells in its front complex are feasible.

Step 3 removes from the current sub-MC the maximum possible number of nodes while maintaining the property that top-cells in its front complex are feasible. It is easy to see that the result is unique in cases when many nodes n can be eliminated, regardless the order in which they are eliminated.

5.6.1.1 Correctness of the Algorithm

The result of the dynamic algorithm is correct with respect to the following requirements:

Theorem 5.6 *Given a Multi-Complex $\mathcal{M}$, a resolution filter R, and a sub-MC $\mathcal{M}'$ of $\mathcal{M}$, the result of Algorithm 5.6 is a cell complex Γ such that:*

1. *Γ is global combined cell complex of $\mathcal{M}$ corresponding to the front cell complex of a sub-MC $\mathcal{M}''$ of $\mathcal{M}$;*
2. *all k-cells of Γ are feasible with respect to the resolution filter R;*
3. *for every leaf C of $\mathcal{M}''$, the sub-MC obtained by removing C from $\mathcal{M}''$ does not satisfy condition 2 (i.e., no allowed eliminations of nodes exist for $\mathcal{M}''$);*

Condition 3 in the above theorem is looser than the corresponding condition in Definition 5.3: there may exist a sub-MC satisfying condition 2, and having a node set completely contained in that of $\mathcal{M}''$, which can be obtained by removing more than one node from $\mathcal{M}''$.

Let us examine this problem more carefully. We note that the value of the resolution filter on the k-cells of a Multi-Complex does not, in general, monotonically decrease while descending the DAG from the root. For instance, let us consider resolution filters induced by error threshold functions. The algorithms which generate the sequences of refinement patterns from which to build an MC are heuristics, and do not guarantee a monotonic behavior in the approximation errors.

Thus, adding a node to the a sub-MC $\mathcal{M}'$ does not always increase the accuracy of its front cell complex, and removing a node from $\mathcal{M}'$ does not always decrease it. It may happen that

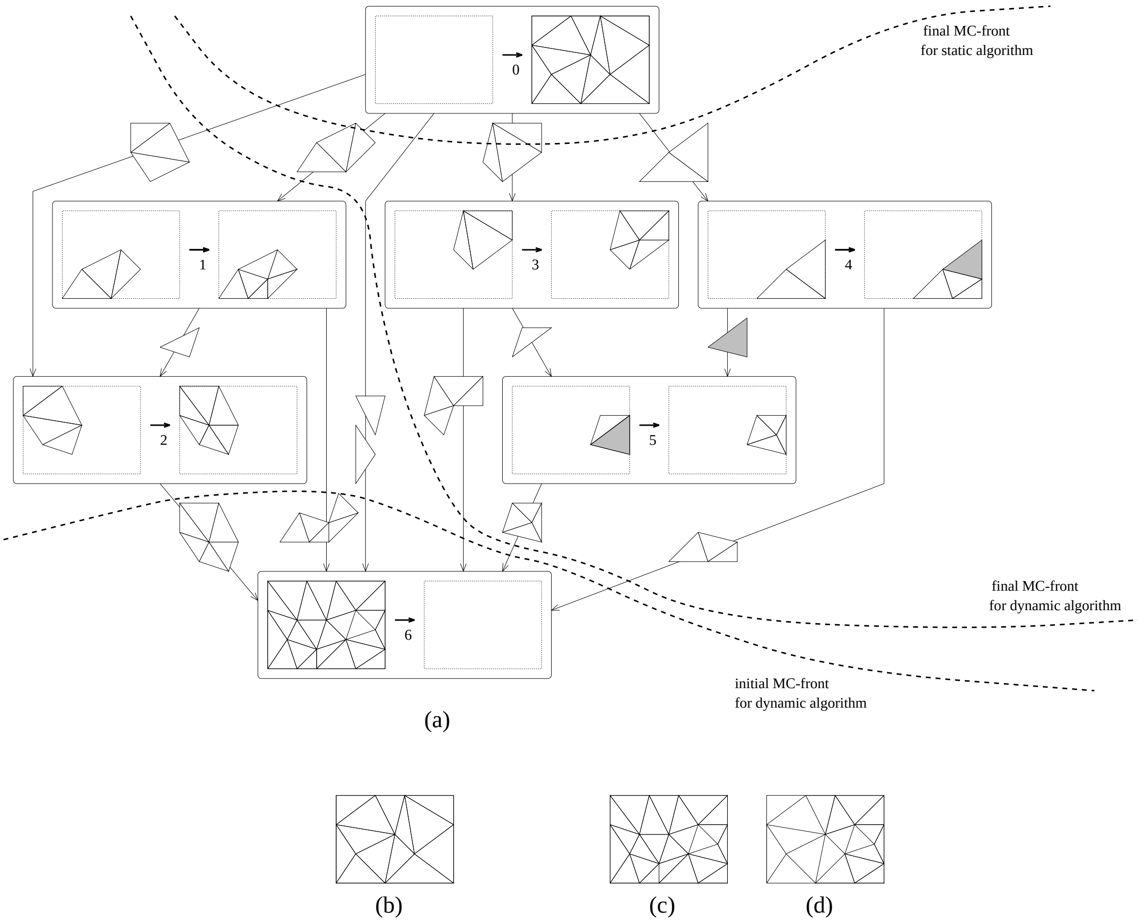


Figure 5.13: An example where the static and the dynamic global query algorithms give different results for the same parameters. (a) The enriched MC. The resolution filter is > 0 on grey triangles. The initial MC-front for the dynamic algorithm, and the final MC-fronts for both algorithms are shown. (b) The extracted mesh of the basic algorithm; (c), (d) the initial and final mesh for the dynamic one. The basic algorithm starts from the triangulation at the root, and stops because all triangles are feasible. The dynamic algorithm eliminates nodes 2 and 1, but then it cannot eliminate any more node because eliminating node 5 would introduce a non-feasible triangle in the mesh.

the elimination of a single node C is not allowed (since it would introduce a violation of the resolution filter on some cell), but eliminating C together with a set of its ancestors would restore a feasible situation. Thus, Algorithm 5.6 gets blocked on a sub-MC that properly contains the one generated by Algorithm 5.2 in its top-down descent of the DAG. In this sense, the result of Algorithm 5.6 is *minimal*, but not *minimum*, as the one of Algorithm 5.2.

Figure 5.13 shows an example where the static and the dynamic algorithms give different results for the same resolution filter.

However, experiments show that this hardly happens, and in any case the difference in terms of number of cells between the two solutions is very little (see Section 5.7).

Now, we prove Theorem 5.6.

Proof Γ is a global combined cell complex because each elementary operation (inclusion or elimination of a node) maintains a consistent sub-MC, and the algorithm returns the front of such a sub-MC.

It is easy to see that, at the end of step 2, all k-cells are feasible, and that step 3 does not introduce any k-cell which is not feasible; thus, all k-cells of Γ are feasible.

From the control condition of the loop at step 3, it follows that no allowed eliminations exist for $\mathcal{M}''$. $\square$

5.6.1.2 Algorithm Complexity

As in the static case, we evaluate the computational complexity of Algorithm 5.6 parametrically on the basis of the costs of the auxiliary operations involved.

Let m^+ and m^- the number of iterations performed in the expansion and in the contraction loop, respectively. Note that m^+ is bounded from above by the number of nodes added during the expansion phase, and m^- is equal to the number of nodes eliminated in the contraction phase.

The cost of step 2 is estimated as in the static case and (using the same notations), it is in $O(\sum_{i=1}^{m^+}(\mathcal{T}_i^{(check)} + \mathcal{T}_i^{(find)}) + \sum_{C\in(\mathcal{M}''-\mathcal{M}')}(\mathcal{T}^{(incl)}(C)))$.

For expressing the cost of step 3, we denote with $\mathcal{T}_i^{(dcheck)}$ the time needed to check the condition of loop 3, and with $\mathcal{T}_i^{(dfind)}$ the cost of finding the node in step 3.a. We denote with $\mathcal{T}^{(elim)}(C)$ the cost of eliminating a node C from the current sub-MC. With this notation, the total cost of step 3 is in

$$O(\sum_{i=1}^{m^-}(_i^{(dcheck)}+\mathcal{T}_i^{(dfind)}) + \sum_{C\in(\mathcal{M}'-\mathcal{M}'')}(\mathcal{T}^{(elim)}(C))).$$

5.6.2 A Dynamic Algorithm for the Best-Resolution Global Query

As in Algorithm 5.6, we start from the input sub-MC, and perform a sequence of elementary operations where each operation either includes or eliminates a node from the current sub-MC. We recall the three basic approaches outlined in Section 5.6.1: performing all eliminations first, performing all inclusions first, or interleaving eliminations and inclusions. Since the size of the extracted complex must be kept under a fixed bound, here we choose approach 1, as the one that permits to obtain a smaller size.

Thus, starting from the input sub-MC, the dynamic algorithm determines the sub-MC which provides the solution of the dynamic query by performing first a contraction, and then an expansion phase.

In both phases, the elimination and the inclusion of nodes are driven by the value of R:

1. In the *contraction* phase, we iteratively eliminate the leaf C which minimizes the value of $\max\{R(\gamma) \mid \gamma \in \Pi_1(C)^{(k)}\}$ among all leaves of the current sub-MC.

   Intuitively, this strategy corresponds to letting the maximum of R grow as slowly as possible.

   Unlike in Algorithm 5.6, we can perform unallowed eliminations (i.e., eliminations of leaves C such that $R(\gamma) > 0$ for some k-cell of $\Pi_1(C)$). We do this in two situations: when the size of the current solution is larger than the given bound b (to reduce the size), and, when the current size is $\leq b$, if the elimination does not increase the maximum of R over the current cell complex.

   We continue the contraction phase as long as the size of the current cell complex is larger than b, or until the minimum value of $\max\{R(\gamma) \mid \gamma \in \Pi_1(C)^{(k)}\}$, over all leaves C, becomes larger than the maximum value of R on the cells of the current complex. At this point, eliminating another node would increase the maximum of R over the current complex.

2. In the *expansion* phase, we iteratively include into the current sub-MC the node C, destination of an arc of the current MC-front, such that $\Pi_1(C)$ contains the cell γ of the current complex which maximizes the value of $R(\gamma)$.

   As in Algorithm 5.3, if including C temporarily increases the maximum of R, then we continue including some of its children, and undo all modifications if during this process the current cell complex exceeds the given size bound.

The expansion phase works as the main loop of the corresponding static algorithm (see step 2 of Algorithm 5.3).

The contraction phase differs from that of Algorithm 5.6 because of the criterion for selecting the next node to eliminate (in 5.6, an arbitrary node is eliminated), and because non-allowed eliminations can be performed as well.

More precisely, the dynamic algorithm performs the following steps:

Algorithm 5.7
Input: a Multi-Complex $\mathcal{M}$, a resolution filter R, a size bound b, and a sub-MC $\mathcal{M}'$ of $\mathcal{M}$.
Output: a cell complex Γ.

1. [Initialization]: as in Algorithm 5.6.

2. [Contraction] While the size of the current mesh is greater than b, or the minimum value of $\max\{R(\gamma) \mid \gamma \in \Pi_1(C)^{(k)}\}$ over all leaves C of the current sub-MC is smaller than, or equal to the maximum value of R over the cells of the current front complex:

   a. find the leaf C corresponding to the minimum value of $\max\{R(\gamma) \mid \gamma \in \Pi_1(C)^{(k)}\}$;

   b. eliminate C from the current sub-MC $\mathcal{M}''$.

3. [Expansion]: as in Algorithm 5.3.

4. [Output] generate Γ as in Algorithm 5.6.

5.6.2.1 Correctness of the Algorithm

The result of the Algorithm 5.7 is correct with respect to the following requirements:

Theorem 5.7 *Given a Multi-Complex $\mathcal{M}$, a resolution filter R, a bound b on the output size, and a sub-MC $\mathcal{M}'$ of $\mathcal{M}$, the result of Algorithm 5.7 is a cell complex Γ such that:*

1. *Γ is global combined cell complex of $\mathcal{M}$, corresponding to the front cell complex of a sub-MC $\mathcal{M}''$ of $\mathcal{M}$;*

2. *the size of γ is less or equal to b;*

3. *Γ minimizes the value of $\max\{R(\gamma) \mid \gamma \in \Gamma^{(k)}\}$ among the front cell complexes of the sub-MCs obtained from $\mathcal{M}''$ by eliminating one leaf.*

Condition 3 in the above theorem is looser than the corresponding condition 3 in Definition 5.4: there may exist a sub-MC having a smaller value of R on the cells of its front complex, and a node set completely contained in that of $\mathcal{M}''$, which can be obtained from $\mathcal{M}''$ by removing more than one node. This happens because Algorithm 5.7 may get blocked on a sub-MC that properly contains the one generated by Algorithm 5.3 in its top-down descent of the DAG. The cell complex output by Algorithm 5.7 may approximate the given resolution requirements worse than the cell complex obtained from running the corresponding static algorithm with the same input parameters.

Now, we prove Theorem 5.7.

Proof Γ is a global combined cell complex because each elementary operation (inclusion or elimination of a node) maintains a consistent sub-MC, and the algorithm returns the front of such a sub-MC.

Γ has a size $\leq b$ because elimination of nodes in step 2 continues until the current size gets $\leq b$, and step 3 does not increase the size over b (as proven in Theorem 5.2).

While the current size is within the bound, each elimination of a node performed in step 2 does not increase the maximum of R over the cells of the current complex. Thus, eliminating any leaf from the current sub-MC after step 3 would increase the maximum of R.

It remains to show that eliminating a leaf added in step 3 increases the maximum of R. This is true because nodes added in step 3 are either nodes C such that $\max\{R(\gamma) \mid \gamma \in \Pi_1(C), \} > \max\{R(\gamma) \mid \gamma \in \Pi_2(C)\}$ (the nodes whose inclusion is directly invoked in step 3) or ancestors of such nodes (the ones recursively included by Algorithm 5.1). A leaf must be of the first type, thus its elimination increases the maximum of R over the current complex. $\square$

5.6.2.2 Algorithm Complexity

The parametric evaluation of the time complexity is unchanged with respect to Algorithm 5.6, but in this variant the cost $\mathcal{T}^{(find)}$ and the cost $\mathcal{T}_i^{(dfind)}$ of finding the next node to include and to eliminate, respectively, are likely to be higher. Moreover, we must take into account the cost of possible work done and undone in step 3 to backtrack if the size of the resulting complex exceeds b (however, this may happen at most once during each execution of the algorithm).

5.7 Experimental Results

In this section, we show experimental results obtained from a prototype implementation of query algorithms representative of the three main approaches described in this Chapter: the basic algorithm for the global query, the local query algorithm, and the dynamic algorithm for the global query. Only the variants minimizing the size for a given resolution filter are demonstrated (i.e., Algorithms 5.2, 5.5 and 5.6). The corresponding variants optimizing the resolution within a given size bound have not been implemented yet.

Experiments have been performed on an SGI Indigo2 workstation (R10000, 195 Mhz, 128 Mb RAM). We present experimental results on two MCs that are representative of the results we obtained on all real world examples:

- an MC representing a terrain (two-dimensional scalar field), in the area of Mount Marcy, NY, USA (courtesy U.S. Geological Survey);
- an MC representing a free-form surface consisting of a bunny (courtesy Department of Computer Science, Stanford University).

For Mount Marcy, the resolution filter is induced by an error threshold function linearly increasing with distance from a viewpoint which moves along a rectilinear trajectory. For the

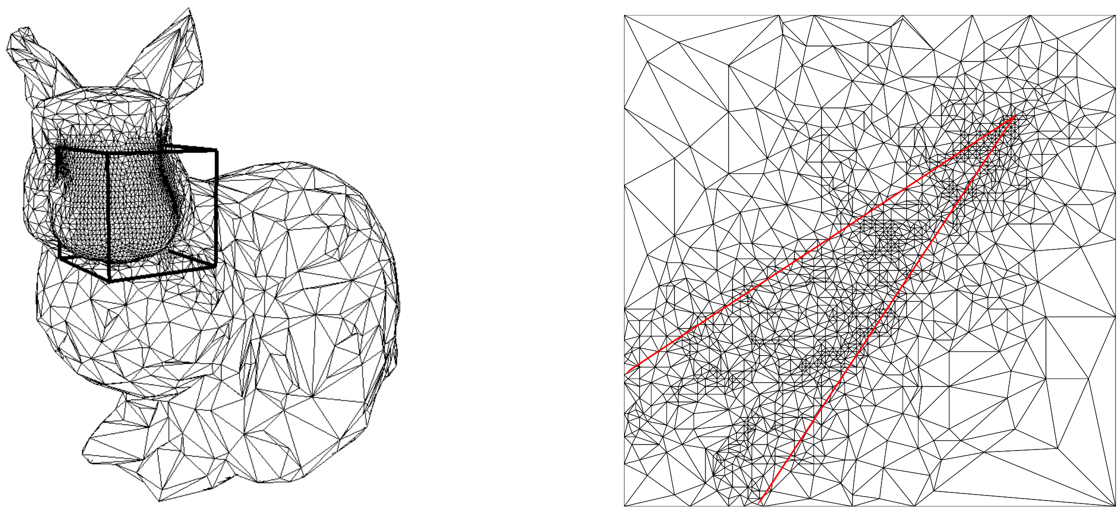


Figure 5.14: Meshes obtained as the result of the global query from the MC representing bunny and from the MC representing Mount Marcy.

local query, the focus set is a sector centered at the viewpoint, and having its axis along the trajectory; for the global query, the resolution filter has been restricted to an area of interest coincident with the above focus set. Both the resolution filter and the focus set are defined on the two-dimensional domain of the terrain and evaluated on triangles in $\mathbb{E}^2$.

For the bunny, we have considered a resolution filter corresponding to a constant error threshold equal to zero, and, for the local query, a focus set consisting of an axis-parallel box which moves along a trajectory in $\mathbb{E}^3$. The same focus set has been used as an area of interest for restricting the resolution filter in the global query.

The two-dimensional MC representing the domain of Mount Marcy is conservative, thus the query algorithms have been run with consideration to (strongly) active triangles. The Bunny MC is not conservative, and loosely active triangles have been considered.

Also, note that, since Mount Marcy is a conservative MC, the local algorithm is guaranteed to return a cell complex while this does not necessarily hold for the Bunny MC (however, in all cases the output was a triangle mesh).

Examples of extracted meshes from both MCs are shown in Figure 5.14. Figures 5.15 and 5.16 show statistics about the number of output triangles and the number of triangles processed by the three algorithms. Figures 5.17 and 5.18 show execution times.

In the experiments, the size of the output complex of the local algorithm is almost identical to the number of active triangles in the output of the basic global algorithm (the difference is a few units). The output size of the two global query algorithms (the basic and the dynamic one) is identical in almost all cases. The number of active output triangles in the Marcy

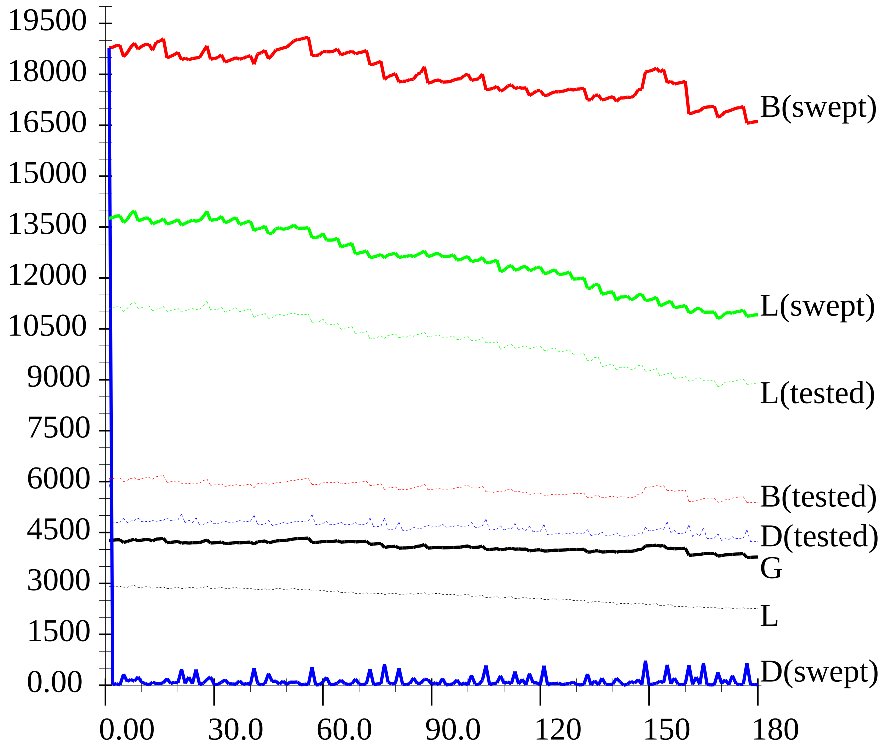


Figure 5.15: Statistics related to the three query algorithms on Mt. Marcy. Black lines represent the number of output triangles: solid for the global query (labelled `G`), and dotted for the local one (labelled `L`). Red, blue, and green solid lines represent the number of triangles swept by the cut in the basic (global) algorithm, the dynamic (global) algorithm, and the local algorithm, the corresponding labels are `B`, `D`, and `L`, respectively. Red, blue, and green dotted lines represent the for triangles tested against the resoltion filter (and focus set) in the three algorithms (labelled `B`, `D`, and `L` as above).

experiment is slightly decreasing as the viewpoint moves toward the opposite border of the domain. In the bunny experiment, the output size first increases and then decreases as the focus box approaches the surface, traverses it, and leaves it again.

In the two global algorithms, the number of tested triangles is roughly proportional to the output size, with a much lower constant in the dynamic case; in the dynamic case, however, more tests are performed when the output mesh changes considerably, and especially when it is coarsened.

The number of triangles swept by the basic algorithm is proportional to the size of the output mesh, while for the dynamic one it is very low when the mesh changes gradually. The number of tested and swept triangles for local algorithm seems proportional to the number of output triangles, but with a rather big constant.

The time difference between the two global algorithms tends to be a constant: the work here

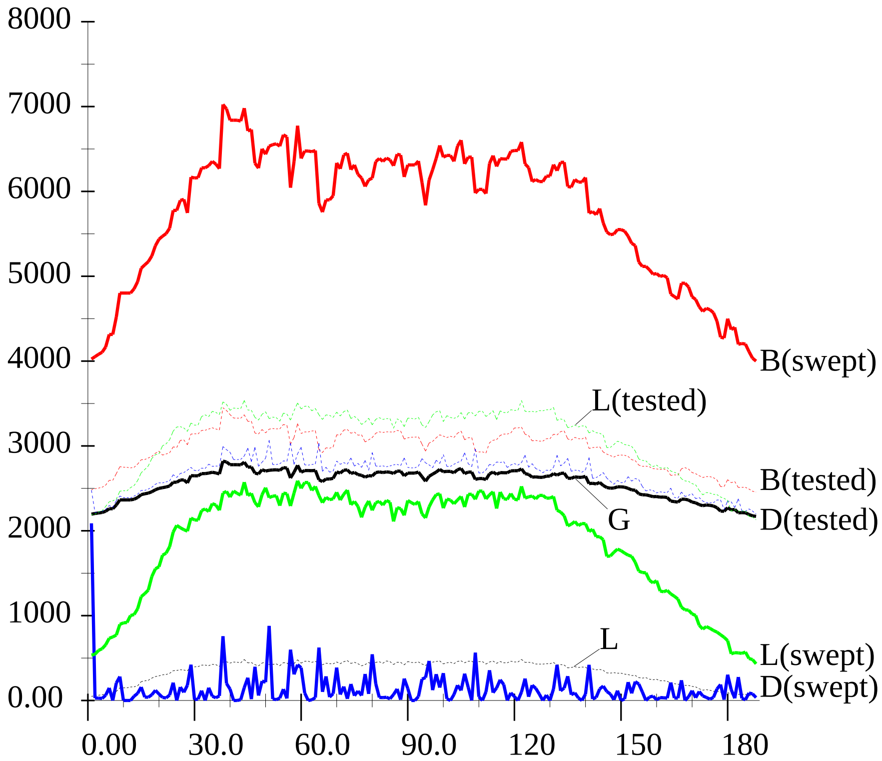


Figure 5.16: Statistics related to the three query algorithms on bunny. Line styles, colors, and labels have the same meaning as in Figure 5.15.

is dominated by testing conditions on the output mesh. Local query time is much lower for the bunny, while it is competitive with the dynamic time for Marcy. This happens because the focus set used on the terrain is less selective. In any case, it is generally proportional to the number of active and feasible triangles reported.

5.8 A Specification of Query Operations on an MC

The Multi-Complex is the core of a *data type*, i.e., a collection of sets of objects connected by operations. The Multi-Complex data type provides a standard framework for modeling the management of multiple resolutions of cell complexes.

In Chapters 4 and 5 up to this point, we have considered Multi-Complexes with generic cells. Here, we focus our attention on *simplicial* k-dimensional Multi-Complexes (i.e., MCs where all cells are simplexes, see Section 4.3) embedded in $\mathbb{E}^d$. For short, we will call them just Multi-Complexes, with the implicit assumption that they are simplicial.

We divide the operations acting on a Multi-Complex into three levels:

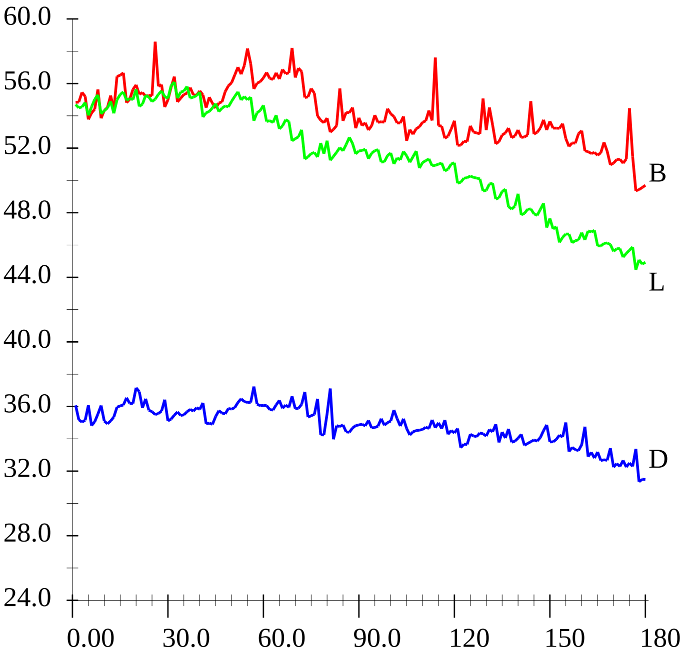


Figure 5.17: Execution times (in milliseconds) of the three query algorithms on Mt. Marcy. Red, blue and green lines (labelled B, D, and L) denote the basic, dynamic and local algorithm, respectively.

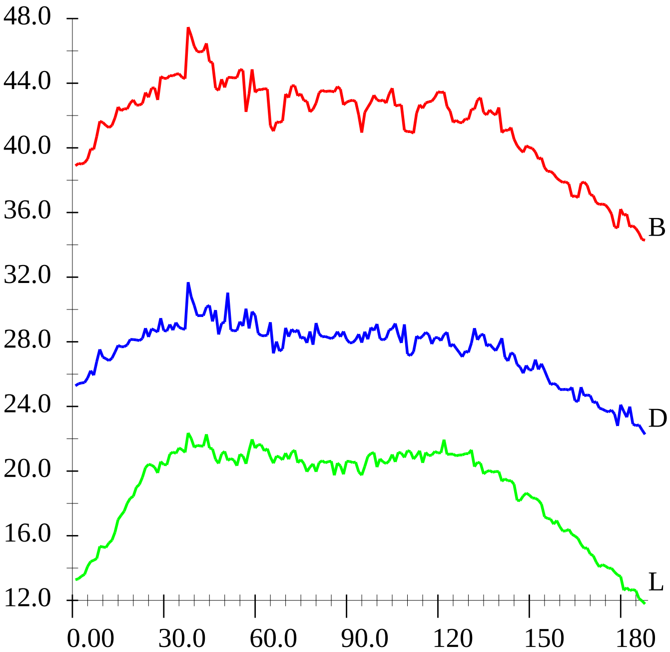


Figure 5.18: Execution times (in milliseconds) of the three query algorithms on bunny. Red, blue and green lines (labelled B, D, and L) denote the basic, dynamic and local algorithm, respectively.

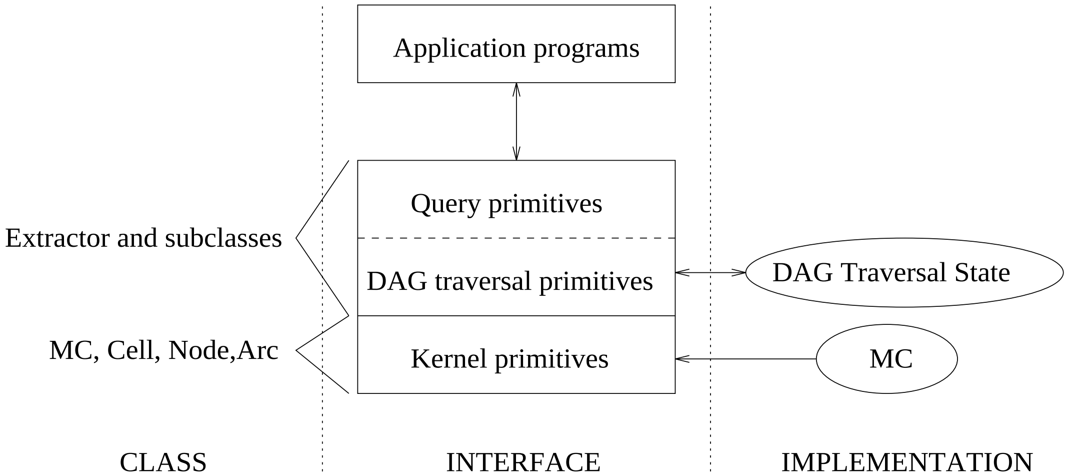


Figure 5.19: The three levels of query operations in the MC data type.

1. *query operations*, which define the interface of an MC towards the application tasks;
2. *DAG traversal operations*, which provide the inner implementation of query operations;
3. *kernel operations*, i.e., basic primitive operations acting on the structure of an MC.

The purpose of *kernel operations* is hiding the specific data structure used to represent a Multi-Complex: they provide a standard interface on which operations of the upper levels can be defined.

Query operations implement the various query algorithms defined earlier in this chapter. We consider here the basic algorithm for the global query (Algorithm 5.2), the dynamic algorithm for the global query (Algorithm 5.6) and the algorithm for the local query (Algorithm 5.5); for the sake of simplicity and brevity, we do not consider the corresponding three variants which extract a mesh of bounded size.

DAG traversal operations are defined on top of the kernel level and used only for implementing query operations. Such operations are concerned with the management of the internal state of DAG traversal in the various algorithms.

Figure 5.19 shows the relations between the three levels of operations.

The restriction to simplicial MCs is fundamental only when specifying kernel primitives concerned with the geometric description of cells; for all other primitives, the shape of cells is not relevant.

Operations will be defined with respect to the *enriched* MC because it allows a uniform treatment of all top-cells of an MC. We recall that, in the enriched MC, every top-cell is in $\Pi_1(C)$ and in $\Pi_2(C')$ for *exactly one pair* of MC-nodes C and C'; this feature makes the definition of operations cleaner, and subsequent implementations (presented in Chapter 6) easier. In this section, we will use the term Multi-Complex improperly, meaning its enriched version. Moreover, the term simplex / cell will denote a top-simplex / top-cell, unless otherwise specified.

We define the operations of the various levels using an object-oriented notation. This does not mean that the implementation of the MC must be done in an object-oriented language. It is simply a clear way to specify the interface of each type of entity involved in the MC data type. We remark that this is not intended to be a formal specification.

5.8.1 Preliminaries

A k-dimensional simplicial Multi-Complex is characterized by the following elements:

- The *top-cells* involved in the model, with their approximation errors. In our case, these cells are *k-simplexes*.
- The *adjacency relations* between top-cells. Such relations describe the way in which pairs of k-simplexes match along their $(k-1)$-faces, which represents a precondition to assemble them into simplicial complexes.
- The *nodes*, which represent *modification patterns*. They specify which sets of cells provide alternative and mutually-exclusive representations of the same portion of an object.
- The *arcs*, which represent *relations of direct dependency* between nodes. The partial order induced by such relations determines the possible ways to combine the cells of the model into consistent representations of the target object.

Within an application, additional information may be attached to a simplex and to its vertices. For instance, in a visualization application, additional attributes may be normals, textures, colors, material properties, etc. Such additional aspects are not considered here.

We organize the kernel level into four classes of objects:

- the class `Cell` of *top-simplices* of an (enriched simplicial) MC;
- the class `Node` of nodes of an (enriched simplicial) MC;
- the class `Arc` of arcs of an (enriched simplicial) MC;
- the class `MC` of (enriched simplicial) MCs.

The interfaces of such classes contain operations that relate them to each other. The connections between the various classes are illustrated graphically in Figure 5.20 (a).

We organize DAG traversal and query operations in a class, called `Extractor`, where query operations are the public interface of the class (available to the user) and DAG traversal ones are

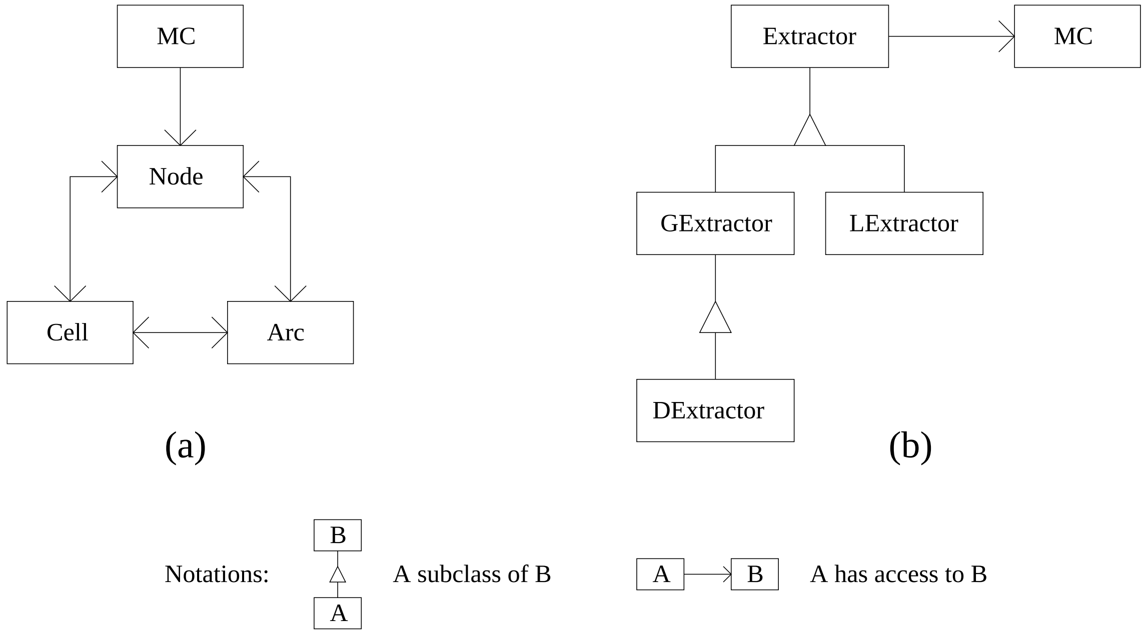


Figure 5.20: The relations between the various classes of the MC data type: (a) kernel classes implementing the MC; (b) classes implementing query algorithms.

private operations of the class and of its subclasses. An object of class `Extractor` is a container for the internal state of a DAG traversal. We have chosen to separate permanent information about an MC (contained in classes `MC`, `Node`, `Arc` and `Cell`) from temporary information about the DAG traversal state (contained in class `Extractor`) because they have a different lifetime: information about an MC survive the execution of a program accessing it, while information about the internal state of query algorithms must be cleaned. This choice also allows sharing the data describing an MC among different query processes active on it.

Since there are several query algorithms, class `Extractor` is a base class and its subclasses provide the implementations of the various query algorithms. We define two main subclasses, corresponding to global and to local query algorithms, respectively, that we denote with the names of `GExtractor` and `LExtractor`, where the prefixed character `G` stands for "Global", `L` stands for "Local". Class `GExtractor` has a further subclass, corresponding to the dynamic variant, and named `DExtractor`, where `D` stands for "Dynamic". The relations between the various classes are illustrated graphically in Figure 5.20 (b).

In addition to the classes of objects introduced above, our specification uses other data types:

- points of d-dimensional Euclidean space, denoted with $\mathbb{E}^d$;
- real numbers, denoted with $\mathbb{R}$;
- Booleans, denoted with $\mathbb{B}$;

- the type of $(k-1)$-simplexes in $\mathbb{E}^d$ (i.e., the type of the $(k-1)$-faces of objects of class `Cell`), that we denote with `Face`;
- the type of functions defined over the objects of class `Cell` and assuming values in $\mathbb{R}$ (i.e., the type of resolution filters), that we denote with `Filter`;
- the type of subsets of $\mathbb{E}^d$ (i.e., the type the focus sets), that we denote with `Focus`;
- finally, we use the notation $\texttt{Parts}(X)$ to denote the type of all finite sets of elements of type X, and $\texttt{Parts}^{(s)}(X)$ for the type of all subsets of set X, of cardinality equal to s.

5.8.2 Kernel Primitives

Kernel operations hide the data structure implementing an MC and provide a standard set of primitives to access it. Such primitives are used in the implementation of the query operations at the upper levels. Kernel primitives correspond to the interface operations of the kernel classes `Cell`, `Node`, `Arc` and `MC`.

5.8.2.1 MC Simplexes

The class `Cell` represents a top-simplex γ *within an MC*. The interface of class `Cell` contains the following public primitives:

1. $\texttt{CellVertices}() \longrightarrow \texttt{Parts}^{(k+1)}(\mathbb{E}^d)$.

   $\gamma.\texttt{CellVertices}()$: returns the set of vertices generating simplex γ.

2. $\texttt{CellError}() \longrightarrow \mathbb{R}$.

   $\gamma.\texttt{CellError}()$: returns the approximation error of γ.

3. $\texttt{CoAdj}_1(\texttt{Face}) \longrightarrow \texttt{Cell}$.

   $\gamma.\texttt{CoAdj}_1(f)$: returns the k-simplex which is in relation co-adjacent$_1$ to γ at f (if it exists).

4. $\texttt{CoAdj}_2(\texttt{Face}) \longrightarrow \texttt{Cell}$.

   $\gamma.\texttt{CoAdj}_2(f)$: returns the k-simplex which is in relation co-adjacent$_2$ to γ at f (if it exists).

5. $\texttt{CntAdj}_{1,2}(\texttt{Face}) \longrightarrow \texttt{Cell}$.

   $\gamma.\texttt{CntAdj}_{1,2}(f)$: returns the k-simplex which is in relation counter-adjacent$_{1,2}$ to γ at f (if it exists).

6. `CntAdj`$_{2,1}$(`Face`) $\longrightarrow$ `Cell`.

   γ.`CntAdj`$_{2,1}(f)$: returns the k-simplex which is in relation counter-adjacent$_{2,1}$ to γ at f (if it exists).

7. `CellSource()` $\longrightarrow$ `Node`.

   γ.`CellSource()`: returns the node C, such that $\gamma \in \Pi_2(C)$, i.e., the source of γ. Note that γ.`CellSource()` may be the root.

8. `CellDest()` $\longrightarrow$ `Node`.

   γ.`CellDest()`: returns the node C, such that $\gamma \in \Pi_1(C)$, i.e., the destination of γ. Note that γ.`CellDest()` may be the drain.

9. `CellArc()` $\longrightarrow$ `Arc`.

   γ.`CellArc()`: returns the arc containing k-simplex γ in its label, i.e., the arc connecting node C_i to C_j, where $\gamma \in \Pi_2(C_i) \cap \Pi_1(C_j)$.

Primitives 1 and 2 provide information concerning a cell in itself. Primitives 3–6 describe the connection of a cell with its adjacent cells in the MC and correspond to the four basic adjacency relations in an MC defined in Section 4.5; they will be called *adjacency primitives* in the reminder of the thesis. Adjacency primitives are *partial operations*, since their result is not always defined.

Primitives 7–9 define the role of a cell in the MC, by relating it to the modification patterns in which it appears and with the arcs representing the direct dependency link that it contributes to define.

5.8.2.2 MC Nodes

The interface of class `Node` contains the following public primitives:

1. `Parents()` $\longrightarrow$ `Parts(Node)`.

   C.`Parents()`: returns the set of parent nodes of C, i.e., the set of modification patterns that directly block C. Note that this set is empty if C is the root.

2. `Children()` $\longrightarrow$ `Parts(Node)`.

   C.`Children()`: returns the set of children nodes of C, i.e., the set of modification patterns directly depending on C. Note that this set is empty if C is the drain.

3. `InArcs()` $\longrightarrow$ `Parts(Arc)`.

   `InArcs`$(\mathcal{M}, C)$: returns the set of arcs entering node C.

4. $\texttt{OutArcs()} \longrightarrow \texttt{Parts(Arc)}$.

   $\texttt{OutArcs}(\mathcal{M}, C)$: returns the set of arcs leaving node C.

5. $\texttt{Cells}_1() \longrightarrow \texttt{Parts(Cell)}$.

   $C.\texttt{Cells}_1()$: returns the set of k-simplices of $\Pi_1(C)$.

6. $\texttt{Cells}_2() \longrightarrow \texttt{Parts(Cell)}$.

   $C.\texttt{Cells}_2()$: returns the set of k-simplices of $\Pi_2(C)$.

Primitives 1–4 are standard primitives on directed graphs. Primitives 5 and 6 provide the two meshes composing the modification pattern represented in an MC-node.

5.8.2.3 MC Arcs

The interface of class `Arc` contains the following public primitives:

1. $\texttt{Source()} \longrightarrow \texttt{Node}$.

   $A.\texttt{Source}()$: returns the source of arc A.

2. $\texttt{Dest()} \longrightarrow \texttt{Node}$.

   $A.\texttt{Dest}()$: returns the destination of arc A.

3. $\texttt{ArcCells()} \longrightarrow \texttt{Parts(Cell)}$.

   $A.\texttt{ArcCells}()$ returns the set of k-simplices labelling arc A.

Primitives 1 and 2 are standard primitives on directed graphs. Primitive 3 provides the cells that determine the direct dependency link encoded in an arc.

5.8.2.4 Multi-Complexes

A Multi-Complex is completely described by its enriched DAG, and by the cells attached to nodes and arcs. An object of class `MC` only needs to provide a mean to “enter” and “exit” the DAG; then, DAG primitives acting on nodes and arcs will provide a way to explore the MC completely. The interface of class `MC` contains only two primitives:

1. $\texttt{Root()} \longrightarrow \texttt{Node}$.

   $\mathcal{M}.\texttt{Root}()$ returns the root node of $\mathcal{M}$.

2. $\texttt{Drain()} \longrightarrow \texttt{Node}$.

   $\mathcal{M}.\texttt{Drain}()$ returns the drain node of $\mathcal{M}$.

5.8.2.5 Properties of Kernel Primitives

In the following we report some useful properties of kernel primitives. The properties which describe the connections between kernel operations are especially interesting for the design of data structures. In fact, they suggest that certain information can be recovered based on other information (although possibly at some cost), and thus a data structure may choose not to store them.

- The parents of a node are the sources of its incoming arcs: $C.\texttt{Parents}() = \cup\{A.\texttt{Source}() \mid A \in C.\texttt{InArcs}()\}$.
- The children of a node are the destinations of its outgoing arcs: $C.\texttt{Children}() = \cup\{A.\texttt{Dest}() \mid A \in C.\texttt{OutArcs}()\}$.

The above two properties hold in any directed graph. They suggest that a data structure does not need to explicitly represent the links of a node to its parents and children if it stores arcs explicitly.

- The simplices associated with an arc $A = (C_i, C_j)$ are the common simplices of $\Pi_2(C_i)$ and $\Pi_1(C_j)$: $A.\texttt{ArcCells}() = A.\texttt{Dest}().\texttt{Cells}_1() \cap A.\texttt{Source}().\texttt{Cells}_2()$.
- The simplices of $\Pi_1(C)$ for a node C are given by the union of those associated with the outgoing arcs of C: $C.\texttt{Cells}_1() = \bigcup\{A.\texttt{ArcCells}() \mid A \in C.\texttt{InArcs}()\}$.
- The simplices of $\Pi_2(C)$ for a node C are given by the union of those associated with the incoming arcs of C: $C.\texttt{Cells}_2() = \bigcup\{A.\texttt{ArcCells}() \mid A \in C.\texttt{OutArcs}()\}$.

The last three properties above suggest that primitives $\texttt{Cells}_1$ and $\texttt{Cells}_2$ can be implemented based on `ArcCells` with a cost linear in the output size (i.e., optimal). Conversely, `ArcCells` can be implemented based on $\texttt{Cells}_1$ and $\texttt{Cells}_2$, but this cannot be done in optimal time. Thus, a data structure may not store the sets $\Pi_1(C)$ and $\Pi_2(C)$ explicitly for each node C, if it stores the incoming and outgoing arcs for each node and the sets of simplices associated with arcs.

- For every arc A, $A.\texttt{ArcCells}() \neq \emptyset$.
- For every simplex γ, there exists exactly one arc A such that $A.\texttt{Source}() = \gamma.\texttt{CellSource}()$ and $A.\texttt{Dest}() = \gamma.\texttt{CellDest}()$.

The two properties above suggest that arcs can be retrieved by looking at the sources and destinations of simplices. Thus, a data structure may not represent arcs explicitly, if it stores $\Pi_1(C)$ and $\Pi_2(C)$ for each node C and the source and destination of each simplex. In this case, each arc A is replaced by as many simplices as those contained in its label (see Figure 5.21).

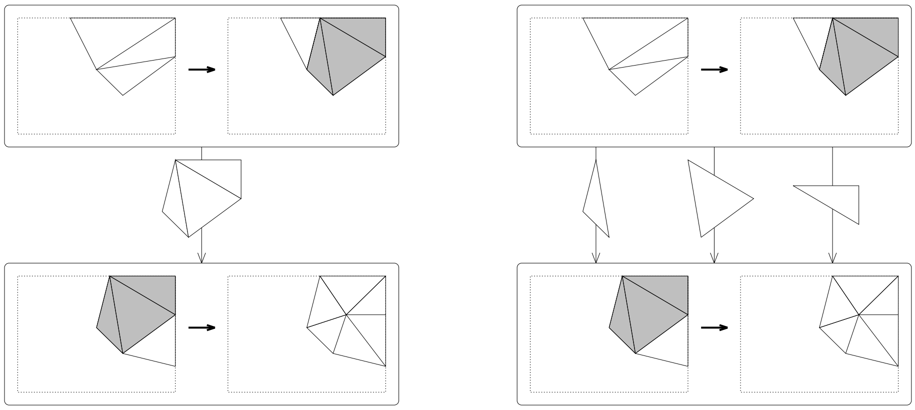


Figure 5.21: Implementing the node-arc-node connection with the node-simplex-node connection replaces an arc with as many representative simplexes as the ones in its label.

5.8.3 DAG Traversal Primitives

DAG traversal primitives are auxiliary operations acting on the internal state of query algorithms. The basic algorithm for the global query and the algorithm for the local query traverse the DAG representing the given MC $\mathcal{M}$ top-down, starting from the root. The dynamic algorithm traverses it both upwards and downwards, starting from a given MC-front.

The global algorithms traverse the DAG while maintaining a current sub-MC $\mathcal{M}'$ of $\mathcal{M}$ at each step. The local algorithm maintains a current pseudo-sub-MC $\mathcal{M}'$. DAG traversal primitives act on (pseudo-)sub-MCs and (pseudo-)MC-fronts.

Here, we provide a set of standard primitives which define the interface for manipulating the current state. The implementation of the current state and of its primitives is addressed in Chapter 6 when describing data structures for an MC.

Formally, the current state is composed by the following elements:

- the MC $\mathcal{M}_{state}$ that is being traversed;
- the current resolution filter R_{state};
- the current focus set F_{state};
- a set $\mathcal{N}$ of nodes of $\mathcal{M}_{state}$;
- a set $\mathcal{A}$ of arcs of $\mathcal{M}_{state}$ connecting a node in $\mathcal{N}$ to a node not in $\mathcal{N}$.

In global query algorithms, set $\mathcal{N}$ contains the nodes of a sub-MC $\mathcal{M}'$ of $\mathcal{M}_{state}$, and set $\mathcal{A}$ contains the arcs of the MC-front of $\mathcal{M}'$. In the local query algorithm, $\mathcal{N}$ and $\mathcal{A}$ describe the set of nodes and of a pseudo-sub-MC and the set of arcs of its pseudo-MC-front.

A *focus set* is present in the internal state of all types of query algorithms: in global ones, it is used as an area of interest for restricting the influence of the resolution filter to active cells only. By default, the focus set F_{state} consists of the whole space and the resolution filter R_{state} is a function returning zero on (i.e., accepting) any cell.

According to the conventions of the object-oriented paradigm, auxiliary operations which manipulate the current state are *protected*, i.e., they can be used only within class `Extractor` itself, or within its subclasses, and are not visible to the user applications. The result of some operations is not completely specified, but it is defined just in terms of *requirements*, leaving a degree of freedom that will be fixed when considering specific data structures for MCs (see Chapter 6).

The auxiliary primitives of the base class `Extractor` permit to:

- set an initial state;
- test whether a node belongs to set $\mathcal{N}$, add a node to $\mathcal{N}$, delete a node from $\mathcal{N}$;
- add an arc to $\mathcal{A}$, remove an arc from $\mathcal{A}$;
- decide whether the current solution must be further refined;
- if the current solution must be further refined, find a node to be included in order to refine it.
- manage the main loop of the expansion phase.

Some of the above operations have a different behavior in the global and in the local case; thus, they will be redefined in each subclass `GExtractor` and `LExtractor`.

Class `DExtractor` adds specific operations for supporting the dynamic algorithm; such operations permit to:

- decide whether the current solution must be coarsened further;
- provided that the current solution must be coarsened further, find a node which to be eliminated in order to coarsen it;
- manage the main loop of the contraction phase.

5.8.3.1 Primitives of the Base Extractor Class

The base class `Extractor` contains the following operations:

- `InitState()`.

  `InitState()`: initializes the DAG traversal state; the initialization is different in the global and local extractor, and thus is specified in the corresponding subclasses.

- `IsInState(Node)` $\longrightarrow \mathbb{B}$.
  `AddToState`$_{node}$`(Node)`.
  `DelFromState`$_{node}$`(Node)`.

  `IsInState`(C): returns true if and only if node C belongs to set $\mathcal{N}$. `AddToState`$_{node}(C)$: adds node C to $\mathcal{N}$. `DelFromState`$_{node}(C)$: deletes node C from $\mathcal{N}$ (if present).

- `AddToState`$_{arc}$`(Arc)`.
  `DelFromState`$_{arc}$`(Arc)`.

  `AddToState`$_{arc}(A)$: adds arc A to $\mathcal{A}$. `DelFromState`$_{arc}(A)$: deletes arc A from $\mathcal{A}$ (if present).

- `ContinueExpansion()` $\longrightarrow \mathbb{B}$.

  `ContinueExpansion()`: checks whether some simplex, among the ones labelling the arcs of set $\mathcal{A}$, is active and not feasible. The implementation does not depend on the specific type of extractor, and can be given in the base class directly. However, since it depends on the data structure used to store the internal state, here we simply specify its requirements:

  > `ContinueExpansion()` = true if and only if set $\{\gamma | \gamma \in A.$`ArcCells()`$, A \in \mathcal{A}, \gamma$ active and not feasible$\}$ is not empty.

- `FindNodeToInclude()` $\longrightarrow$ `Node`.

  `FindNodeToInclude()`: returns a node C such that C is the destination of some active and not feasible simplex in the labels of some arc of set $\mathcal{A}$. Similarly to `ContinueExpansion`, its implementation does not depend on the type of extractor, but depends on the data structure used to represent the MC and the internal state. Here, we simply specify its requirements:

  > `FindNodeToInclude()` = C only if $C \notin \mathcal{N}$, $C.$`InArcs()` $\cap \mathcal{A} \neq \emptyset$, and set $\{\gamma | \gamma \in A.$`ArcCells()`$, A \in C.$`InArcs()` $\cap \mathcal{A}, \gamma$ active and not feasible$\}$ is not empty.

  Note that `FindNodeToInclude` is a partial function: its result is defined if and only if `ContinueExpansion()` returns true in the current state.

- `IncludeNode(Node)`.

  `IncludeNode`(C): updates the DAG traversal state by including node C according to Algorithm 5.1, or 5.4, respectively, for the global and for the local approach. Thus, the implementation is different in the global and in the local case, and is given in the corresponding subclasses. As a prerequirement, node C must not be in $\mathcal{N}$, and at least one of its incoming arcs must be in $\mathcal{A}$.

- `Expansion()`.

  `Expansion()`: updates the state by iteratively including nodes, until all active simplices of the arcs of set $\mathcal{A}$ are feasible. It corresponds to the expansion phase of Algorithms 5.2, 5.6 and 5.5. Its implementation is given in the base class, but it is based on auxiliary operations which are implemented differently in the global and in the local case:

  while `ContinueExpansion()` do
  $\quad C :=$ `FindNodeToInclude()`;
  $\quad$ `IncludeNode(`C`)`;

5.8.3.2 Primitives of the Global Extractor Class

Class `GExtractor` specializes primitives `InitState` and `IncludeNode`, inherited from the base class `Extractor`, in the following way:

- `InitState()`: initializes an internal state where the sub-MC contains the root of $\mathcal{M}_{state}$, and the MC-front contains all the arcs outgoing from the root:

  $\mathcal{N} := \{\mathcal{M}_{state}.\texttt{Root()}\}$;
  $\mathcal{A} := \mathcal{M}_{state}.\texttt{Root().OutArcs()}$;

- `IncludeNode(`C`)`: updates the current state by including node C and all its ancestors according to Algorithm 5.1:

  for every $C' \in C.\texttt{Parents()}$ do
  $\quad$ if not `IsInState(`C'`)` then `IncludeNode(`C'`)`;
  $\texttt{AddToState}_{node}(C)$;
  for every $A \in C.\texttt{OutArcs()}$ do $\texttt{AddToState}_{arc}(A)$;
  for every $A \in C.\texttt{InArcs()}$ do $\texttt{DelFromState}_{arc}(A)$;

5.8.3.3 Primitives of the Dynamic Extractor Class

Class `DExtractor` adds the following primitives:

- `ContinueContraction()` $\longrightarrow \mathbb{B}$.

  `ContinueContraction()`: checks whether some node C exists, such that all simplices of $\Pi_2(C)$ are in the current solution and all simplices in $\Pi_1(C)$ are either not active or feasible.

  The implementation depends on the data structure used to implement the internal state. Here, we simply specify its requirements:

$\texttt{ContinueContraction}()$ = true if and only if
set $\{C \mid C.\texttt{OutArcs}() \subseteq A$, and for all $\gamma \in C.\texttt{Cells}_1(), \gamma$ is not active or feasible$\}$ is not empty.

- $\texttt{FindNodeToEliminate}() \longrightarrow \texttt{Node}$.

  $\texttt{FindNodeToEliminate}()$: returns a node C such that all simplices of $\Pi_2(C)$ are in the current solution, and none of the simplices in $\Pi_1(C)$ is active and not feasible. Its implementation depends on the data structures used for the MC and for the internal state. Here, we simply specify its requirements:

  $\texttt{FindNodeToEliminate}() = C$ only if $C \in \mathcal{N}$, $C.\texttt{OutArcs}() \subseteq \mathcal{A}$, and set $\{\gamma | \gamma \in C.\texttt{Cells}_1(), \gamma$ active and not feasible$\}$ is empty.

  As `FindNodeToInclude`, `FindNodeToEliminate` is a partial function: its result is defined if and only if $\texttt{ContinueContraction}()$ is true in the current state.

- $\texttt{EliminateNode(Node)}$.

  $\texttt{EliminateNode}(C)$: updates the current state by eliminating a node C, according to Algorithm 5.6. As a prerequirement, C must be in $\mathcal{N}$, and all its outgoing arcs must be in $\mathcal{A}$. It is implemented in the following way:

  $\texttt{DelFromState}_{node}(C)$;
  for every $A \in C.\texttt{InArcs}()$ do $\texttt{AddToState}_{arc}(A)$;
  for every $A \in C.\texttt{OutArcs}()$ do $\texttt{DelFromState}_{arc}(A)$;

- $\texttt{Contraction}()$.

  $\texttt{Contraction}()$: corresponds to the main loop of the contraction stage of Algorithm 5.6. It eliminates nodes from a given sub-MC as long as this can be done without getting some unfeasible active simplex in its front complex:

  while $\texttt{ContinueContraction}()$ do
  $\quad C := \texttt{FindNodeToEliminate}()$;
  $\quad \texttt{EliminateNode}(C)$;

5.8.3.4 Primitives of the Local Extractor Class

Class `LExtractor` specializes operations `InitState` and `IncludeNode`, inherited from the base class, in the following way:

- $\texttt{InitState}()$: initializes an internal state where the pseudo-sub-MC contains the root of $\mathcal{M}_{state}$, and the pseudo-MC-front contains all the arcs outgoing from the root that contain some active simplex in their label:

$\mathcal{N} = \{\mathcal{M}_{state}.\texttt{Root}()\}$;
$\mathcal{A} = \{A \mid A \in \mathcal{M}_{state}.\texttt{Root}().\texttt{OutArcs}(),$
$A.\texttt{ArcCells}()$ contains some active cell$\}$;

- `IncludeNode`(C): corresponds to Algorithm 5.4. The implementation is:

  for every $C' \in C.\texttt{Parents}()$ do
      if not `IsInState`(C') and not `SafetyTest`(C, C') then
          `IncludeNode`(C');
  $\texttt{AddToState}_{node}(C)$;
  for every $A \in C.\texttt{OutArcs}()$ such that $A.\texttt{ArcCells}()$ contains some active cell do
      $\texttt{AddToState}_{arc}(A)$;
  for every $A \in C.\texttt{InArcs}()$ do $\texttt{DelFromState}_{arc}(A)$;

  where `SafetyTest`(C, C') implements the safety test on C, C' and F_{state}.

5.8.4 Query Operations

A user queries an MC by passing through an object of class `Extractor` or of a subclass.

The public interface of `Extractor` contains primitives that permit to:

1. connect an extractor to a certain MC;
2. assign the resolution filter and the focus set; in the global case, the focus set (if present) is used as an area of interest for restricting the resolution filter;
3. update its internal state according to the current resolution filter and focus set;
4. get the simplicial complex corresponding to the solution of the query in the current state.

Class `GExtractor` adds an operation returning the complex formed by just the active simplexes of the current MC-front; such operation is useful for implementing the two-step approach to local queries (see Section 5.5.1).

The type `Parts(Cell)` is used to represent cell complexes in the output parameters of primitives. Now, we specify the primitives and the requirements for their implementation:

- `ConnectExtractor(MC)`.

  `ConnectExtractor`($\mathcal{M}$): sets the extractor on the Multi-Complex $\mathcal{M}$, resets the resolution filter and the focus set, and places the extractor in an initial state. Its implementation looks the same in any approach:

$\mathcal{M}_{state} := \mathcal{M}$;
R_{state} := function returning 0 on any cell;
$F_{state} := \mathbb{E}^d$;
`InitState`($\mathcal{M}$);

In fact, the implementation is different because of the different implementation the auxiliary operation `InitState`.

Note that the same MC may be connected to more than one extractor at a time, since all information about the current state of DAG traversal is contained in the extractor, and not in the MC itself.

- `SetFilter(Filter)`.

  `SetFilter`(R): assigns the current resolution filter R_{state} to R.

- `SetFocus(Focus)`.

  `SetFocus`(F): assigns the current focus set F_{state} to F.

- `UpdateExtractor()`.

  `UpdateExtractor()`: updates the internal state of DAG traversal to reflect the current resolution filter and focus set. The implementation of `UpdateExtractor` is different in the static and in the dynamic case.

  The implementation of `UpdateExtractor` within class `Extractor` corresponds to the basic approach in which the DAG traversal state is reset at each query. It reduces to two stages: (i) reset the DAG traversal state (to an initial sub-MC, containing just the root node); and (ii) update such sub-MC through iterative inclusion of nodes:

  ```
  InitState();
  Expansion();
  ```

  The implementation remains the same in the global and in the local case, but it relies on a different implementation of auxiliary operations.

  The implementation of `UpdateExtractor` changes in the dynamic case (class `DExtractor`) because the internal state survives to be used to compute the answer to the next call. The two stages become: (i) update the current sub-MC (inherited from the last query) through iterative inclusion of nodes; and (ii) update the current sub-MC through iterative elimination of nodes:

  ```
  Expansion();
  Contraction();
  ```

- `GetComplex`() $\longrightarrow$ `Parts(Cell)`.

  `GetComplex`(): returns the solution defined by the current state. The implementation is different in the global and in the local case.

  In global algorithms (subclass `GExtractor`) operation `GetComplex`() collects all the simplices of the arcs of set $\mathcal{A}$:

  return $\{\gamma \,|\, \gamma \in A.\texttt{ArcCells}(), A \in \mathcal{A}\}$;

  In the local algorithm (subclass `LExtractor`), it collects just the active simplices of the arcs of $\mathcal{A}$:

  return $\{\gamma \,|\, \gamma \in A.\texttt{ArcCells}(), A \in \mathcal{A}, \gamma \text{ is active}\}$;

- `GetActiveComplex`() $\longrightarrow$ `Parts(Cell)`.

  This primitive is in the interface of class `GExtractor` and its subclasses only.

  `GetActiveComplex`(): returns the combined complex formed by the *active* simplexes labelling the current MC-front:

  return $\{\gamma \,|\, \gamma \in A.\texttt{ArcCells}(), A \in \mathcal{A}, \gamma \text{ is active}\}$;

  Note that the implementation of `GetActiveComplex` looks the same as that of `GetComplex` in the local case, but sets $\mathcal{N}$ and $\mathcal{A}$ have different contents.

5.9 Summary

In this chapter, we have defined two general queries on Multi-Complexes: one returning a cell complex representing the target object according to certain resolution requirements, and another one extending interference queries with an additional parameter that permits to choose the desired resolution. In both cases, we have addressed the min-size version as well as the best-resolution version.

We have proposed algorithms for the global query (using a static approach) which find an optimal (minimum-size or best-approximation) solution, and dynamic algorithms that may find an approximated solution, but in practice work very well. For the local query, we have proposed algorithms that find approximated solutions for conservative MCs. The algorithms have been implemented, and have been compared experimentally in Section 5.7. The basic algorithms for the min-size global query is an interruptible version of the algorithm proposed in [Pup96, Pup98]; the other algorithms are new contributions.

Chapter 6

Data Structures and Algorithms

6.1 Introduction

In this chapter, we address the implementation of the Multi-Complex and of its operations into concrete data structures and algorithms. As in Section 5.8, we restrict our attention to simplicial MCs (which are the most interesting cases from an application point of view). In particular, we consider two-dimensional simplicial MCs, also called Multi-Triangulations, in $\mathbb{E}^3$. However, many of the proposed data structures and algorithms extend to higher-dimensions in a direct way, and a sketch of their extensions will be given.

The result of query algorithms is a cell complex (here, a triangle mesh), and may be reported to the user either without or with information about pairs of edge-adjacent triangles. In the latter case, it is necessary to reconstruct adjacency relations among the extracted triangles. We start this chapter with a discussion about the alternative approaches for determining adjacency relations in the output mesh, based either on adjacency primitives implemented on the MC data structure, or on other techniques.

We propose three basic approaches to the design of data structures for Multi-Complexes: an approach that explicitly represents the DAG [DFPM97, DFMP98c], an approach that stores just an ordered list of nodes, and an approach based on adjacency information. In our opinion, these *three families of data structures* capture all possible ways to encode a Multi-Complex.

Each family stores a certain subset of the information describing an MC and, thus, supports some of the kernel primitives. The set of available kernel primitives effects the implementation of primitives at the upper levels. In particular, we will see that all data structures support the basic algorithm for the global query, but not all of them support the dynamic one, or the algorithm for the local query. The DAG-based approach is the most complete, since it supports all query algorithms and the implementations are simpler than in other approaches. We present a sample data structure for each family and the corresponding implementation of

MC operations, which is used to compare the three approaches.

Next, we propose some *compressed data structures* which are especially compact and apply to MCs *having certain characteristics*. Such characteristics are due to the specific properties of the sequence of mesh modifications from which an MC is constructed. Here again, we describe sample implementations and compare them with general data structures.

Finally, we briefly address representation schemes for storing an MC on *secondary storage*. We discuss solutions for organizing an MC on disk in such a way that relevant parts of it can be loaded into memory at run-time and used by an application.

This chapter is organized as follows:

- In Section 6.2, we discuss the problem of generating adjacency information for the output mesh of query algorithms;
- In Section 6.3, we describe the three basic representation schemes for MCs.
- In Section 6.4, we describe compressed data structures suitable to represent certain classes of MCs.
- In Section 6.5, we address the problem of representing an MC on secondary storage.

In this chapter, we always consider the *enriched MC*, that we call simply MC, for short. Throughout the chapter, we will use the following notations throughout the chapter: the symbol σ is used for top-simplices (i.e., triangles), edges and vertices are denoted with e and p, respectively.

6.2 Generation of Adjacency Relations in the Output Mesh

The result of query operations is a regular cell complex, i.e., in the case of a two-dimensional simplicial Multi-Complex, a triangle mesh. In an implementation, we must specify what information is reported to the user to describe an extracted triangle mesh. In Section 2.2, we reviewed two basic types of data structures for (two-dimensional) simplicial complexes:

1. data structures storing triangle-vertex connectivity (as the *indexed data structure* in Section 2.2), which provide, for each triangle σ, just the boundary relation linking σ to its three vertices;
2. data structures storing also triangle-triangle connectivity, i.e., adjacency relations between triangles sharing a common edge (as the *enriched indexed data structure* in Section 2.2).

Both representation schemes are complete, i.e., they are sufficient to characterize a triangle mesh uniquely. Scheme 1 is obviously more compact. Scheme 2 is somehow "redundant", but it is motivated by the cost of computing adjacencies for a mesh where triangles are known just as triplets of vertices.

The decision whether to output an extracted triangle mesh with or without adjacency information depends on the successive use of such mesh within the application. For instance, adjacency information is not necessary for visualization; it is necessary when the extracted mesh must undergo analysis operations which involve traversing its connecting structure (e.g., terrain operations in GISs, such as the computation of paths, of visibility properties, etc.).

The mesh resulting from a query is made up of triangles associated with different arcs and nodes of an MC; the same triangle can be part of many output meshes, each time with different triangles adjacent to it. Thus, edge-adjacency (from now called simply *adjacency*) information for an extracted mesh has to be computed on-line at each query.

We propose three main approaches to generate adjacency information for an extracted mesh:

1. *Reconstruct adjacency relations in a post-processing step* at the end of DAG traversal. This takes $O(m \log m)$ time in the worst case, if m is the size of the output mesh.

2. *Maintain and update adjacency relations* while traversing the DAG by using adjacency primitives of the kernel.

   This approach requires that the four adjacency primitives $\texttt{CoAdj}_1$, $\texttt{CoAdj}_2$, $\texttt{CntAdj}_{1,2}$ and $\texttt{CntAdj}_{2,1}$ are supported by the data structure representing the MC in optimal time, i.e., $O(1)$. Such primitives are used to update adjacency links in the current mesh each time a node is included in, or eliminated from the current sub-MC. The total time complexity is $O(m)$ in the worst case, where m is the sum of the number of triangles present in all the nodes visited by a query algorithm. If the MC has bounded width, then the time is linear in the number of visited nodes.

3. *Incrementally reconstruct adjacency relations* during DAG traversal without using adjacency primitives of the kernel.

   This approach applies to MCs which do not support the four kernel primitives $\texttt{CoAdj}_1$, $\texttt{CoAdj}_2$, $\texttt{CntAdj}_{1,2}$ and $\texttt{CntAdj}_{2,1}$. As in Approach 2, adjacency information for the current mesh is maintained and updated at each stage. Rather than using adjacency primitives of the kernel, a process of adjacency reconstruction, similar to that applied to the whole extracted mesh in approach 1, is applied locally to the part of the current mesh affected by the inclusion / elimination of a node. The time complexity is $O(m \log M)$ in the worst case, where m is as in Approach 2, and M is the maximum number of triangles in a traversed node. If the MC has a bounded width, the total cost reduces to $O(m)$.

In the following subsections, we describe the above three methods. Note that the same three approaches, with analogous time complexities, can be used to generate $(k-1)$-adjacency in-

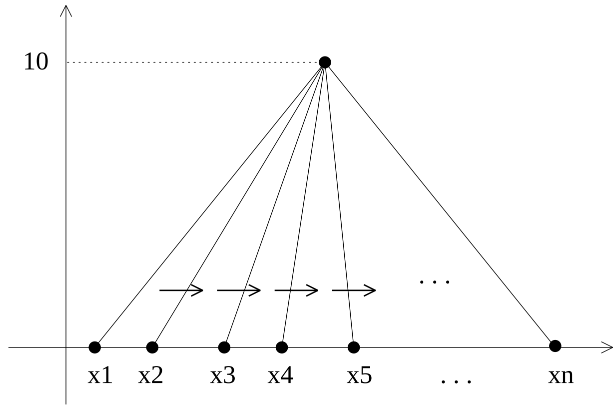


Figure 6.1: Sorting n real numbers reduced to adjacency reconstruction in a triangulation.

formation for a k-dimensional simplicial complex extracted from a k-dimensional simplicial MC: simply, the $(k-1)$-faces of a top-simplex are considered, whenever, here, we consider the edges of a triangle.

6.2.1 Adjacency Reconstruction as a Post-Processing Step

Given the triangles of a mesh Σ as triplets of vertices, the problem of determining the pairs of edge-adjacent triangles in Σ has an intrinsic complexity equal to $\Theta(m \log m)$, where m is the number of edges of Σ, since it is equivalent to a sorting problem.

Lemma 6.1 *Sorting m real numbers $x_1, \ldots, x_m$ reduces to reconstructing the adjacency relations of a mesh of triangles, with a total of $2m-1$ edges, constructed in this way: for $i = 1 \ldots m-1$, number x_i is mapped into the triangle having its vertices at the points $(\frac{x_1+x_n}{2}, 10), (x_i, 0), (x_{i+1}, 0)$ (see Figure 6.1).*

Lemma 6.2 *Adjacency reconstruction within a triangle mesh Σ reduces to sorting by considering all the pairs (e, σ), where σ is a triangle in Σ and e is an edge of σ; sorting all these pairs using the edge as a key (e.g., based on a topological order of the endpoints); and then collecting the adjacent triangles from any two consecutive pairs (e_1, σ_1), and (e_2, σ_2) in the sorted list, where $e_1 = e_2$.*

Note that the same equivalence, with similar reduction lemmas, holds for the problem of adjacency reconstruction in a k-dimensional simplicial complex Σ.

As an alternative to Lemma 6.2, we use a search structure such as a balanced binary tree to find the pairs of adjacent triangles, in the following way:

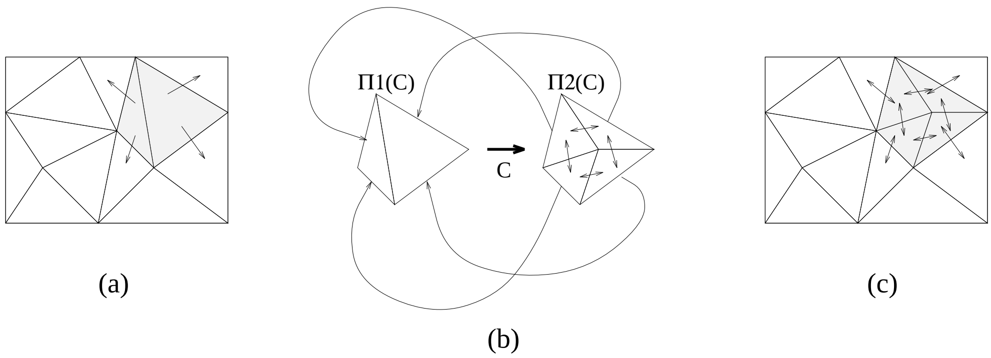


Figure 6.2: Incremental adjacency update when including a node: (a) the current mesh, (b) the modification pattern corresponding to the node C to be included, (c) the new mesh. In (a) and (b) the arrows show the existing connectivity information exploited for the update; in (c) the arrows show the adjacency links established in the new mesh.

- Let the search structure be initially empty.
- For every triangle σ in the mesh, for every edge e of σ:
  - if a pair (e, σ') is present in the search structure, then delete such pair from the search structure, and set the adjacency link between σ and σ' along edge e in the mesh,
  - otherwise, insert pair (e, σ) into the search structure.

For each edge of the mesh, at most two searches, an insertion and a deletion are performed on the search structure, each taking a logarithmic time in the worst case. The search structure contains m pairs in the worst case (i.e., if all insertions are performed first, followed by all deletions). Thus, the complexity is $O(m \log m)$ in the worst case, where m is the number of edges in the mesh.

6.2.2 Incremental Adjacency Update Using Adjacency Primitives

Here, we assume that kernel primitives $\texttt{CoAdj}_1$, $\texttt{CoAdj}_2$, $\texttt{CntAdj}_{1,2}$ and $\texttt{CntAdj}_{2,1}$ are supported in constant time by the data structure encoding the MC.

When a node C is included in the current sub-MC, the connectivity structure of the current mesh is updated by using primitives $\texttt{CoAdj}_2$ and $\texttt{CntAdj}_{2,1}$ in the following way (see Figure 6.2):

- For every triangle $\sigma \in C.\texttt{Cells}_2()$, for every edge e of σ:
  - if $\sigma.\texttt{CoAdj}_2(e)$ exists (i.e., e is an internal edge of $\Pi_2(C)$), then, letting $\sigma'' = \sigma.\texttt{CoAdj}_2(e)$, link σ and σ'' along e in the current mesh;

– otherwise, if σ.$\texttt{CntAdj}_{2,1}(e)$ exists (i.e., e is a boundary edge for both $\Pi_2(C)$ and $\Pi_1(C)$), then, denoted σ.$\texttt{CntAdj}_{2,1}(e)$ with σ', pick the triangle σ'', adjacent to σ' along e in the current mesh, and link σ and σ'' along e in the current mesh.

Provided that primitives $\texttt{CoAdj}_2$ and $\texttt{CntAdj}_{2,1}$ are implemented in $O(1)$ time, the cost for adjacency update is linear in the number of triangles of $\Pi_2(C)$.

Symmetrically, when, in a dynamic algorithm, a node C is eliminated from the current sub-MC, the connectivity of the current mesh is updated by using primitives $\texttt{CoAdj}_1$ and $\texttt{CntAdj}_{1,2}$. The procedure works as described above, simply replacing primitive $\texttt{Cells}_2$ with $\texttt{Cells}_1$, $\texttt{CoAdj}_2$ with $\texttt{CoAdj}_1$, and σ.$\texttt{CntAdj}_{2,1}$ with σ.$\texttt{CntAdj}_{1,2}$.

Provided that primitives $\texttt{CoAdj}_1$ and $\texttt{CntAdj}_{1,2}$ are implemented in $O(1)$ time, the cost for adjacency update is linear in the number of triangles of $\Pi_1(C)$, i.e., in the number of triangles of $\Pi_2(C)$, since C represents a refinement pattern.

The total cost for adjacency management in a query is $O(m)$, where m is the sum of the sizes of $\Pi_2(C)$ over all processed nodes C. If the MC has bounded width, then adjacency update (both when including and when eliminating an MC-node) is done in $O(1)$ time, and the overall cost in a query becomes linear in the number of traversed nodes.

6.2.3 Incremental Adjacency Reconstruction

This approach can be used with MC data structures which do not support adjacency primitives of the kernel.

When a node C is included into, or eliminated from the current sub-MC, adjacency relations of the current mesh are reconstructed *locally* to be part of the mesh that has changed its connectivity (namely, the triangles of node C and their neighbors in the current mesh), by using an approach similar to that used in Section 6.2.1.

When a node C is included in the sub-MC, we operate as follows:

- Empty the search structure.
- For every triangle $\sigma \in C$.$\texttt{Cells}_1()$, for every edge e of σ:
  - let σ' be the triangle adjacent to σ along e in the current mesh;
  - if σ'.$\texttt{CellDest}() \neq C$ (i.e., e is a boundary edge for $\Pi_1(C)$), then insert the pair (e, σ') into the search structure;
  - otherwise, e is an internal edge of $\Pi_1(C)$, and nothing has to be done.
- For every triangle $\sigma \in C$.$\texttt{Cells}_2()$, for every edge e of σ:

- if a pair (e, σ') is present in the search structure, then remove it, and link σ and σ' along e in the current mesh;
- otherwise, insert the pair (e, σ) into the search structure.

Symmetrically, when a node C is eliminated from the current sub-MC, the same operations are performed, where simply we replace primitive $\texttt{Cells}_1$ with $\texttt{Cells}_2$ and `CellDest` with `CellSource`.

In both cases, for each edge, a pair containing the edge is inserted, and possibly removed from the search structure, and at most two searches are performed. The cost is proportional to the number m_i of triangles of $\Pi_1(C) \cup \Pi_2(C)$ multiplied by the access time to the search structure, thus, $(m_i \log m_i)$ time. If the MC has bounded width, then, for every MC-node C_i, $m_i = O(1)$ and the time necessary to reconstruct adjacencies when including or eliminating C_i becomes constant.

6.3 General-Purpose Data Structures

A two-dimensional simplicial Multi-Complex is fully characterized by its triangles, its nodes, its arcs, and by the information that kernel primitives associate with such entities.

The whole set of all of this information is redundant, since some information is logically derived from other information (some relevant cases have been presented in Section 4.6). Thus, it is possible to select subsets of information, which are sufficient to describe an MC uniquely.

In this section, we examine different approaches to storing an MC, thus proposing *families* of data structures. Each approach stores a certain subset of the information characterizing an MC explicitly and uses the properties of kernel operations for deriving the remaining information, if they are needed.

The data structures considered here do not make any assumption about the modification patterns contained in an MC. In Section 6.4, we will present compressed versions of the approaches proposed here, suitable for MCs where modification patterns obey to predefined rules.

Kernel primitives *must be efficient*, since they are used several times within algorithms at the upper levels. We say that a kernel primitive *is supported* by a data structure if it is supported in optimal time, i.e., linear in the output size. If a kernel primitive can only be supported with a sub-optimal time complexity, then we consider the primitive as *not available*. Unavailable primitives have effect on the operations at the upper levels that can be supported by a data structure. If a primitive, that is not available, is absolutely necessary to implement a certain query algorithm, then such algorithm is not available on that data structure.

For each family of data structures, we provide:

- a *high level description of the approach*, in terms of: the stored information, the MC properties used to retrieve missing information;
- a *sample implementation* into a concrete data structure, which is proposed as a representative of the family, and an evaluation of its space complexity;
- an *implementation of the supported query operations* on the sample data structure, and an evaluation of their time complexity, as well as an evaluation of the space complexity of the structures used to describe the internal state of the process.

We will consider the basic and the dynamic algorithm for the global query, and the algorithm for the local query. For the sake of simplicity and brevity, only the min-size variant of query algorithms is considered (i.e., Algorithms 5.2, 5.6, and 5.5). Such algorithms will be called the *basic*, the *dynamic*, and the *local* one, respectively. Moreover, we consider the *heuristic* variant of the local algorithm, where a parent C' of an MC-node C is considered as relevant if and only if the arc (C', C) does not contain any active triangle in its label (see Section 5.5.4). For convenience, we will use the term *static* to denote the two non-dynamic algorithms (i.e., the basic and the local one). As in Section 5.8.3, we assume to have a focus set in the global case as well, which is used as an area of interest to restrict the current resolution filter.

The sample implementations provided for the three families of data structures are all based on storing information as compactly as possible by using static structures such as arrays. In Section 6.3.5, we compare them in terms of their space complexity, and of the time and space complexity of the query algorithms implemented on them.

For each family of data structures, we will describe up to three variants, each supporting a different subset of the adjacency primitives of the kernel. More precisely, we consider:

- A basic variant supporting none of them, called the variant with *zero-adjacency*. With this variant, if adjacency information for the extracted mesh is required, then we must reconstruct it either at each modification of the current sub-MC, or at the end of the process (as explained in Sections 6.2.3 and 6.2.1, respectively).
- A variant supporting just $\texttt{CoAdj}_2$ and $\texttt{CntAdj}_{2,1}$ (i.e, primitive sufficient for updating adjacencies when including a node, but not when excluding it), called the variant with *half-adjacency*; this variant allows adjacency update (as explained in Section 6.2.2) in all query algorithms with the exception of the dynamic one. Thus, it is convenient if we do not use the dynamic algorithm.
- A variant supporting all four primitives (thus, allowing adjacency update in the dynamic algorithm as well), called the variant with *full-adjacency*.

We introduce some notations that are used in the following subsections when evaluating the storage requirements of data structures and the time complexities of algorithms:

- n, a, v, t denote the number of nodes, arcs, vertices and triangles of an MC, respectively;
- `int` and `real` denote the (machine-dependent) cost of storing either an integer and a real number, respectively;
- n_{sub}, a_{sub}, t_{sub} denote the number of nodes, arcs, and triangles of the sub-MC defining the result of the basic query algorithm (i.e., the sub-MC visited by such algorithm);
- in the dynamic algorithm, n_{dyn}, a_{dyn}, t_{dyn} denote the number of nodes, arcs, and triangles in the part of DAG formed by the input MC-front, the output MC-front, and all the nodes and arcs lying between them, (i.e., the part of the DAG visited by the dynamic algorithm).
- n_{loc} denotes the number of nodes of the pseudo-sub-MC defining the result of the local algorithm; a_{loc}, t_{loc} denote the number of arcs and of triangles labelling the arcs incident in the nodes of such pseudo-sub-MT; these are the same as the nodes, arcs, and triangles visited by the local algorithm.
- $\mathcal{T}_{cond}$ denotes the time complexity of computing the value of the resolution filter R on a triangle and of testing whether that triangle is active with respect to focus set F; we assume to implement query algorithms by referring to "active" triangles; the modified version taking into account of "loosely-active" triangles is straight-forward.

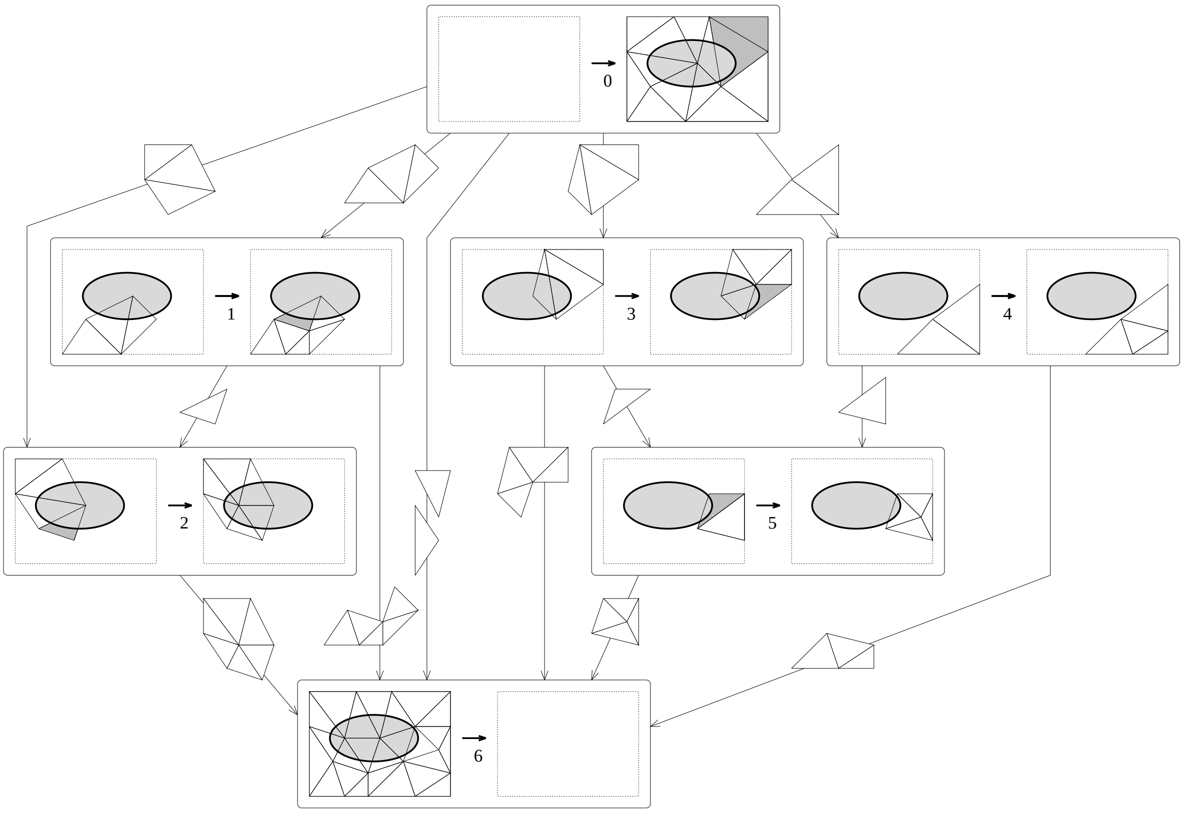
Figure 6.3 illustrates the relation between parameters n_{sub}, n_{dyn}, and n_{loc}. Note that $n_{loc} \leq n_{sub}$, while a bound cannot be given for n_{dyn} with respect to n_{sub}.

Note that $a \leq t$, since every arc is labelled by at least one triangle, and each triangle appears in exactly one arc, and $n \leq t/2$, since each node C (with the possible exception of the root) contains at least two triangles in $\Pi_2(C)$ (due to the fact that it represents a refinement pattern, and $\Pi_1(C)$ is not empty), and each triangle is in $\Pi_2(C)$ for exactly one node C. In any case, $n_{sub}, n_{dyn}, n_{loc} \leq n$, $a_{sub}, a_{dyn}, a_{loc} \leq a$, and $t_{sub}, t_{dyn}, t_{loc} \leq t$.

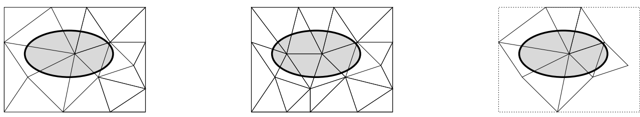
6.3.1 An Explicit Data Structure

The data structure described in this section is called *explicit* since it directly represents the enriched MC. It stores information corresponding to each kernel primitive explicitly, with the exception of a few primitives which are trivially redundant and can be derived in a straigth-forward way by exploiting standard properties of the DAG (discussed in Section 4.6):

- the parents and the children of a node C can be retrieved as the sources of the incoming and outgoing arcs, respectively, of C;
- the source and the destination node of a triangle σ can be found as the source and destination, respectively, of the arc containing σ;



(a)



(b) (c) (d)

Figure 6.3: Different sets of nodes examined by the three query algorithms considered. We assume to run the two global query algorithms with a resolution restricted to focus set of the local algorithms. (a) The enriched MC; the focus set is the grey oval, the resolution filter is greater than zero on shaded triangles. The final sub-MC for the basic algorithm contains nodes 0,3,4,5 and yields mesh (b). The initial sub-MC for the dynamic algorithm contains nodes 0,1,2,3,4,5 and is the same as the final sub-MC, the corresponding triangle mesh is (c). The final pseudo-sub-MC for the local algorithm contains nodes 0,3,4 and yields mesh (d). Thus, $n_{sub} = 4$, $n_{dyn} = 0$, $n_{loc} = 3$.

- the triangles of $\Pi_1(C)$ and those of $\Pi_2(C)$ for a node C can be found in linear time as the union of all the triangles associated with the incoming and outcoming arcs, respectively, of C.

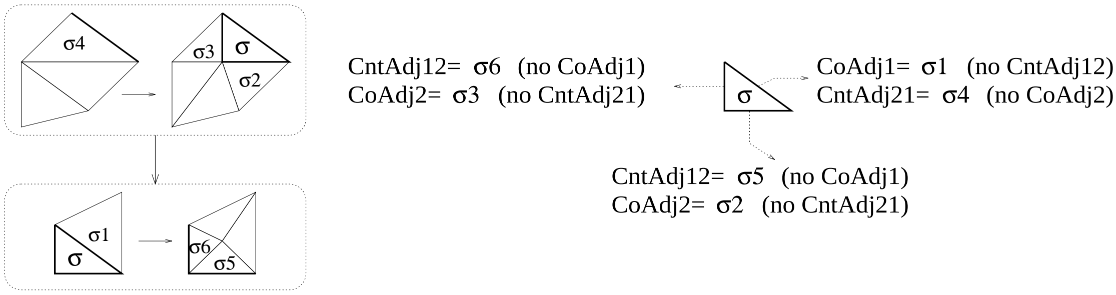


Figure 6.4: The values of the four adjacency primitives for a triangle σ of an MC.

6.3.1.1 High-Level Description of the Explicit Structure

The explicit data structure stores the vertices, triangles, nodes and arcs of an MC, and the following information:

- For each vertex, its coordinates.
- For each triangle: the links to its three vertices (primitive `CellVertices`), its approximation error (primitive `CellError`), a link to the arc containing it (primitive `CellArc`), and *adjacency information* depending on the variant (zero-, half-, or full-adjacency), as described below.
- For each node: links to its incoming arcs (primitive `InArcs`), and links to its outgoing arcs (primitive `OutArcs`).
- For each arc: a link to its source (primitive `Source`), a link to its destination (primitive `Dest`), links to the triangles of its label (primitive `ArcCells`).

Depending on the variant of the explicit structure, a different amount of adjacency information is stored for each triangle σ:

- *Full-adjacency*: σ stores links corresponding to all adjacency primitives $\texttt{CoAdj}_2$, $\texttt{CoAdj}_1$, $\texttt{CntAdj}_{2,1}$, $\texttt{CntAdj}_{1,2}$ (see Figure 6.4). In principle, this requires four links for each edge of σ, i.e., twelve links per triangle.

  We recall from Section 4.6 that, for every triangle σ and for every edge e of σ, at most one of $\sigma.\texttt{CoAdj}_2(e)$ and $\sigma.\texttt{CntAdj}_{2,1}(e)$ exists, and at most one of $\sigma.\texttt{CoAdj1}(e)$ and $\sigma.\texttt{CntAdj}_{1,2}(e)$ exists. In addition, $\sigma.\texttt{CoAdj}_2(e)$ has the same source as σ, while $\sigma.\texttt{CntAdj}_{2,1}(e)$ has a different source; $\sigma.\texttt{CoAdj}_1(e)$ has the same destination as σ, while $\sigma.\texttt{CntAdj}_{1,2}(e)$ has a different destination.

  Based on these properties, we store just one link to represent both $\sigma.\texttt{CoAdj}_2(e)$ and $\sigma.\texttt{CntAdj}_{2,1}(e)$ for each edge e of σ. If σ' is the triangle found at this link, then the value of σ'.CellSource() discriminates whether σ' is σ.$\mathtt{CoAdj}_2(e)$ or σ.$\mathtt{CntAdj}_{2,1}(e)$. In a similar way, just one link is sufficient to represent both σ.$\mathtt{CoAdj}_1(e)$ and σ.$\mathtt{CntAdj}_{1,2}(e)$ at any edge e of σ: the value of σ.CellDest() discriminates which is the case. Thus, a total of *six links* are stored for σ.

- *Half-adjacency*: for each edge e of σ, we provide just the links to σ.$\mathtt{CoAdj}_2(e)$ and to σ.$\mathtt{CntAdj}_{2,1}(e)$, which are stored by using the same physical link (for a total of *three links*); the value of the source node is used as a discriminant, according to the same convention adopted for full-adjacency.
- *Zero-adjacency*: no adjacency information is stored.

6.3.1.2 Implementation of Kernel Primitives on the Explicit Structure

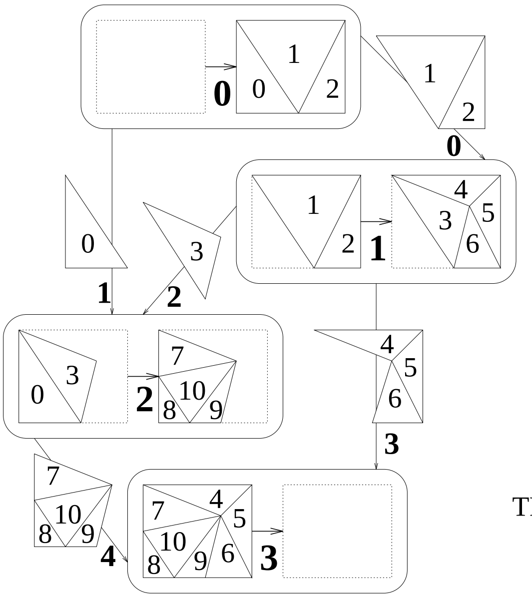
Primitives for querying the DAG structure and the mesh updates are either directly represented in the data structure (Root, Source, Dest, CellArc, InArcs, OutArcs), or obtained as a simple combination of the information encoded into the structure:

- σ.CellSource() = σ.CellArc().Source();
- σ.CellDest() = σ.CellArc().Dest();
- C.Parents() $= \cup \{a$.Source() $\mid a \in C$.InArcs()$\}$,
- C.Children() $= \cup \{a$.Dest() $\mid a \in C$.OutArcs()$\}$,
- C.$\mathtt{Cells}_1$() $= \cup \{a$.ArcCells() $\mid a \in C$.InArcs()$\}$,
- C.$\mathtt{Cells}_2$() $= \cup \{a$.ArcCells() $\mid a \in C$.OutArcs()$\}$.

Thus, all such operations are implemented in optimal time, i.e., $O(1)$ time for Root, Source, Dest, CellSource, CellDest, CellArc, and in time linear in the output size for Parents, Children, InArcs, OutArcs, $\mathtt{Cells}_1$, $\mathtt{Cells}_2$.

In the variant with full-adjacency, primitives for querying adjacencies are directly implemented.

In the variant with half-adjacency, only $\mathtt{CoAdj}_2$ and $\mathtt{CntAdj}_{2,1}$ are directly represented. Primitives $\mathtt{CoAdj}_1$ and $\mathtt{CntAdj}_{1,2}$ are not supported, since they can only be retrieved with a sub-optimal time complexity. In order to retrieve σ.$\mathtt{CoAdj}_1(e)$, we must test a triangle σ against all the triangles in $\Pi_1(C)$, where C is the node such that $\sigma \in \Pi_1(C)$ (i.e., against the triangles in σ.CellDest().$\mathtt{Cells}_1$()) to check if they share edge e; this requires a time linear in the number of edges of mesh $\Pi_1(C)$ which is not optimal. In order to retrieve σ.$\mathtt{CntAdj}_{1,2}(e)$, we must test σ against all the triangles in $\Pi_2(C)$ (retrieved as σ.CellDest().$\mathtt{Cells}_2$()), with a time linear in the number of edges of mesh $\Pi_2(C)$ (not optimal).



NODE ARRAY

	0	1	2	3
first out arc	0	2	4	-
first in arc	-	0	1	3

ARC ARRAY

	0	1	2	3	4
source	0	0	1	1	2
destination	1	2	2	3	3
first triangle	1	0	3	4	7
next arc	-	2	-	4	-

TRIANGLE ARRAY

	0	1	2	3	4	5	6	7	8	9	10
arc	1	0	0	2	3	3	3	4	4	4	4

Figure 6.5: An MC and its representation through the explicit data structure. The figure shows just the part of the structure describing the relations between nodes, arcs, and triangles: the vertex array is not shown, as well as the fields of the triangle array corresponding to the three vertices and to the approximation error.

Finally, the variant with zero-adjacency does not support any of the four adjacency primitives; in fact, they can only be retrieved in sub-optimal time.

6.3.1.3 A Sample Implementation of the Explicit Structure

We store the vertices, triangles, arcs, and nodes of an MT, in four arrays:

- Each entry in the *vertex* array represents a vertex v, and stores the *three cartesian coordinates* of v.
- Each entry in the *triangle* array represents a triangle σ, and stores: the indexes of the *three vertices* of σ; its *approximation error*; and the index of the *arc* having σ in its label. In the case of full-adjacency: the indexes of *six adjacent triangles*; in the case of half-adjacency: the indexes of *three adjacent triangles*. Triangles labelling the same arc are stored at consecutive entries in the triangle array.
- Each entry in the *arc* array represents an arc A, and stores: the indexes of the *source and destination* node of A; the index of the *first triangle* associated with A (all the triangles of A are found by scanning the triangle array from the position of the first triangle as long as the arc referenced in the current triangle is A); and the index of the *next arc* having

the same destination node. Arcs having the same source node are stored at consecutive positions in the arc array.

- Each entry in the *node* array represents a node C, and stores: the index of the *first arc outgoing* from C (all the out-arcs of C are found by scanning the arc array starting at the first arc while the current arc has C as its source); the index of the *first arc entering* C (all the in-arcs of C are found by starting from the first one and following the "next" link). Conventionally, the root node and the drain node are stored in the first, and the last position of the node array, respectively.

An example is shown in Figure 6.5. The total cost of this implementation of the explicit data structure is:

- $v(3\texttt{real}) + t(10\texttt{int} + 1\texttt{real}) + a(4\texttt{int}) + n(2\texttt{int})$ for full-adjacency,
- $v(3\texttt{real}) + t(7\texttt{int} + 1\texttt{real}) + a(4\texttt{int}) + n(2\texttt{int})$ for half-adjacency, and
- $v(3\texttt{real}) + t(4\texttt{int} + 1\texttt{real}) + a(4\texttt{int}) + n(2\texttt{int})$ for zero-adjacency.

6.3.1.4 Representation of the DAG Traversal State for the Explicit Structure

In Section 5.8.3, we modeled the internal state as a set $\mathcal{N}$ containing the nodes of the current (pseudo-)sub-MC, and a set $\mathcal{A}$ containing the arcs for the corresponding (pseudo-)MC-front.

We store set $\mathcal{A}$ into a linked list containing the indexes of the arcs in $\mathcal{A}$. Insertions in such list are performed at the rear; deletions may occur in arbitrary positions. In order to allow deletions in constant time, we use another array to mark virtually deleted arcs without actually removing them from the list.

We represent set $\mathcal{N}$ as an array with one entry for each MC-node, where each entry stores a special value discriminating whether the corresponding node is in $\mathcal{N}$ or not.

For the static query algorithms (i.e., the basic and the local one), the value stored for a node $C \in \mathcal{N}$ is a progressive count, incremented at each call of the algorithm. The value stored for a node $C \notin \mathcal{N}$ is either zero, or an old (i.e., smaller) value of the counter. This avoids scanning the array from end to end when set $\mathcal{N}$ must be emptied (i.e., at each new query).

In the dynamic case, the content of set $\mathcal{N}$ survives for the next call of `UpdateExtractor`, thus the problem of resetting does not exist. The value stored for a node $C \in \mathcal{N}$ is an integer equal to the number of children of C which are in $\mathcal{M}$, plus one; this permits to keep track of the nodes that are/become/cease to be leaves. The value stored for a node $C \notin \mathcal{N}$ is zero.

For supporting the contraction phase of the dynamic algorithm, we use a linked list containing the indexes of the nodes in $\mathcal{M}$ which are leaves. Deletions from such list occur at the head, insertions occur at the rear.

Now, we summarize the data structures representing the internal state of query algorithms and evaluate their spatial requirements. We do not consider information items with a constant size (independent of the specific MC represented), such as single counters, pointers to the head and the rear of a list, array sizes, etc. For *all query algorithms*, we maintain:

- An array with as many entries as the nodes of the MC, representing the set $\mathcal{N}$ of nodes belonging to the current (pseudo-)sub-MC. The entry of a node C stores an integer number representing:
  - for the static algorithms:
    - a number equal to the current extraction counter, if $C \in \mathcal{N}$,
    - any different (in practice, smaller) number, if $C \notin \mathcal{N}$;
  - for the dynamic algorithm:
    - $k+1$, if $C \in \mathcal{N}$, where $k = \text{size}(C.\texttt{Children}() \cap \mathcal{N})$,
    - 0, if $C \notin \mathcal{N}$.

  This array requires a space equal to $n(1\texttt{int})$.

- A linked list containing the arc of the current (pseudo-)MC-front $\mathcal{A}$. Each item of the list contains the index of an arc, and a pointer to the next item. The maximum number of arcs in this list (including virtually deleted ones) is at most a_{sub}, a_{dyn}, and a_{loc}, for the basic, dynamic, and local algorithm, respectively; all such quantities are bounded from above by a. Thus the space required is $\leq a(2\texttt{int})$.

- An array of bits, with as many entries as the arcs of the MC, used to mark arcs that are virtually deleted in the above list. This array requires a bits.

The internal state of the *dynamic algorithm* contains in addition a linked list of leaf nodes, where each item contains the index of a node, and a pointer to the next item. The maximum number of nodes in the leaf list is at most and n_{dyn}. Since $n_{dyn} \leq n$, the space required is $\leq n(2\texttt{int})$.

In *all algorithms*, if we want to generate the connecting structure for the output mesh, and we choose to do that by maintaining it either through incrementally update (with full-adjacency, or half-adjacency in the static case) or incremental local reconstruction (in other cases), then we need, in addition, a data structure for storing the current triangle mesh and its adjacency relations. Such structure must store, for each triangle of the current mesh, three pointers to adjacent triangles. By bounding the number of current triangles by t, we obtain that the space required is $\leq t(3\texttt{int})$. Otherwise, the triangles of the output mesh can be simply retrieved from the arcs of $\mathcal{A}$ at the end of DAG traversal.

In summary the space required for representing the internal state is at most:

- $n(1\texttt{int}) + a(1 + 2\texttt{int})$ for the basic and local algorithms, plus $t(3\texttt{int})$ if we keep the connecting structure of the current mesh;

- $n(3\texttt{int}) + a(1 + 2\texttt{int})$ for the dynamic algorithm, plus $t(3\texttt{int})$ if we keep the connecting structure of the current mesh.

6.3.1.5 Implementation of DAG Traversal Primitives for the Explicit Structure

Now, we provide the implementation of DAG traversal operations and evaluate their computational cost. For the sake of simplicity, we assume that we *do not* maintain the connectivity structure of the current mesh.

In the static algorithms, operation `InitState` first increments the extraction counter, empties the arc list and resets the marks for its elements. Then, it initializes the array representing set $\mathcal{N}$ and the list representing $\mathcal{A}$ according to the specification given in Section 5.8.3 (which is different in the basic and in the local case), and places a cursor before the first item in the list representing $\mathcal{A}$.

Emptying list $\mathcal{A}$ and resetting the associated bit array requires a time linear in the number of arcs in $\mathcal{A}$, but we charge this cost to the previous call of the query algorithm. Except for that, operation `InitState` has a cost linear in the number of triangles of $\Pi_2(C_0)$, where C_0 is the root (multiplied by $\mathcal{T}_{cond}$ in the local case).

In the dynamic algorithm, the implementation of `InitState` is different because we must scan the array representing $\mathcal{N}$ and fill it with zeroes, with a cost in $O(n)$. Note that in the dynamic case operation `InitState` is called just once when connecting the extractor to an MC, and it is not used within queries.

Operation `IsInState` reduces to reading the array entry of the given node and comparing it with the extraction counter in the static algorithms, and with zero in the dynamic one; the cost is $O(1)$.

In the static algorithms, operations $\texttt{AddToState}_{node}$ and $\texttt{AddToState}_{arc}$ reduce to insertion at the rear of a linked list; operations $\texttt{DelFromState}_{node}$ and $\texttt{DelFromState}_{arc}$ reduce to setting a flag in an array. Thus, they require $O(1)$ time.

In the dynamic algorithm, $\texttt{AddToState}_{node}$ and $\texttt{DelFromState}_{node}$, in addition, update the counters in array $\mathcal{N}$ for the parents and children of the node that is being inserted into, or deleted from, $\mathcal{N}$; when, in performing $\texttt{DelFromState}_{node}$ on one of its children, the counter of a node changes from 2 to 1 (i.e., the node becomes a leaf), then it is inserted in the leaf list. Thus, the cost of $\texttt{AddToState}_{node}$ and $\texttt{DelFromState}_{node}$, in the dynamic case, is linear in the total number of arcs entering and leaving the given node.

The implementation of `ContinueExpansion` exploits the property that the triangles of the current mesh are those associated with the arcs of set $\mathcal{A}$. Thus, `ContinueExpansion` scans the list representing $\mathcal{A}$ until it finds either an arc A containing some active and not feasible triangle, or the end of the list; in the first case, it returns true, and false otherwise. In order to avoid scanning $\mathcal{A}$ from the beginning, we keep a cursor on list $\mathcal{A}$ in such a way that the arcs lying before the cursor have been already checked, and they do not contain any active and not feasible triangle. Operation `ContinueExpansion` scans the list starting from the current position of the cursor, and advances the cursor progressively.

In the dynamic algorithm, at the end of the expansion phase we initialize the leaf list. This is done within operation `ContinueExpansion` in case of a negative answer. The leaf list is initialized by collecting all the sources of arcs in $\mathcal{A}$, whose value stored in array $\mathcal{N}$ is equal to one (i.e., their children counter is zero). We recall that nodes that become leaves afterwards will be appended to the leaf list within primitive $\mathtt{DelFromState}_{node}$, as explained before.

Operation `FindNodeToInclude` reads the arc A lying at the current cursor position on list $\mathcal{A}$, advances the cursor after A, and returns $A.\mathtt{Dest}()$. Since `FindNodeToInclude` is called just after `ContinueExpansion` has returned a positive answer, this implementation is correct; the cost is $O(1)$.

Operation `ContinueContraction` scans the leaf list in the following way; its examines the first element C: if all triangles of $C.\mathtt{Cells}_1()$ are either not active or feasible, then it returns the value true; otherwise, it deletes C from the list and examines the next element; if the list becomes empty, then it returns false. After `ContinueContraction` has returned true, operation `FindNodeToEliminate` reads the first node of the leaf list, deletes it from the list, and returns it.

The implementation of operation `IncludeNode` and `EliminateNode` have been completely specified in Section 5.8.4 based on kernel operations and on other DAG traversal operations; thus, they do not need to be implemented here. Operations `IncludeNode`, except for the recursive calls, and `EliminateNode` (in the dynamic case), have a cost linear in the number of arcs incident in the argument node C, for the basic and dynamic algorithms. For the local algorithm, operation `IncludeNode` tests all the triangles of an arc against the focus set (see the specification in Section 5.5.2): this leads to a complexity linear in the size of $\Pi_1(C) \cup \Pi_2(C)$ multiplied by $\mathcal{T}_{cond}$, for a node C.

Operations `Expansion` and `Contraction` have been completely specified in Section 5.8.3 in terms of other primitives. Here, we evaluate their computational complexity.

All the calls to `ContinueExpansion` within `Expansion` have a total cost linear in $\mathcal{T}_{cond}(\text{size}(\cup\{\mathtt{ArcCells}(a) \mid a \text{ scanned arc in the list }\}))$, i.e., $O(\mathcal{T}_{cond} t_{sub})$ in the basic algorithm, $O(\mathcal{T}_{cond} t_{dyn})$ in the dynamic algorithm, and $O(\mathcal{T}_{cond} t_{loc})$ in the local one. For the dynamic algorithm, all the calls to `ContinueContraction` within `Contraction` have a total cost linear in $\mathcal{T}_{cond}(\text{size}(\cup\{C.\mathtt{Cells}_1() \mid C \text{ scanned leaf in the list }\}))$, i.e., in $O(\mathcal{T}_{cond} t_{dyn})$.

In the static algorithms, the total cost of all calls to `IncludeNode` within `Expansion` is in $O(\mathcal{T}_{cond} t_{sub})$ and in $O(\mathcal{T}_{cond} t_{loc})$, respectively. In the dynamic case, the total cost of all calls to `EliminateNode` within `Contraction` and to `IncludeNode` within `Expansion` is $O(\mathcal{T}_{cond} t_{dyn})$.

6.3.1.6 Implementation of Query Algorithms for the Explicit Structure

Operation `UpdateExtractor` has been completely specified in Section 5.8.3 in terms of lower-level operations, and thus is not implemented here. From the analysis of the costs of its subcomponents, given before, it follows that the cost of `UpdateExtractor` is:

- for the two global algorithms: linear in the number of triangles contained in the (pseudo-) sub-MC defining the result, times the cost of evaluating the resolution filter and the intersection test with the focus, i.e., $O(\mathcal{T}_{cond}t_{sub})$ for the basic algorithm, and $O(\mathcal{T}_{cond}t_{loc})$ for the local one. If the MC has linear growth, then the computational time is linear in the output size, multiplied by $\mathcal{T}_{cond}$ (i.e., optimal when $\mathcal{T}_{cond} = O(1)$).
- for the dynamic algorithm: linear in the number of triangles contained in the difference between the sub-MCs defining the sizes of input and of the output mesh, times the cost of evaluating resolution filter, i.e., $O(\mathcal{T}_{cond}t_{dyn})$. If the MC has linear growth, then the computational time is linear in the sum of the sizes of the input and output mesh, multiplied by $\mathcal{T}_{cond}$ (i.e., optimal when $\mathcal{T}_{cond} = O(1)$).

Operation `GetComplex` reduces to scanning the list representing $\mathcal{A}$ and collecting all their associated triangles (in algorithms for the global query), or just the strongly active ones (in the local case). While scanning the list, virtually deleted arcs must be skipped. In the dynamic case, we physically remove such arcs in order to "clean" set $\mathcal{A}$ for the next call to the query operation. In the static algorithms, the whole list will be freed at the beginning of the next query. The cost is in $O(t_x)$ for global algorithms and in $O(t_x\mathcal{T}_{cond})$ for the local algorithm, where $x = sub, dyn, loc$ depending on the algorithm (basic, dynamic or local, respectively).

6.3.1.7 Remarks

We have shown the implementation of the query algorithms in their min-size variants. The best-resolution variants can be implemented in a similar way; the main changes concern the implementation of set $\mathcal{A}$ (representing the current (pseudo-)MC-front) and, in the dynamic case, that of the set of leaves. Such sets have been implemented as linked lists with a FIFO policy, and they must be changed into priority queues keyed with the value of R on the active triangles of each arc or node, respectively. This implies that insertions and deletions now take a logarithmic time in the size of the set rather than a constant time, thus adding a logarithmic factor to the time complexity of the whole algorithm.

6.3.2 A Sequential Data Structure

In this section, we describe a data structure for representing MCs that we call *sequential* since it does not encode the DAG: it replaces the DAG with a sequence of its nodes, sorted in a total order compatible with the partial order represented in the DAG. It does not store arcs, just nodes, triangles, and vertices.

Only a subset of the partial order information present in an MC is stored. The links of direct dependency encoded in an MC can be viewed in two orientations:

- the forward orientation, corresponding to primitives `CellDest` and `Children` for triangles and nodes, respectively,
- the backward orientation, corresponding to primitives `CellSource` and `Parents` for triangles and nodes, respectively.

The sequential data structure provides just primitives describing the *backward orientation* of dependency. Such information is sufficient to support the two static query algorithms, but *not the dynamic one*. We will see that also the local algorithm is not supported efficiently.

On the sequential data structure, static query algorithms (i.e., the basic and the local one) scan the list of nodes sequentially, skipping those nodes which do not need to be included, and including those which need. Thus, they examine all the nodes in the sorted sequence from the beginning up to the position of the last node belonging to the final (pseudo-)sub-MC. This is in contrast with explicit data structure, where static algorithms examine *only* the nodes of the final (pseudo-)sub-MC. Thus, the sequential data structure can be less efficient in cases when the required resolution is highly variable in space and (for the local algorithm) when the focus set interferes with a small number of nodes (e.g., a point focus). In such cases, in fact, the algorithm may traverse a large number of nodes which do not belong to the final (pseudo-)sub-MC, in order to reach the few ones that belong to it.

The local algorithm does not make much sense on the sequential structure. On the DAG, the advantage of the local over the integrated two-step approach is that of pruning some branches of the DAG during traversal. In the sequential structure, the list must be entirely scanned up to the last node of the solution in any case, thus the gain of a local approach is almost null.

Now, we explain why a dynamic query algorithm cannot be supported. The expansion stage of static query algorithms stops scanning the list as soon as all the active triangles in the current mesh are feasible; without such stopping criterion, they would have to traverse the list up to the end. With the dynamic algorithm, the problem is in the contraction phase. The natural idea is traversing the sorted list of nodes backwards, and, during the traversal, eliminating a leaf whenever this does not insert unfeasible active triangles in the current mesh. Unfortunately, we do not have any criterion to stop such backward traversal before reaching the beginning of the list because the data structure cannot tell us which nodes are leaves. Therefore, there is no gain in applying a dynamic approach implemented in this way, with respect to a static one.

6.3.2.1 High-Level Description of the Sequential Structure

The sequential data structure stores the vertices, triangles, and nodes of an MC, and the following information:

- For each vertex, its coordinates.
- For each triangle: links to its three vertices (primitive `CellVertices`), its approximation error (primitive `CellError`), link to its source node (primitive `CellSource`), and *adjacency information* (depending on the variant considered, see below).
- For each node: the next node in the sequence, links to the triangles composing the two meshes of the modification pattern represented by it (primitives $\texttt{Cells}_1$ and $\texttt{Cells}_2$).

We consider two variants of the sequential data structure which store half-, or zero-adjacency, respectively. A sequential data structure with full-adjacency is wasteful since the dynamic algorithm is not supported by this data structure.

In the case of half-adjacency, we store, for each triangle σ, three links to adjacent triangles, one for each edge of σ. According to the same convention used for the explicit structure, the triangle found at each of such links corresponds either to the value of primitive $\sigma.\texttt{CoAdj}_2()$ (if it has the same source node as σ), or to the value of $\sigma.\texttt{CntAdj}_{2,1}$ (otherwise).

6.3.2.2 Implementation of Kernel Primitives on the Sequential Structure

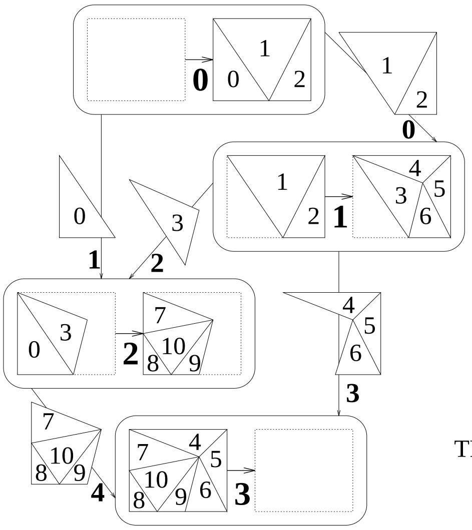
Since arcs are not represented in the structure, it does not make sense to consider the implementation of primitives having arcs among their input or output parameters.

Primitives `Root`, `CellSource`, $\texttt{Cells}_1$, $\texttt{Cells}_2$ are directly represented, and have an optimal time complexity. Operation `Parents` is also supported in optimal time by combining information presented in the structure:

$$C.\texttt{Parents}() = \cup\{\sigma.\texttt{CellSource}() \mid \sigma \in C.\texttt{Cells}_1()\}.$$

Operations `Children` and `CellDest` are not supported. `Children` could only be implemented by scanning the whole node list and collecting all the nodes C which have the given parent. `CellDest` could only be implemented by scanning the whole node list and collecting all the nodes C such that $\Pi_1(C)$ contains the given triangle. Clearly, in both cases the time complexity cannot be optimal.

As far as primitives for querying adjacencies are concerned, the version with one-way adjacency information supports primitives $\texttt{CoAdj}_2$ and $\texttt{CntAdj}_{2,1}$ in $O(1)$ time, while $\texttt{CoAdj}_1$ and $\texttt{CntAdj}_{1,2}$ are not supported. The version with zero-adjacency does not support any of the four adjacency primitives.



NODE ARRAY

	0	1	2	3
first triangleΠ1	-	1	0	4
first triangleΠ2	0	3	7	0

TRIANGLE ARRAY

	0	1	2	3	4	5	6	7	8	9	10
source	0	0	0	1	1	1	1	2	2	2	2
next triangle	3	2	-	-	5	6	7	8	9	10	–

Figure 6.6: An MC (the same as in Figure 6.5) and its representation through the sequential data structure. The figure shows just the part of the structure describing the relations between nodes and triangles: the vertex array, and the fields of the triangle array corresponding to the three vertices and to the approximation error, are not shown.

6.3.2.3 A Sample Implementation of the Sequential Structure

We store the vertices, triangles, and nodes of an MT, in three arrays:

- Each entry in the *vertex* array represents a vertex v, and stores the *three cartesian coordinates* of v.
- Each entry in the *triangle* array represents a triangle σ, and stores: the indexes of the *three vertices* of σ; its *approximation error*; the index of its *source* node, the index of the *next triangle* having the same destination. In the case of one-way adjacency: the indexes of *three adjacent triangles*. Triangles having the same source node are stored at consecutive entries in the triangle array.
- Each entry in the *node* array represents a node C, and stores: the index of the *first triangle* of $\Pi_1(C)$ (all the triangles of $\Pi_1(C)$ are found by scanning the triangle array starting from the first one and following the "next" link); the index of the *first triangle* of $\Pi_2(C)$ (all the remaining triangles of $\Pi_2(C)$ are found by scanning the triangle array from the first triangle, and stopping when encountering a triangle σ such that the source of σ is different from C). Conventionally, the order in which nodes are stored in the array coincides with a scan of the node sequence (since the sequence corresponds to a valid sequences of modification patterns, this means that the root node and the drain node are stored in the first and the last position of the node array, respectively, and, if a node is stored in position i, then all its parents are stored in positions $< i$).

An example is shown in Figure 6.6. The total cost of this implementation of the sequential data structure is:

- $v(3\texttt{real}) + t(8\texttt{int} + 1\texttt{real}) + n(2\texttt{int})$ for one-way adjacency, and
- $v(3\texttt{real}) + t(5\texttt{int} + 1\texttt{real}) + n(2\texttt{int})$ for zero-adjacency.

6.3.2.4 Representation of the DAG Traversal State for the Sequential Structure

Algorithms working on the sequential structure do not traverse a DAG but a sorted list of nodes. For uniformity with other data structures, we still use the term "DAG traversal" here, even if improperly. For the sake of brevity, the information stored to represent the traversal state is described as difference from that maintained with the explicit data structure.

Since arcs are not represented, we do not maintain the current (pseudo-)MC-front $\mathcal{A}$; instead, we maintain the set Σ of triangles forming the current mesh; for the local algorithm, only active triangles are stored in Σ. The triangles of Σ are kept in a linked list; triangles are inserted in the list from the rear, and deleted arbitrarily. In order to support deletion of arbitrary triangles in constant time, we use a binary array to mark virtually deleted elements (as for the list representing $\mathcal{A}$ in the explicit data structure). Moreover, we keep a counter of the unfeasible (active) triangles of Σ, which enables us to test the control condition of the expansion loop.

The set $\mathcal{N}$ of the nodes of the current (pseudo-)sub-MC is kept in an array, which the same conventions as for the explicit structure in the static case.

In summary, the data structures representing the state of DAG traversal are:

- An array with as many entries as the nodes of the MC, representing set $\mathcal{N}$. The contents of such array are the same as described in Section 6.3.1.4, and the space complexity is $n(1\texttt{int})$.
- A linked list containing the triangles of the current mesh. Each item of the list contains the index of a triangle, and a pointer to the next item. The maximum number of triangles in the list is at most t_{sub} and t_{loc} in the basic and local case, respectively; thus, the space required is $\leq t(2\texttt{int})$.
- An array of bits, with as many entries as the triangles of the MC, used to mark triangles that are virtually deleted in the above list. This requires t bits.

As observed for the explicit data structure, if we want to generate the connecting structure for the output mesh by maintaining and updating it during extraction, then we need a data structure for storing the adjacency relations of the current triangle mesh. This can be implemented by enriching the list Σ with three indexes of adjacent triangles for each item, with an additional storage cost of $\leq t(3\texttt{int})$.

In summary, the space required for representing the internal state is

> $n(1\texttt{int}) + t(1 + 2\texttt{int})$, plus $t(3\texttt{int})$ if we maintain the adjacency relations of the current mesh;

for the basic and local algorithms, which are the only supported ones.

6.3.2.5 Implementation of DAG Traversal Primitives for the Sequential Structure

Since the sequential data structure supports just the basic and the local algorithm, we do not consider the specific primitives of the dynamic extractor.

Operation `InitState` behaves as explained for the explicit structure in Section 6.3.1.5 with the difference that, instead of initializing the arc list, we initialize the triangle list representing the current mesh Σ with the triangles of $\Pi_2(C_0)$, where C_0 is the root (only the active ones in the local case); moreover, we initialize the counter of unfeasible (active) triangles of Σ. The cost of the operation is linear in the size of $\Pi_2(C_0)$, multiplied by $\mathcal{T}_{cond}$ in the local case.

Operation `IsInState` is implemented in $O(1)$ as explained for the explicit structure.

Operation `AddToState`$_{node}$ updates the triangle list by deleting the triangles of $\Pi_1(C)$ and adding those of $\Pi_2(C)$, where C is the argument node; `DelFromState`$_{node}$ does the opposite. Moreover, they update the counter of unfeasible (active) triangles of the current mesh. Such operations take a time linear in the number of triangles in $\Pi_1(C) \cup \Pi_2(C)$, multiplied by $\mathcal{T}_{cond}$.

Operation `ContinueExpansion` looks at the counter of unfeasible (active) triangles present in the current mesh Σ and returns true if and only if the counter is zero. The time complexity is constant.

Operation `FindNodeToInclude` is implemented by keeping a cursor on the array of nodes of the MC, placed after the last node included in the sub-MC. In order to find the next node to be included, we advance the cursor until we reach a node C such that $\Pi_1(C)$ contains some unfeasible active triangle from the current mesh Σ. Let C be the node currently at the cursor position. Not all (active) triangles of $\Pi_1(C)$ necessarily belong to the Σ, since some (possibly all) of the parents of C may not be in the current sub-MC. For this purpose, we test only those the triangles of $\Pi_1(C)$ which have their source in the current sub-MC (i.e., which belong to Σ).

As noticed for the explicit structure, we do not have to implement operations `IncludeNode` and `Expansion` here, since their implementation has been completely specified in Section 5.8.3.

The cost of `IncludeNode` on a node C, except the recursive calls, is linear $\Pi_1(C) \cup \Pi_2(C)$, times $\mathcal{T}_{cond}$ in both the basic and the local case. The total cost of calls to `IncludeNode` within `Expansion` is $O(t_{sub}\mathcal{T}_{cond})$ and $O(t_{loc}\mathcal{T}_{cond})$, respectively.

We denote with t_{seq} the total number of triangles contained in the nodes lying between the beginning of th sorted list up to the last node of the (pseudo-)sub-MC giving the solution. The total cost of all the calls to `FindNodeToInclude` within `Expansion` is in $O(\mathcal{T}_{cond} t_{seq})$.

6.3.2.6 Implementation of Query Algorithms for the Sequential Structure

As noticed for the explicit data structure, the only operation that needs to be implemented here is `GetComplex`; `UpdateExtractor` has been completely specified in Section 5.8.3.

From the previous analysis of the costs of its subcomponents, it follows that the cost of `UpdateExtractor` is linear in $(\mathcal{T}_{cond} t_{seq})$. Note that $t_{seq} > t_{sub}$ and $t_{seq} > t_{loc}$ in the global and local case, respectively, as clearly shown in Figure 6.7. Moreover, even if the MC has linear growth, the computational time for answering the query is not necessarily linear in the output size, because extra nodes may be traversed.

Operation `GetComplex` is implemented in a similar way as in the explicit data structure, with the difference that here the list of the triangles of the current mesh is maintained as a part of the current state, and we do not have to pass through arcs. The cost is in $O(t_x)$, where $x = sub, loc$ depending on the algorithm (basic or local, respectively).

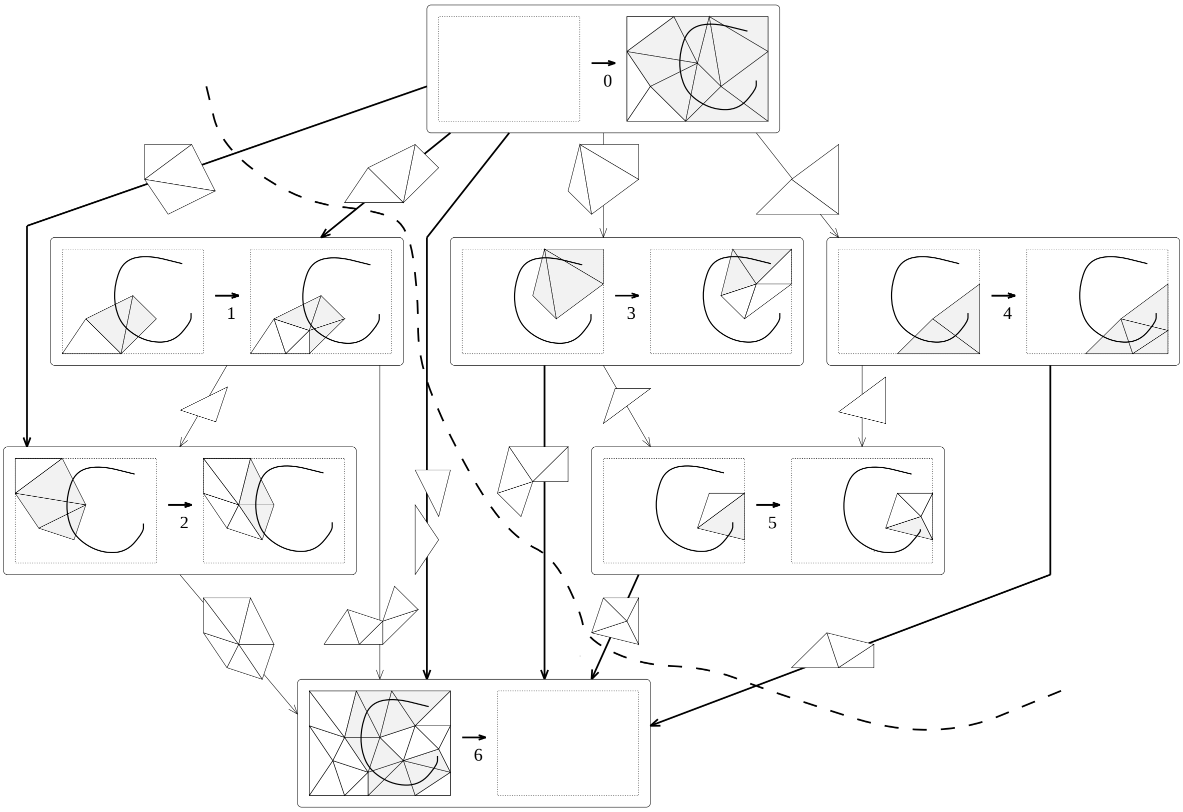
6.3.2.7 Remarks

Unlike in the explicit structure, the implementation given here for the min-size variant of static query algorithms *cannot* be adapted to realize the corresponding best-resolution variants. This is due to the fact that the next node included is found by sequentially scanning the list, and not by looking at the current set of triangles in the mesh.

In the literature, multiresolution models have been proposed which are organized as sequences of modification patterns [SZL92, Hop96]. Unlike the sequential structure proposed here, such models do not store backward dependency links for nodes. The capability of traversing dependency links backwards is fundamental to check whether a node C can be included in the current sub-MC now, or some parents of C have to be included before including C. In [SZL92, Hop96], dependency links must be reconstructed on the fly during a query, thus leading to complicated algorithms. The models in [SZL92, Hop96] do not provide explicitly the triangles of $\Pi_1(C)$ and $\Pi_2(C)$ for each modification pattern C, but they encode them in a compressed way; techniques for encoding triangles in an MC data structure will be discussed in Section 6.4.

6.3.3 A Data Structure Based on Adjacency Information

In this section, we describe a data structure that, instead of representing the DAG explicitly, encodes the modification patterns and their dependency relations through connectivity information.



(a)

(b) (c)

Figure 6.7: We consider a global query with a resolution filter induced by an area of interest consisting of the curve shown. The solution is the one shown. The basic algorithm implemented on the explicit data structure visits nodes $0, 3, 4, 5$ (thus, $n_{sub} = 4$ and $t_{sub} = 26$). Assuming that the sequential data structure stores the nodes sorted as $0, 1, 2, 3, 4, 5, 6$, the basic algorithm implemented on the sequential data structures visits nodes $0, 1, 2, 3, 4, 5$ (thus, $n_{seq} = 6$ and $t_{seq} = 38$).

A fundamental requirement for the MC to be represented with this data structure is that, for each node C, different from the root or drain, the boundaries of $\Pi_1(C)$ and $\Pi_2(C)$ share at least one edge. This prevents the representation of MCs corresponding to layered multiresolution models (see Section 3.3), but it is not a relevant limitation since such models are of little interest.

The partial order of nodes is retrieved from adjacency information by exploiting the following

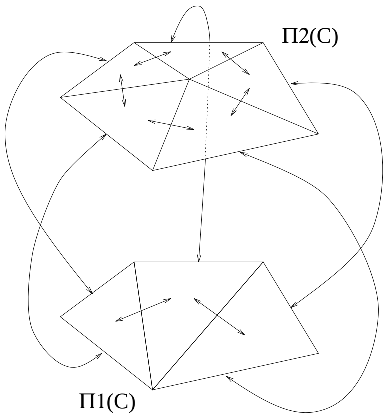


Figure 6.8: Adjacency information in a modification pattern C: arrows within $\Pi_i(C)$ represent $\texttt{CoAdj}_i$ links, for $i = 1, 2$; arrows between $\Pi_1(C)$ and $\Pi_2(C)$ represent $\texttt{CntAdj}_{1,2}$ links (the upward direction of the arrow) and $\texttt{CntAdj}_{2,1}$ links (the downward direction of the arrow).

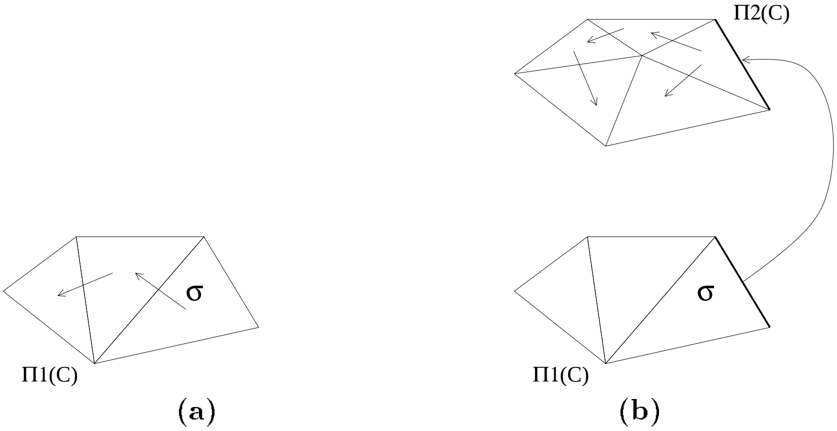


Figure 6.9: (a) Retrieving all the triangles of $\Pi_1(C)$ starting from a triangle $\sigma \in \Pi_1(C)$ through primitive $\texttt{CoAdj}_1$. (b) Retrieving all the triangles of $\Pi_2(C)$ starting from a boundary triangle $\sigma \in \Pi_1(C)$ and using primitives $\texttt{CntAdj}_{1,2}$ and $\texttt{CoAdj}_2$.

properties of adjacency primitives (see Figure 6.8):

- primitive $\texttt{CoAdj}_1$ connects triangles forming mesh $\Pi_1(C)$ for an MC-node C.
- primitive $\texttt{CoAdj}_2$ connects triangles forming mesh $\Pi_2(C)$ for an MC-node C;
- primitives $\texttt{CntAdj}_{1,2}$ and $\texttt{CntAdj}_{2,1}$ connect pairs of triangles belonging one to $\Pi_1(C)$ and the other one to $\Pi_2(C)$ for an MC-node C (we can imagine that the two meshes forming a modification pattern are glued along their common boundary edges);

The triangles forming each of the two meshes of a modification pattern C in an MC are found starting from a triangle $\sigma \in \Pi_i(C)$ (for $i = 1$ or 2), based on the following properties:

- All the triangles of $\Pi_i(C)$ are reachable from σ by iteratively applying primitive $\texttt{CoAdj}_i$ (see Figure 6.9 (a) for the case $i = 1$).
- There is at least one triangle σ' of $\Pi_i(C)$ such that, for some edge e of σ, $\sigma.\texttt{CntAdj}_{i,3-i}(e)$ exists. Let $\sigma' = \sigma.\texttt{CntAdj}_{i,3-i}(e)$; all the triangles of $\Pi_{3-i}(C)$ are reachable from σ' by iteratively applying primitive $\texttt{CoAdj}_{3-i}$ (see Figure 6.9 (b) for the case $i = 1$).

6.3.3.1 High-Level Description of the Adjacency-Based Structure

The data structure stores the vertices and triangles of an MC explicitly, and:

- For each vertex, its coordinates.
- For each triangle: links to its three vertices (primitive `CellVertices`), its approximation error (primitive `CellError`), the index of its source (primitive `CellSource`), the index of its destination (primitive `CellDest`), a total of six links implementing primitives $\texttt{CoAdj}_1$ and $\texttt{CntAdj}_{1,2}$, $\texttt{CoAdj}_2$ and $\texttt{CntAdj}_{2,1}$, as explained in Section 6.3.1.
- No information is stored for a node, but each MT node has a unique index which is used within triangles to refer it.

For the purpose of inclusion in, or elimination from, the current sub-MC, a node C is indirectly represented by its index, and by a triangle $\sigma \in \Pi_1(C)$ such that $\texttt{CntAdj}_{1,2}$ exists at at least one edge of σ (i.e., σ can be used as a "bridge" to reach a triangle of $\Pi_2(C)$). These parameters are passed to primitives `IncludeNode` and `EliminateNode`.

6.3.3.2 Implementation of Kernel Operations on the Adjacency-Based Structure

Since arcs are not represented in the structure, we do not consider primitives having arcs among their input or output parameters.

The four adjacency primitives $\texttt{CoAdj}_1$, $\texttt{CoAdj}_2$, $\texttt{CntAdj}_{1,2}$, $\texttt{CntAdj}_{2,1}$, and primitives `CellSource`, `CellDest` are directly represented in the structure.

Primitives $\texttt{Cells}_1$ and $\texttt{Cells}_2$ for a node C can be retrieved by starting from a triangle, such that $\sigma \in \Pi_1(C)$ or $\sigma \in \Pi_2(C)$, as follows:

1. starting from a triangle $\sigma \in \Pi_1(C)$, the triangles of $C.\mathtt{Cells}_1()$ are found by navigating through primitive $\mathtt{CoAdj}_1$ [1];
2. starting from a triangle $\sigma \in \Pi_2(C)$, the triangles of $C.\mathtt{Cells}_2()$ are found by navigating through primitive $\mathtt{CoAdj}_2$ [2];
3. given a triangle $\sigma \in \Pi_1(C)$, such that $\sigma' = \sigma.\mathtt{CntAdj}_{1,2}(e)$ exists for at least one edge e of σ, all the triangles of $C.\mathtt{Cells}_2()$ are found by starting from σ' as explained in item 2 above, by exploiting the property that $\sigma.\mathtt{CellDest}() = \sigma'.\mathtt{CellSource}()$;
4. given a triangle $\sigma \in \Pi_2(C)$, such that $\sigma' = \sigma.\mathtt{CntAdj}_{2,1}(e)$ exists for at least one edge e of σ, all the triangles of $C.\mathtt{Cells}_1()$ are found by starting from σ' as explained in item 1 above, by exploiting the property that $\sigma.\mathtt{CellSource}() = \sigma'.\mathtt{CellDest}()$.

The time complexity of all implementations described above is linear in the number of triangles returned.

6.3.3.3 A Sample Implementation of the Adjacency-Based Structure

We store the vertices and the triangles of an MT, in two arrays:

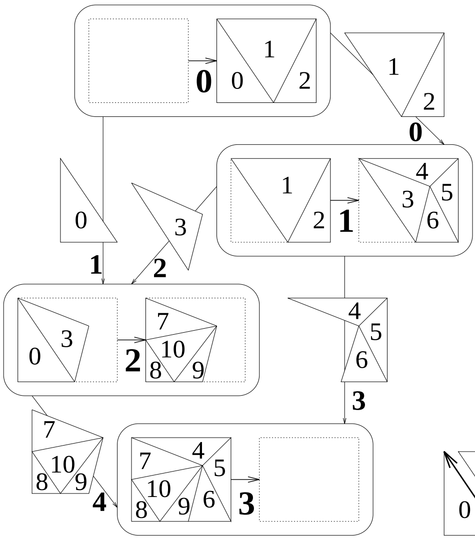
- Each entry in the *vertex* array represents a vertex v, and stores the *three cartesian coordinates* of v.
- Each entry in the *triangle* array represents a triangle σ, and stores: the indexes of the *three vertices* of σ; its *approximation error*; the indexes of its *source and destination* node; the indexes of *six adjacent triangles*; Conventionally, the first and last triangle in the array are a triangle in $\Pi_2(C_0)$ and a triangle in $\Pi_1(C_D)$, respectively, where C_0 and C_D denote the root and the drain, respectively.
- Conventionally, the root node has index equal to one, and the drain node has index equal to the total number of nodes.

An example is shown in Figure 6.10. The total cost of this implementation of the data structure based on adjacency is:

$$v(3\mathtt{real}) + t(11\mathtt{int} + 1\mathtt{real}).$$

[1] We perform a traversal of a graph where nodes are triangles, and an arc exists between any pair of co-adjacency$_1$ triangles; any standard traversal algorithm (e.g., a DFS search) can be applied.

[2] Same remark as for navigation through $\mathtt{CoAdj}_1$.



TRIANGLE ARRAY	0	1	2	3	4	5	6	7	8	9	10	
	0	0	0	1	1	1	1	2	2	2	2	source
	2	1	1	2	3	3	3	3	3	3	3	destination
	3	2	*5*	*7*	5	−	−	10	−	6	7	CoAdj1 or
	−	*4*	1	−	−	4	5	4	10	10	8	CntAdj12
	−	*3*	*6*	*9*	7	6	9	−	−	−	9	
	1	2	−	4	5	*2*	*2*	10	*0*	*3*	7	CoAdj2 or
	−	−	1	*1*	*1*	4	5	*3*	10	10	8	CntAdj21
	−	1	−	6	3	6	3	−	−	−	9	

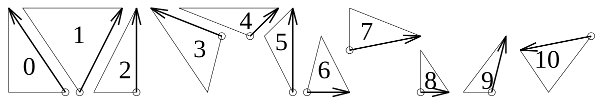


Figure 6.10: An MC (the same as in Figure 6.5) and its representation through the adjacency-based data structure. The figure shows just the part of the structure describing the relations between triangles and node indexes: the vertex array, and the fields of the triangle array corresponding to the three vertices and to the approximation error are not shown. For each triangle σ, the three adjacent triangles provided by each primitive are sorted in the same way as the three edges of σ. The bottom line explains how the sequence of boundary edges of each triangle is sorted: it is sorted in counterclockwise order starting from the arrowed edge.

6.3.3.4 Representation of the DAG Traversal State for the Adjacency-Based Structure

We describe the representation the DAG traversal state as difference with respect to the case of the explicit structure.

The current mesh Σ, i.e., the front mesh of the current sub-MC, is explicitly maintained as a part of the state, and represented through a data structure storing adjacencies between triangles. The adjacency links are incrementally updated by exploiting the four adjacency primitives of the kernel (see Section 6.2.2).

The explicit structure maintains the arc set $\mathcal{A}$ in a list. Since, here, we do not represent arcs, we use a list containing the triangles of the current mesh Σ (i.e., the triangles labelling the arcs of $\mathcal{A}$); in the local algorithm, only active triangles are stored. A bit array is used to mark triangles that are virtually deleted from the list.

The set $\mathcal{N}$ of nodes of the current (pseudo-)sub-MC is represented through an array, as for the explicit structure. We recall that the contents of the entries of such array are different in the static and in the dynamic case.

In the dynamic case, we need a mechanism to keep track leaf nodes. To this aim, we keep, for each node $C \in \mathcal{N}$, two counters and one triangle:

- the number of triangles of $\Pi_2(C)$; this number is computed while including C into $\mathcal{N}$;
- the number of triangles of $\Pi_2(C)$ which belong to the current sub-MC; this counter is updated when including or eliminating either C or one of its children.
- the index of one of the triangles in $\Pi_1(C)$; this triangle is set when including C into $\mathcal{N}$;

A node is a leaf if and only if the values of the two counters are equal. The stored triangle is used as a starting point for traversing $\Pi_1(C)$ in order to test whether all its active triangles are feasible. A list of all leaves is maintained, as explained for the explicit data structure.

In summary, the data structures representing the internal state are:

- An array with as many entries as the nodes of the MC, representing the set $\mathcal{N}$ of nodes belonging to the current (pseudo-)sub-MC. The contents of such array are as described in Section 6.3.1.4, and are different in the static and in the dynamic case. The space required if $n(1\texttt{int})$.
- A linked list containing the triangles of the current mesh Σ. Each item of the list contains the index of an triangle, and a pointer to the next item. The maximum number of elements in the triangle list (including virtually deleted ones), is at most t_{sub} for the basic algorithm, t_{loc} for the local one, and t_{dyn} for the dynamic one. In any case, the list requires a space $\leq t(2\texttt{int})$.
- An array of bits, with as many entries as the triangles of the MC, used to mark triangles that are virtually deleted in the above list. This requires t bits.
- A data structure representing the adjacency relations of the current mesh Σ. This can be implemented by enriching the triangle list and requires an additional space $\leq t(3\texttt{int})$, as described in Section 6.3.2.4. Note that such data structure is optional for the other two data structure (only needed if adjacencies of the current mesh are updated during extraction), but is mandatory here.

In the dynamic case, we store in addition:

- a linked list of leaf nodes, where each item contains the index of a node, and a pointer to the next item. The maximum number of elements in the leaf list is at most n_{dyn}, thus the list needs at most $n_{dyn}(2\texttt{int}) \leq n(2\texttt{int})$.
- three arrays, each with one entry for every MC-node, containing the two integer counters and the triangle associated with the node, as explained above. The space required is $n(3\texttt{int})$.

In summary, the space required for representing the internal state is:

- $n(1\texttt{int}) + t(1 + 5\texttt{int})$ for the basic and local algorithms;
- $n(6\texttt{int}) + t(1 + 5\texttt{int})$ for the dynamic one.

6.3.3.5 Implementation of DAG Traversal Primitives for the Adjacency-Based Structure

Since a node index has no associated information, all operations having a node among their input or output parameters change it into a triangle; the corresponding node will be the destination of such triangle.

Operation `InitState` behaves as in the explicit data structure and, in addition, it initializes the structure for the current mesh as a copy of $\Pi_2(C_0)$ (where C_0 is the root of the MC) and the list of triangles; only active triangles of $\Pi_2(C_0)$ are considered in the local case. The time complexity is linear in the size of $\Pi_2(C_0)$, multiplied by $\mathcal{T}_{cond}$ in the local case.

Operation `IsInState` is implemented as in the explicit structure, in $O(1)$.

The implementation of `IncludeNode` with this data structure does not use primitives $\texttt{AddToState}_{node}$ and $\texttt{DelFromState}_{node}$; therefore, they do not need to be implemented.

Operations `ContinueExpansion` scans the triangle list with the help of a cursor that is placed after the last examined triangle; the triangle list is managed here in similar way as the list of arcs in the explicit structure. In the dynamic algorithm, and in the case of a negative result, `ContinueExpansion` also initializes the leaf list.

Operation `FindNodeToInclude` returns a triangle instead of a node: it returns a triangle of the current mesh belonging to $\Pi_1(C)$, where C is the node that must be included. This operation reads the triangle σ at the head of the triangle list, and returns σ.

The node argument C of operation `IncludeNode` is replaced with a triangle $\sigma \in \Pi_1(C)$, which belongs to the current mesh; in the outer call, such triangle is the one returned by `FindNodeToInclude`. From σ, we start a simultaneous traversal of $\Pi_1(C)$ and of the current mesh; two cases may occur:

1. The simultaneous traversal ends after having visited all the triangles of $\Pi_1(C)$ (i.e., $\Pi_1(C)$ is entirely contained in the current mesh). This means that all the parents of C are already in $\mathcal{N}$. Thus, we include C by simply replacing the triangles of $\Pi_1(C)$ with those of $\Pi_2(C)$ in the current mesh, and updating adjacency links. In order to find the triangles of $\Pi_2(C)$, we start from a triangle σ' of $\Pi_1(C)$ (which has been determined during the traversal) such that $\sigma'.\texttt{CntAdj}_{1,2}(e)$ exists for at least one edge e of σ'. Note that $\sigma'.\texttt{CntAdj}_{1,2}(e).\texttt{CellSource}() = \sigma.\texttt{CellDest}()$, thus we retrieve $\Pi_2(C)$ starting from $\sigma'.\texttt{CntAdj}_{1,2}(e)$ and using links $\texttt{CoAdj}_2$ (see Figure 6.11).

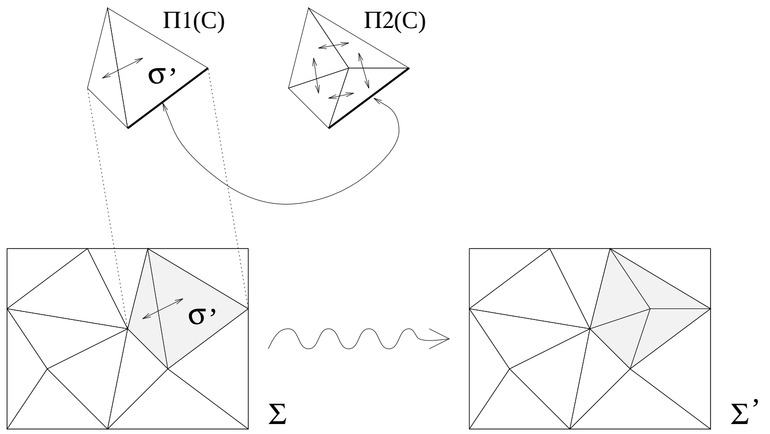


Figure 6.11: All triangles of $\Pi_1(C)$ are in the current mesh Σ; Σ is updated into Σ' by replacing $\Pi_1(C)$ with $\Pi_2(C)$; All the triangles of $\Pi_2(C)$ are found starting from σ, using primitive $\mathtt{CntAdj}_{1,2}$ once, and $\mathtt{CoAdj}_2$ repeatedly.

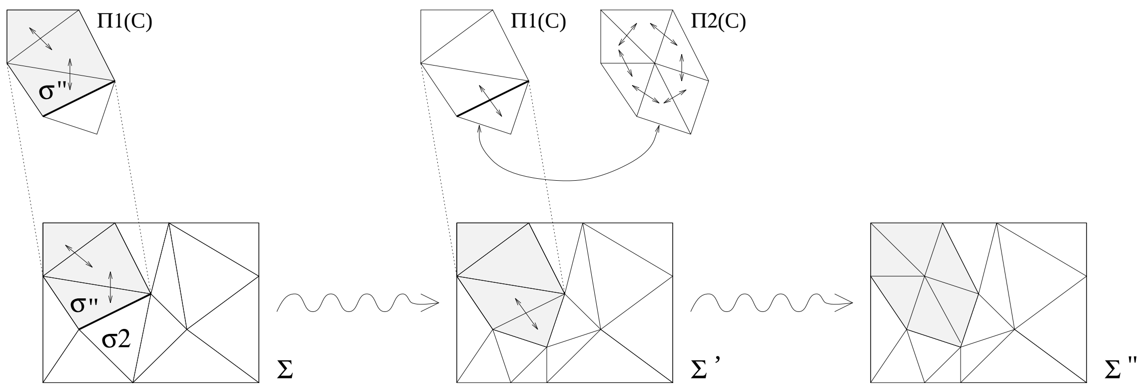


Figure 6.12: Not all triangles of $\Pi_1(C)$ are in the current mesh Σ; we stop on a triangle σ'' where the adjacent triangle in Σ (here, σ_2) is different from the one in $\Pi_1(C)$. We first update Σ into Σ' by including the destination of σ_2, and then Σ' into Σ'' by replacing $\Pi_1(C)$ with $\Pi_2(C)$.

2. Otherwise, the simultaneous traversal stops at a triangle σ'' and an edge e' of σ'' such that $\sigma_1 \equiv \sigma''.\mathtt{CoAdj}_1(e')$ (i.e., the adjacent to σ'' along e' in $\Pi_1(C)$) is different from the adjacent σ_2 to σ'' along e' in the current mesh. Then, we have to include the destination C_2 of σ_2 before including C. In order to do that, we recursively call `IncludeNode` on σ_2 and C_2. Then, we resume the simultaneous traversal of $\Pi_1(C)$ and of the current mesh (see Figure 6.12).

In addition, in the dynamic algorithm, `IncludeNode` counts the number of triangles of $\Pi_2(C)$ (where C is the given node) and stores it in the two arrays at the entry corresponding to C; it stores a triangle of $\Pi_1(C)$ in the appropriate array; for each parent C' of C, it decreases the counter of the triangles of $\Pi_2(C')$ belonging to the current mesh.

Operation `ContinueContraction` is implemented by scanning the leaf list with a cursor, as in the explicit data structure. The test whether a leaf can be eliminated is performed by traversing $\Pi_1(C)$ starting from the triangle σ stored for C. Note that σ is any triangle of $\Pi_1(C)$; during such traversal, we replace σ with a triangle σ', still in $\Pi_1(C)$, such that $\sigma'.\mathtt{CntAdj}_{1,2}(e)$ is defined for at least one of its edges (this feature will be useful when eliminating C).

Operations `FindNodeToEliminate` returns a triangle instead of a node. It reads the first element of the leaf list, and, if C is the node, it returns the triangle stored for C.

The argument node C of operation `EliminateNode` is replaced with a triangle $\sigma \in \Pi_1(C)$ such that $\sigma.\mathtt{CntAdj}_{1,2}(e)$ is defined for at least one edge e of σ; such triangle is the one associated with C in the array.

Operation `EliminateNode` works exactly as `IncludeNode`, with the difference that Π_1 plays the role of Π_2, and Π_2 that of Π_1. From $\sigma.\mathtt{CntAdj}_{1,2}(e)$, we start a simultaneous traversal of $\Pi_2(C)$ and of the current mesh. In this case, the traversal always ends after having visited all the triangles of $\Pi_2(C)$ (since C is a leaf). Then, we eliminate C by simply replacing the triangles of $\Pi_2(C)$ with those of $\Pi_1(C)$ in the current mesh, and updating adjacency links.

The time complexity of `IncludeNode` and `EliminateNode`, for a node C, is linear in the size of $\Pi_1(C) \cup \Pi_2(C)$, multiplied by $\mathcal{T}_{cond}$ in the local case since we have to test whether a triangle is active or not.

6.3.3.6 Implementation of Query Algorithms for the Adjacency-Based Structure

We evaluate just the cost of `UpdateExtractor` since its implementation is already determined. The cost is:

- For the static algorithms: linear in the number of triangles contained in the (pseudo-)sub-MC defining the result, multiplied by $\mathcal{T}_{cond}$, i.e., in $O(\mathcal{T}_{cond} t_{sub})$ for the basic algorithm, and $O(\mathcal{T}_{cond} t_{loc})$ in the local one; if the MC has linear growth, then t_{sub} and t_{loc} are linear in the output size.

- For the dynamic algorithm: linear in the number of triangles contained in the modification patterns of the visited portion of the DAG, i.e., $O(\mathcal{T}_{cond} t_{dyn})$; if the MC has linear growth, then t_{dyn} is linear in the sum of input and output size.

Operation `GetComplex` is implemented in a straight-forward way by outputting the current front mesh, which is explicitly maintained as a part of the traversal state. The cost is linear in the output size.

6.3.3.7 Remarks

The best-resolution variants of query algorithms can be defined by changing the policy of management of the list of triangles and of the list of leaf nodes, which become priority queues, as explained for the explicit structure.

Another adjacency-based structure for a multiresolution model was proposed by Cignoni et al. [CPS97]. Their data structure is a modification of the facet-edge data structure for representing cell complexes in $\mathbb{E}^3$. The structure does not store triangles as triplets of vertices, but, for each triangle σ, it stores three elements, called *facet-edges*, each corresponding to an edge e of σ. Each facet-edge points to the next facet-edge around the same triangle. A facet-edge, representing an edge e of a triangle σ, stores just the first endpoint of e, and the second endpoint is found in the next facet-edge.

Facet-edges of different triangles sharing the same edge are connected through pointers. In terms of MC primitives, each facet-edge, representing an edge e of a triangle σ, stores two pointers corresponding to $\sigma.\mathtt{CntAdj}_{1,2}(e)$ and $\sigma.\mathtt{CntAdj}_{2,1}(e)$, and a third pointer, which corresponds to $\sigma.\mathtt{CoAdj}_2(e)$, or, if $\sigma.\mathtt{CoAdj}_2(e)$ is not defined, to $\sigma.\mathtt{CntAdj}_{2,1}(e).\mathtt{CoAdj}_2(e)$ (in [CPS97], it is defined as the triangle that was created adjacent to σ along e when σ was created, or the triangle incident to e that survived when σ was created).

If t is the number of triangles in the model, the resulting data structure has $3t$ facet-edges, each storing a pointer to a vertex and four pointers to other facet edges, with a space requirement of $t(15\mathtt{int})$. Our adjacency-based data structure requires just $t(11\mathtt{int})$ (in both cases, we have not counted the space for vertex coordinates and approximation errors). Thus, our structure is more compact.

Cignoni et al. only provide a static algorithm for a global query working on their structure, which can generate meshes only for error threshold functions decreasing with distance from a point, and does not allow specifying an area of interest. Conversely, the adjacency-based data structure proposed here supports both global and local query, any resolution filter and focus set, and a dynamic algorithm for the global query.

6.3.4 Extensions to Higher Dimensions

The three families of data structures introduced in Section 6.3 and the related implementations of kernel, DAG traversal, and query primitives, extend to k-dimensional simplicial complexes in $\mathbb{R}^d$ for arbitrary k and d, in a direct way.

The only changes are in the number of coordinates describing each vertex, in the number of vertices describing each top-simplex, and in number of adjacency links for each top-simplex. This last difference is relevant only for adjacency-based data structures and for explicit or sequential structures with half- or full-adjacency.

Note that, since adjacency primitives must necessarily be stored, the complexity of the adjacency-based structure rapidly explodes when increasing the dimension of the Multi-Complex. For the other two families of structures, the same considerations suggest that versions with half- or full-adjacency are less convenient in higher-dimensions.

6.3.5 Comparison of General-Purpose Data Structures

In this subsection, we compare the three families of data structures proposed. We consider *query algorithms* supported by the three families of data structures in terms of their efficiency and simplicity of implementation.

The explicit data structure supports all the three algorithms considered and their implementation is simple and clear. Moreover, modifying the algorithms described here to solve the best-resolution variant of the query is not difficult.

The sequential structure does not support the dynamic algorithm, and the local algorithm is not efficient. The implementation of the basic algorithm is simple, but the time complexity is worse than in the other two structures because of the need of traversing nodes that do not contribute to the solution. Moreover, the approach cannot be extended to best-resolution queries.

The adjacency-based structure supports all three query algorithms, but the implementations are much more complicated than in the other two data structures.

As far as the storage requirements are concerned, Table 6.1 summarizes the space requirements of the three structures (in the sample implementations), and table 6.2 shows their sizes on three specific MCs (evaluated according to the formulae provided in the previous subsections).

We recall that full-adjacency is needed if we want to get adjacency information for the extracted mesh in both static and dynamic query algorithms, paying only a linear cost in the number of triangles associated to nodes and arcs of the traversed portion of the DAG. Half-adjacency is sufficient for such purpose within static algorithms. With zero-adjacency, the connectivity of the extracted mesh, if needed, must be reconstructed either in a preprocessing step or during DAG traversal, by using additional search structures, as discussed in Sections 6.2.1 and 6.2.3. Note that taking the rows marked with full- or half-adjacency in Table 6.1 (a) only makes sense if we take the "optional" column in Table 6.1 (b). Some meaningful combinations are the ones considered in Table 6.2.

In the case of full-adjacency, the adjacency-based data structure saves up to $t(4\texttt{int})$ with respect to the explicit one in representing the MC, but the quantity of memory it needs for representing the DAG traversal state is either comparable with that of the explicit structure (for a static query algorithm), or larger (for a dynamic one). Experimental results confirm such estimates. The explicit structure is more compact than the adjacency-based one in all other cases (note that the adjacency-based structure has full adjacency information in *any* case, even if not needed in the output).

MC	adj info	space	
explicit	full	$t(10\texttt{int} + 1\texttt{real}) + a(4\texttt{int}) + n(2\texttt{int})$	$\leq t(15\texttt{int} + 1\texttt{real})$
	half	$t(7\texttt{int} + 1\texttt{real}) + a(4\texttt{int}) + n(2\texttt{int})$	$\leq t(13\texttt{int} + 1\texttt{real})$
	zero	$t(4\texttt{int} + 1\texttt{real}) + a(4\texttt{int}) + n(2\texttt{int})$	$\leq t(10\texttt{int} + 1\texttt{real})$
sequence	half	$t(8\texttt{int} + 1\texttt{real}) + n(2\texttt{int})$	$\leq t(9\texttt{int} + 1\texttt{real})$
	zero	$t(5\texttt{int} + 1\texttt{real}) + n(2\texttt{int})$	$\leq t(6\texttt{int} + 1\texttt{real})$
adj-based	full	$t(11\texttt{int} + 1\texttt{real})$	$= t(11\texttt{int} + 1\texttt{real})$

(a)

state	adj info	space		optional
explicit	static	$a(1 + 2\texttt{int}) + n(1\texttt{int})$	$\leq t(1 + 2.5\texttt{int})$	$+t(3\texttt{int})$
	dynamic	$a(1 + 2\texttt{int}) + n(3\texttt{int})$	$\leq t(1 + 3.5\texttt{int})$	$+t(3\texttt{int})$
sequence	static	$n(1\texttt{int}) + t(1 + 2\texttt{int})$	$\leq t(1 + 2.5\texttt{int})$	$+t(3\texttt{int})$
adj-based	static	$n(1\texttt{int}) + t(1 + 5\texttt{int})$	$\leq t(1 + 5.5\texttt{int})$	0
	dynamic	$n(6\texttt{int}) + t(1 + 5\texttt{int})$	$\leq t(1 + 8\texttt{int})$	0

(b)

Table 6.1: Storage requirements of the sample implementations for the three families of data structures: (a) representation of the MC; (b) representation of the internal state of DAG traversal. Table (a) does not report the space needed for vertices, which is the same in all structures. The "optional" column in (b) contains the extra space needed for reconstructing adjacency relations of the output mesh; no additional space is necessary with the adjacency-based structure.

Compared with the explicit one, the sequential data structure saves $t(4\texttt{int})$ in the MC representation, with theoretically almost the same size for the DAG traversal state; experiments show larger space requirements for the state in the sequential case, due to the fact that the theoretical bound $a \leq t$ used in Table 6.1 is not tight (in practice, there are relevantly fewer arcs than triangles). However, only the basic algorithm is supported in the sequential structure, and it has a worse time complexity.

In principle, the presence of adjacency information in an MC data structure is motivated by the need to generate the connectivity of the output mesh without having to reconstruct it. Data structures that do not store adjacency primitives have to reconstruct adjacency links by using search structures. Let us consider the approach using *incremental* reconstruction during DAG traversal (see Section 6.2.3). In theory, this is suboptimal for a factor $\log M$, where M is the maximum size of $\Pi_2(C)$ for all nodes C in the MC. If the MC has bounded width, then $M = O(1)$, and the time is optimal; in practical MCs, M is small, thus the time is almost optimal, and the save in memory space is high.

The explicit data structure (in the version with zero-adjacency) has been implemented both

Terrain: 14767 vertices, 73089 triangles, 14765 nodes, 42969 arcs

structure (adj)	output adj generation	MC size	state size (total size MC+state)			
			static		dynamic	
explicit (zero)	no	2267404	408184	(2675588)	526304	(2793708)
explicit (zero)	yes	2267404	1285252	(3552656)	1403372	(3670776)
explicit (half)	yes	3144472	1285252	(4429724)	–	–
explicit (full)	yes	4021540	–	–	1403372	(5424912)
sequence (zero)	no	1872256	652909	(2525165)	–	–
sequence (zero)	yes	1872256	1529977	(3402233)	–	–
sequence (half)	yes	2749324	1529977	(4279301)	–	–
adj-based	yes	3508272	1529977	(5038249)	1825277	(5333549)

Bunny: 34834 vertices, 194927 triangles, 33946 nodes, 96134 arcs

structure (adj)	output adj generation	MC size	state size (total size MC+state)			
			static		dynamic	
explicit (zero)	no	5708252	916873	(6625125)	1188441	(6896693)
explicit (zero)	yes	5708252	3255997	(8964249)	3527565	(9235817)
explicit (half)	yes	8047376	3255997	(11303373)	–	–
explicit (full)	yes	10386500	–	–	3527565	(13914065)
sequence (zero)	no	4949816	1719566	(6669382)	–	–
sequence (zero)	yes	4949816	4058690	(9008506)	–	–
sequence (half)	yes	7288940	4058690	(11347630)	–	–
adj-based	yes	9356496	4058690	(13415186)	4737610	(14094106)

Fandisk: 6475 vertices, 36829 triangles, 6317 nodes, 17848 arcs

structure (adj)	output adj generation	MC size	state size (total size MC+state)			
			static		dynamic	
explicit (zero)	no	1072684	612231	(1242967)	662767	(1293503)
explicit (zero)	yes	1072684	612231	(1684915)	662767	(1735451)
explicit (half)	yes	1514632	612231	(2126863)	–	–
explicit (full)	yes	1956580	–	–	662767	(2619347)
sequence (zero)	no	934432	324504	(1258936)	–	–
sequence (zero)	yes	934432	766452	(1700884)	–	–
sequence (half)	yes	1376380	766452	(2142832)	–	–
adj-based	yes	1767792	766452	(2534244)	892792	(2660584)

Table 6.2: Space requirements for the MC data structure and for the internal state of mesh extraction for the three sample data structures on some MCs. We have assumed that both `int` and `real` are represented on 32 bits; sizes are in bytes.

for the two- and in the three-dimensional MCs, along with the algorithms for the three variants of queries (i.e., the basic, the dynamic, and the local one). We are currently extending it with storage of half- and full-adjacency. Experiments reported in this thesis are obtained from such implementations. The other two data structures have not been implemented.

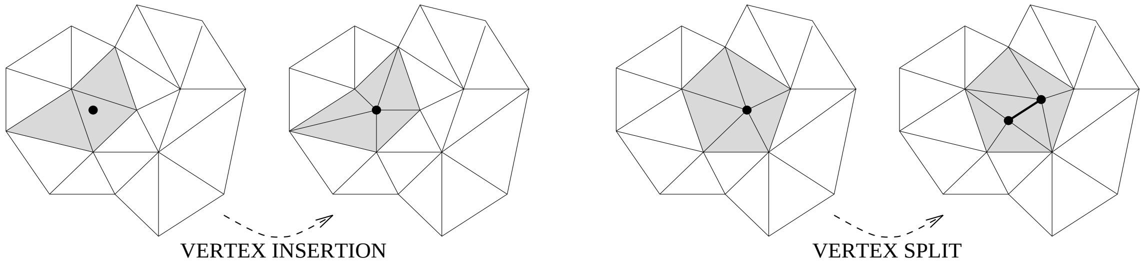


Figure 6.13: Two types of modification patterns which allow the design of compressed data structures for an MC. The shaded triangles are those involved in the modification pattern.

The explicit data structure has been chosen as *the reference data structure* for MCs because it is the one that best reflects the underlying Multi-Complex, it has no limitations on the class of representable MCs, and it allows an easy and efficient implementation of all query algorithms with a reasonable storage space.

6.4 Compressed Data Structures

In this section, we address the design of compressed data structure which can represent special classes of MCs more compactly than general-purpose data structures presented in Section 6.3.

Looking at the storage requirements of general data structures (shown in Table 6.1), we see that, even in the variant with no adjacency information, a large part of the overall memory space is used for triangles. A triangle needs at list three vertex indices and an approximation error, i.e., 3`int` + 1`real`. The basic idea of compressed data structures is to reduce storage cost by *avoiding an explicit representation of triangles.*

In order to admit a compressed data structure, all the modification patterns of an MC must be of a certain predefined type such that the triangles involved in them can be described in an implicit way. Examples of schemes of modification patterns are vertex insertion and vertex split (i.e., expanding a vertex into an edge), shown in Figure 6.13. Each compressed structure is designed for MCs in which modification patterns are of a specific type.

Compressed data structures do not provide all the information characterizing an MC, and, thus, are intrinsically *lossy.* Missing information is *approximated* based on the available information. Because of this approximation, the quality of extracted meshes may be not as good as by using general-purpose data structures (e.g., they may contain more triangles for the same parameters of a min-size query). Techniques for compressing triangles in an MC are not compatible with the adoption of an adjacency-based data structure, since they do not represent triangles; they can be used within any of the two other families of structures.

In this section, we propose two compressed data structures that we have designed for MCs based on *vertex insertion*, i.e., MCs where, for each modification pattern C (except at most the root), the vertices of $\Pi_2(C)$ are the same as those $\Pi_1(C)$ plus an additional vertex p, and all triangles of $\Pi_2(C)$ are incident at p. Such MCs can be built in a top-down approach through incremental insertion of vertices, or bottom-up through iterative vertex removal (see Section 7.5).

Before describing the structures, we outline the general scheme of a compressed structure parametrically on the type of modification patterns involved, and, thus, on the information used to compactly describe them.

6.4.1 General Scheme of a Compressed Data Structure

Compressed structures do not store the triangles of an MC as individual entities. Based on fact that all modification patterns are of a predefined type, such structures provide, within each MC-node C, just a few parameters that are sufficient to describe how a mesh is modified when applying the modification pattern corresponding to C, or the reverse modification pattern of C. Such information is sufficient to modify the current mesh when C is included in the current sub-MC, or, in the dynamic algorithm, when C is eliminated from the sub-MC. The type and number of parameters depend on the specific type of modification patterns for which a certain compressed data structure is designed.

Since compressed structures do not store triangles as individual entities, they do not provide information about the approximation error of a specific triangle σ, but they attach a single approximation error to each node. For the same reason, they do not provide information about the arc containing σ in its label. In this sense, such structures are lossy.

6.4.1.1 Compressed Explicit Structures

A *compressed explicit data structure* stores the vertices, nodes, and arcs of an MC. For vertices, it stores the same information as the general-purpose explicit structure; for arcs and nodes, it stores:

- For each node C: links to its incoming arcs (primitive `InArcs`), links to its outgoing arcs (primitive `OutArcs`), the maximum error of a triangle in $\Pi_2(C)$ and the maximum error of a triangle in $\Pi_1(C)$ (this latter only needed for supporting the dynamic algorithm); moreover, the appropriate *parameter values* (depending on the specific compressed structure).
- For each arc: a link to its *source* (primitive `Source`), a link to its *destination* (primitive `Dest`).

The sample implementation of Section 6.3 is modified as follows:

- For each node C: we store the index of the *first arc outgoing* from C; the index of the *first arc entering* C; *two approximation errors*, and the specific *parameters* describing the corresponding modification pattern.
- For each arc A: we store the indexes of the *source and destination* of A; the index of the *first triangle* associated with A and the index of the *next arc* having the same destination node. Arcs with the same source are stored at consecutive positions.

Let us denote with $\mathcal{S}_{opt}$ the space required by the parameters describing the pattern. The space required by the data structure is $v(3\texttt{real}) + a(4\texttt{int}) + n(2\texttt{int} + 2\texttt{real} + \mathcal{S}_{opt})$.

Kernel primitives are implemented as in the explicit structure (see Section 6.3.1), except that all primitives involving triangles among their input and/or output parameters are not supported.

A representation of the current triangle mesh is explicitly maintained as a part of the DAG traversal state in a format that provides adjacency relations among the triangles. In fact, these are needed in order to modify the current mesh.

When the current mesh must be updated by including or eliminating a node C, the triangles of $\texttt{Cells}_1(C)$ and $\texttt{Cells}_2(C)$ are constructed on the fly, by using information contained in the node, connectivity of the current mesh, and knowledge about the type of modification pattern represented in C. Each triangle introduced in the current mesh by including or eliminating a node C receives the error value stored in C related to $\Pi_2(C)$ or $\Pi_1(C)$, respectively. Thus, for the triangles of the current mesh, primitive `CellError` is available, and it is a pessimistic approximation of their real approximation errors.

Now, we consider the implementation of operations concerned with the determination of nodes to include and to eliminate (in the dynamic case).

First, let us consider the static case. In Section 6.3.1, operations `ContinueExpansion` and `FindNodeToInclude` are implemented by keeping the current front arcs in a list and by using the stored primitive `ArcCells` to access triangles from arcs. Here, we replace such arc list with a list of the triangles of the current mesh Σ; for every triangle, we also maintain a pointer to its source node. A triangle σ is inserted in this list when it is created in Σ, i.e., when including its source node in the current sub-MC; thus, at that moment the source of σ is known. Operation `ContinueExpansion` behaves in a similar way as in Section 6.3.1, simply moving a cursor on a list of triangles instead of a list of arcs. If `ContinueExpansion` returns true, then `FindNodeToInclude` reads the triangle at the cursor position and returns all the children of the source of such triangles to be included into the current sub-MC.

The implementation of `FindNodeToInclude` is different in the dynamic algorithm. In this case, a triangle σ is inserted in Σ either when including the source of σ, or when eliminating its destination; thus, either the source or the destination of σ is known. We store, for each triangle $\sigma \in \Sigma$, a pointer to either its source of to its destination, with a binary flag indicating which is the case. Operation `FindNodeToInclude` reads the triangle σ at the position of the cursor. If

σ points to its destination, then we include the destination of σ; otherwise, we include all the children of its source (as in the static case).

Operations `ContinueContraction` and `FindNodeToEliminate` are implemented by keeping a list of leaf nodes, as in the general-purpose explicit structure. In order to check whether the elimination of a node C brings some unfeasible active triangle in the current mesh, we have to temporarily create the triangles of $\Pi_1(C)$ by simulating an update of the mesh.

6.4.1.2 Compressed Sequential Structures

A *compressed sequential data structure* stores the vertices, the nodes and the arcs of an MC, and the following information for nodes and arcs:

- For each node C: the next node in the sequence, links to its incoming arcs (primitive `InArcs`), one *approximation error* (i.e., the maximum error of a triangle in $\Pi_2(C)$), and the appropriate *parameter values* (depending on the specific compressed structure).
- For each arc, its source node (primitive `Source`).

We need to make two remarks. First, we have introduced arcs to connect MC-nodes, where this role was played by triangles in the general-purpose sequential structure.

Secondly, note that is not necessary to store the second approximation error of a node (i.e., the maximum error of a triangle in $\Pi_1(C)$) because this is only needed for the contraction phase of the dynamic algorithm, which is not supported by sequential data structures.

The sample implementation of Section 6.3.2 is modified as follows. Arcs are stored in an additional array, and the contents of the node and arc array are:

- For each node C: the index of the *first and last arc entering* C (the other incoming arcs of C lie between such two arcs of the arc array), *one approximation errors*, and the specific *parameters* describing the corresponding modification pattern. Nodes are sorted in the array according to the given sequence.
- For each arc A: the index of the *source* of A. The arc array is sorted in such a way that arcs entering the same node are consecutive.

The space required by this sample compressed sequential structure is $v(3\texttt{real}) + n(2\texttt{int} + 1\texttt{real} + \mathcal{S}_{opt}) + a(1\texttt{int})$, where $\mathcal{S}_{opt}$ is the size of the parameters.

Kernel primitives are implemented as in the sequential structure (see Section 6.3.2), except that all primitives involving triangles among their input and/or output parameters are not supported.

As in the previous case, a representation of the current triangle mesh with adjacency relations is explicitly maintained as a part of the DAG traversal state, and it is updated at any inclusion of a node C in the current sub-MC; on this occasion, the triangles of $\mathtt{Cells}_1(C)$ and $\mathtt{Cells}_2(C)$ are determined on the fly.

In order to support operation `FindNodeToInclude`, we mark all the nodes $C \in N$, that contain some active and not feasible triangle in $\Pi_2(C)$. A node C is marked inside operation `IncludeNode`: while updating the current mesh, the new triangles are checked and, if some of them is active and not feasible, then C is marked. Operation `FindNodeToInclude` scans the list of nodes, and, for every node, checks if some of its parents is marked; if so, then the current node is returned to be included in the sub-MC.

6.4.1.3 Quality of Extracted Meshes

Meshes returned from global query algorithms on MCs encoded through compressed data structures have in general a larger size than those returned from a general-purpose one for the same query parameters. This happens for two reasons:

1. Approximation errors are provided for entire nodes, and for each node C the two errors stored at C are the maximum error of a triangle in $\Pi_2(C)$ and $\Pi_1(C)$, respectively. Such errors are set during the construction of the data structure, and individual triangle errors are forgotten after the data structure has been built. Every triangle σ, when it is reconstructed during DAG traversal, receives the approximation error either from its source or from its destination; this is an upper bound to the real approximation error of σ; thus, based on information available in the compressed data structure, σ may appear as not feasible, even if σ is feasible.

2. The data structure provides dependency links between nodes but does not tell which triangles label each arc. Thus, if an active triangle $\sigma \in \Pi_2(C)$ is (or appears to be) not feasible, we do not know which child of C is the destination of σ (and, thus, which child must be included in the sub-MC): the only possible choice is including all the children of C. This leads to over-refine some areas in which a lower resolution would be sufficient.

In Section 6.4.4, we will compare the meshes extracted from a compressed version and the general-purpose version of the explicit structure.

6.4.2 Some Sample Compressed Structures

In this subsection, we propose two compressed data structures for MCs based on vertex insertions in a Delaunay triangulation and vertex insertion in a generic triangle mesh, respectively. Such structures were proposed in [DFMP98c].

Compressed structures can be designed for other types of modification patterns as well. Note that vertex splits of the kind where splitting a vertex v generates an edge having one endpoint in v and the other endpoint in a new vertex v' are a special case of vertex insertion and, thus, the second data structure proposed here applies to them as well. Generic vertex splits (where both endpoints of the edge generated by splitting v are new vertices) need compressed structures especially designed.

6.4.2.1 A Structure for Delaunay Meshes

We present a compressed structure designed for two-dimensional simplicial MCs in $\mathbb{E}^2$ (such as the ones used to represent the domain of a two-dimensional field or of a parametric patch) based on Delaunay triangulations.

In a Delaunay MC $\mathcal{M}$ with root C_0, $\Pi_2(C_0)$ is a Delaunay triangulation, and every other MC-node C_i represents the insertion of a new vertex in a Delaunay triangulation. Thus, every combined cell complex is a Delaunay triangulation.

For a set of points in general positions (no four points are co-circular), the Delaunay triangulation is unique; thus, a Delaunay triangulation is completely determined by the set of its vertices and the modification due to the insertion of a new vertex is completely determined by the vertex being inserted.

The basic idea is the following, illustrated in Figure 6.14:

- for the root C_0, we store the vertices of $\Pi_2(C_0)$;
- for any other node C_i, we store the extra vertex of $\Pi_2(C_i)$ with respect to $\Pi_1(C_i)$.

Information stored in the root is sufficient to initialize the current mesh in the static query algorithms (i.e., the basic and the local one); such operation reduces to computing the Delaunay triangulation of a set of points. Given the current DAG traversal state and the corresponding mesh, the vertex stored in a node C is sufficient to determine how the current mesh has to be updated when including or eliminating C. This operation reduces to vertex insertion or deletion in a Delaunay triangulation, for which algorithms are well-known (see Section 3.2.1).

For static query algorithms (i.e. the basic and the local one), we can quantify the total time required as the time complexity of building a Delaunay triangulation of r points (where r is the number of vertices in the extracted mesh), with an on-line algorithm. This time is $O(r^2)$ in the worst case, and $O(r \log r)$ in the expected case [BT93].

The compression scheme for Delaunay MCs can be applied to the explicit or to the sequential data structure, as explained in Section 6.4.1. Here, the information stored at each node is a vertex (a set of vertices for the root).

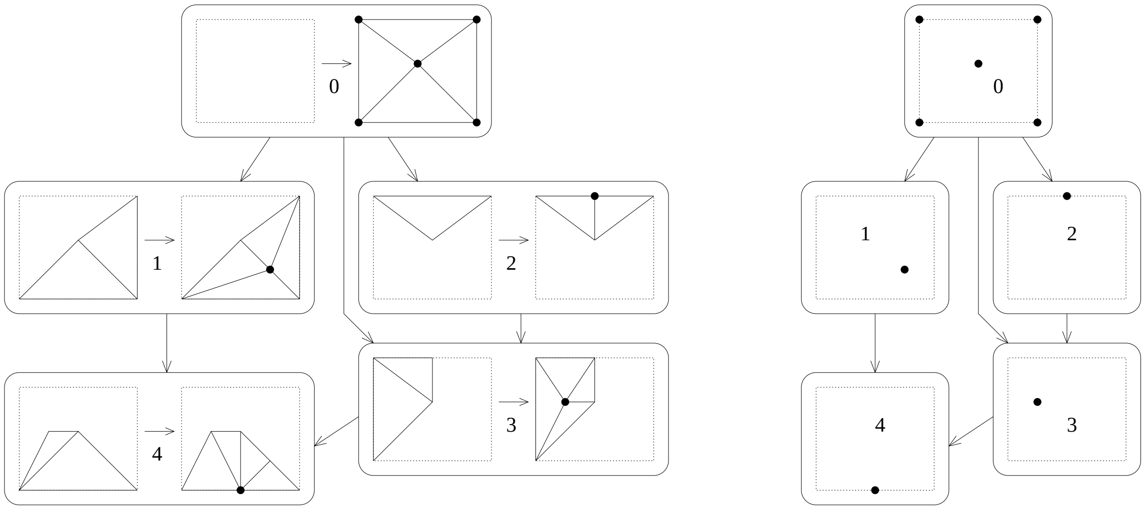


Figure 6.14: A Delaunay MC and the vertices stored at each node of the compressed structure.

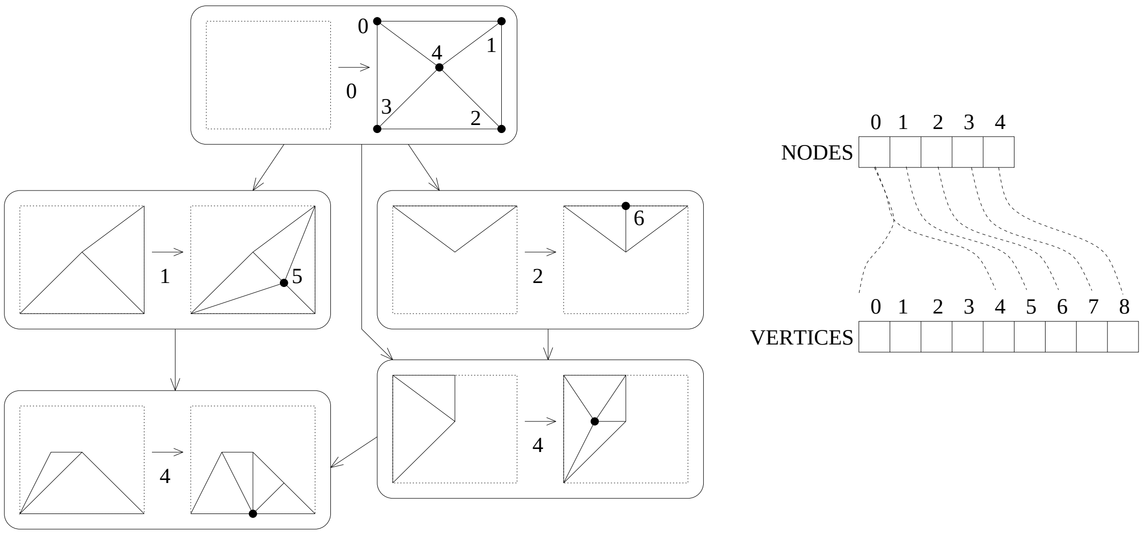


Figure 6.15: An MC based on vertex insertions, and the correspondence between node array and vertex array; the offset is four.

We note that, except for the root, there is a one-to-one correspondence between nodes and vertices. Therefore, if both nodes and vertices are stored in arrays, we can sort the elements of the two arrays in such a way that the index of the central vertex of a node is equal to the index of the node, plus an offset. For instance, in Figure 6.15, the offset is four. Thus, we do not need to store the central vertex of a node explicitly, and $\mathcal{S}_{opt} = 0$.

The space required by an explicit data structure compressed in this way (disregarding vertex coordinates) is $a(3\texttt{int}) + n(2\texttt{int} + 2\texttt{real})$, which is bounded from above by $t(3.5\texttt{int} + 1\texttt{real})$.

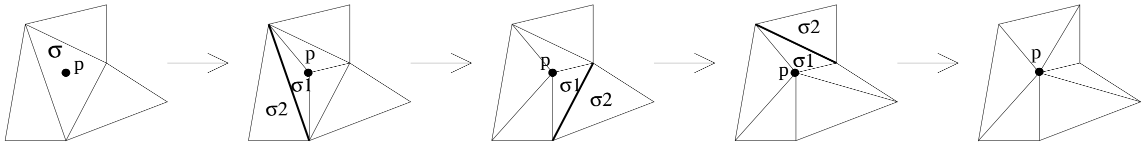


Figure 6.16: Transformation of $\Pi_1(C)$ into $\Pi_2(C)$ through triangle split and edge flips. At each flip, the pair σ_1, σ_2 of triangle sharing the flipped edge is indicated.

The saved space with respect to the general-purpose version is about $t(6.5\texttt{int})$, i.e., the compressed version requires about one third of the space.

In the case of a sequential data structure, the space required by the compressed version is $n(2\texttt{int} + 1\texttt{real}) + a(1\texttt{int}) \leq t(3\texttt{int} + 0.5\texttt{real})$, i.e, about half of the size of the general-purpose one.

The DAG traversal state increases its storage requirements because, in order to maintain Delaunay triangulation through vertex insertion and removal, we must store the current triangulation in a data structure with adjacency information even if such information is not required in the output. Efficient point location requires an additional search structure [BT93].

We have implemented an explicit data structure based on this compression scheme and the basic query algorithm working on it. Experiments are reported in [DFMP98c] and in Section 6.4.4. This compression scheme is used in [SZL92] with a data structure that resembles the sequential one for MCs, but it does not store dependency links; therefore, query algorithms perform involved tests to reconstruct such links at running time.

6.4.2.2 A Structure Based on Edge Flips

This compression scheme can encode any MC where nodes represent vertex insertions. It is efficient for MCs having a bounded width, and where the bounding constant b is small.

The basic idea is that, for each MC-node C, mesh $\Pi_2(C)$ (i.e., a triangulation not containing vertex p) can be transformed into $\Pi_2(C)$ (i.e., one containing p) by first inserting p in a greedy way, and then performing a sequence of edge flips. This process is illustrated in Figure 6.16.

First, p is inserted into $\Pi_1(C)$ by splitting one of its triangles, that we call the *reference triangle* for p. This creates three triangles incident in p (two triangles if p is on the boundary of the domain), which may or may not belong to $\Pi_2(C)$, and, in general, leaves in the mesh some triangles of $\Pi_1(C)$.

Starting from such initial configuration, a sequence of edge flips is performed. Each edge flip affects a triangle σ_1 incident in p and the triangle σ_2 adjacent to σ_1 along the edge opposite to p, and replaces σ_1 and σ_2 with two new triangles incident at p. At the end, p has a fan of incident triangles which form $\Pi_2(C)$. Note that, in $\mathbb{E}^3$, intermediate triangulations of the process are always consistent topologically, but may not be consistent geometrically (i.e., there may be self-intersections), but this is not relevant since the final triangulation (i.e., $\Pi_2(C)$) is consistent both topologically and geometrically.

Based on the above idea, the modification pattern represented in a node C is fully described by the new vertex p, a reference triangle $\sigma \in \Pi_1(C)$, and a sequence of edge flips.

Edge flips are represented by numerical codes. Let us consider an intermediate situation where a flip replaces the j-th incident triangle of p in a radial order around p (e.g., in counterclockwise order starting from the topmost one): then we use the number j to code the flip. The code of the first flip is in the range $0 \ldots 2$ since, at the beginning, there are only three triangles incident at p. Since each edge flip increases the number of these triangles by one unit, hence the code for the j-th flip is in $0 \ldots j+1$. The total number of edge flips for a vertex p is $k-3$, where k is the number of triangles (incident in p) in $\Pi_2(C)$. Since the MC has a bounded width, $k \leq b$, and thus the flip sequence consists of at most $b-3$ integers, where the j-th integer is in the range $0 \ldots j+1$. Therefore, the sequence of flips can be packed in a *flip code* of $\sum_{j=1}^{b-3}(\log_2(j+2)) = \sum_{i=3}^{b-1}(\log_2(i)) = \log_2((b-1)!) - 1$ bits.

The reference triangle σ for p belongs to $\Pi_2(C')$ for some parent C' of C. Thus, to uniquely define σ, it is sufficient to give a reference to C' and an integer number identifying one of the k triangles of $\Pi_2(C')$ according to a radial (e.g., counterclockwise) order around the central vertex of C'. Conventionally, we organize the data structure in such a way that parent C' is the first parent (i.e., the source of the first arc) stored for C; thus, there is no need to encode it explicitly. Since $k \leq b$, the number identifying σ is in the range $0 \ldots b-1$. We can pack the flip code and the index of σ together in $\log_2(b!) - 1$ bits. The central vertex of C does not need to be stored explicitly since we can sort the vertex and the node array consistently, as in Section 6.4.2.1.

This compression scheme can be applied to the explicit and to the sequential data structures, as explained in Section 6.4.1. Here, $\mathcal{S}_{opt} = \log_2(b!) - 1$ bits. The overall requirements of this compression scheme (except for vertex coordinates) are $a(3\mathtt{int}) + n(2\mathtt{int} + 2\mathtt{real} + \log_2(b!) - 1)$ for the explicit structure, and $n(2\mathtt{int} + 1\mathtt{real} + \log_2(b!) - 1) + a(1\mathtt{int})$ for the sequential one. Such quantities are bounded from above by $t(4\mathtt{int} + 1\mathtt{real} + \log_2(b!))$ and $t(2\mathtt{int} + 0.5\mathtt{real} + 0.5\log_2(b!))$, respectively. The space saved with respect to the corresponding general-purpose versions is $t(6\mathtt{int} - \log_2(b!))$ and $t(4\mathtt{int} + 0.5\mathtt{real} - 0.5\log_2(b!))$. Assuming that the value of b is small, we save need just one third and two thirds of the space used by the corresponding general-purpose structure, respectively.

A compressed explicit structure based on edge flips is under implementation.

6.4.3 Extensions to Higher Dimensions

Unlike general data structures described in Section 6.3, compressed structures just described do not generalize easily to higher dimensions.

The only exception is the structure for MCs based on vertex insertions in a Delaunay triangulation. Such compression scheme directly extends to d-dimensional simplicial Multi-Complexes in $\mathbb{E}^d$ based on vertex insertions in a Delaunay complex. The stored information is the same; what changes is the number of coordinates for each vertex and the algorithm used to incrementally update the current Delaunay complex. On-line algorithms for maintaining d-dimensional Delaunay complexes have been proposed in [Bow81, Wat81, Joe91, Raj91].

Extending the compression scheme based on edge flips to higher dimensions is a non-trivial problem. Edelsbrunner and Shah [ES96] showed that insertion of a point in a Delaunay simplicial complex reduces to a sequence of flips of $(d-1)$-faces. This result could suggest that a coding based on flips may be possible at least for d-dimensional Delaunay MCs built through incremental refinement. However, in d-dimensions it is not clear how the $(d-1)$-faces incident at a vertex could be sorted in such a way to allow the definition of a compact flip code. Moreover, it is difficult to guarantee a bounded degree of vertices while constructing a simplicial mesh through refinement, and, in any case, the number of flips is not guaranteed to be linear in the degree of the inserted vertex.

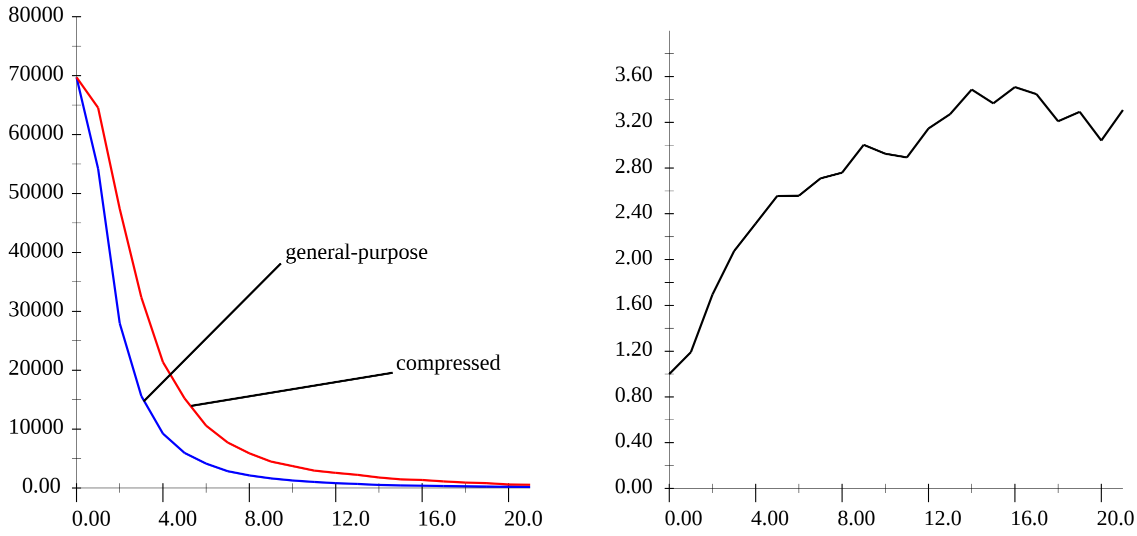
6.4.4 Comparison of Compressed and General Structures

We have performed experiments to compare the quality of triangle meshes resulting from querying an MC encoded through a general-purpose explicit data structure and from the same MC encoded through a compressed explicit data structure.

Here, we report results obtained from the explicit data structure (see Section 6.3.1) and from its compressed version based on Delaunay vertex insertion (see Section 6.4.2.1) on a terrain MC based on Delaunay triangulations. An compressed explicit structure based on edge flips is under implementation. Note that query algorithms provide the same meshes for the same input parameters with all compressed data structures, but the time needed to obtain them varies, depending on the amount of work needed for reconstructing triangles with the specific structure.

We present experimental results on an MC representing a terrain (two-dimensional scalar field), in the area of Mount Marcy, NY, USA (courtesy U.S. Geological Survey). The original data are on a grid of 200×200 points. The MC contains 35,160 nodes, 103,093 arcs, 175,922 triangles and 35,162 vertices. The sizes of the basic explicit structure and of the compressed one are about 6.5 and 2.5Mb, respectively.

We have encoded such MC with both data structures and performed global queries at uniform



(a) **(b)**

Figure 6.17: Results of global queries according to uniform error thresholds. The x-axis corresponds to the threshold values, expressed as a percentage of the height range in the terrain data. (a) Number of triangles in the output; the general-purpose structure is in blue, the compressed one in red. (b) Ratio of the number of output triangles with the two data structures.

	General-purpose		Compressed	
error	time (msecs)	triangles	time (msecs)	triangles
20%	0.68	176	69.3	566
5%	5.41	1,281	319.5	3,726
1%	39.05	8,438	1,543.9	19,756
0.2%	252.05	37,073	3,554.5	51,900
0.0%	462.32	69,718	4,820.6	69,718

Table 6.3: Results of global queries at uniform accuracy for the explicit data structure in the general-purpose and in the compressed version. The error threshold is expressed as a percentage of the height range in the terrain.

accuracy, according to resolution filters induced by an increasing sequence of constant error thresholds functions. Figure 6.17 compares the output sizes; times (obtained on an SGI Indigo2 workstation (R10000, 195 Mhz, 128 Mb RAM)) are reported in Table 6.3.

With the compressed data structure, query times increase of an average factor of ten, and the sizes of the resulting meshes are about doubled with respect to the results obtained with the explicit structure. The increase in the sizes of the meshes is due to the impossibility of testing the accuracy of each triangle individually, as discussed in Section 6.4.1. The increase in execution times is due to the on-line computation of a Delaunay triangulation, and to the fact that a larger portion of the DAG is traversed.

6.5 A Data Structure for Secondary Storage

Data structures and algorithms for the management of MCs on secondary storage can be designed for solving two different problems (which may also exist in combination):

1. given a large MC on disk, load just a part of it, and then apply query algorithms in main memory to the loaded part;
2. define and apply query algorithms having a direct access to a large MC resident on disk.

For Problem 1, strategies have to be developed to split an MC into parts which are still MCs and which can be loaded separately, as well as algorithms for selecting the parts of the MC which must be loaded for answering a given query.

In Problem 2, a query must be answered by directly manipulating an MC stored on disk. Data structures must be developed for representing an MC on disk, as well as algorithms that implement kernel operations on such data structures. Here, the granularity of information items that are moved from secondary to primary storage is much finer than in Problem 1: not full MCs but individual elements of an MC (e.g., vertices, cells, nodes, arcs).

In this section, we propose a solution to Problem 1. We have not addressed Problem 2 yet, since it seems more difficult.

For Problem 1, we propose a strategy that allows building and querying a modular set of MCs representing different portions of the same geometric object, where two or more MCs can be combined together into a larger MC covering a larger portion of the object. The individual MCs are contained in separate files and combined together on loading. Each application program loads just those MCs that it needs. The approach applies to two-dimensional simplicial MCs in $\mathbb{E}^2$ (such of those used to represent the domain of a two-dimensional field).

Before describing the proposed technique, in Section 6.5.1 we introduce some auxiliary concepts and definitions. In Section 6.5.2 we describe how individual MCs are organized and the information maintained which permit to merge them at run-time. In Section 6.5.3, we describe the management of a query which involves loading and merging one or more sub-MCs at run-time. The construction of a set of mergeable MCs will be addressed in Section 8.2.1.2. The approach described here, as well as the construction algorithm described in Section 8.2.1.2, have been completely developed and are currently under implementation.

6.5.1 Matching MCs

In this subsection, we define when a pair of MCs *match at their boundaries*; the matching condition is a prerequirement for merging two MCs.

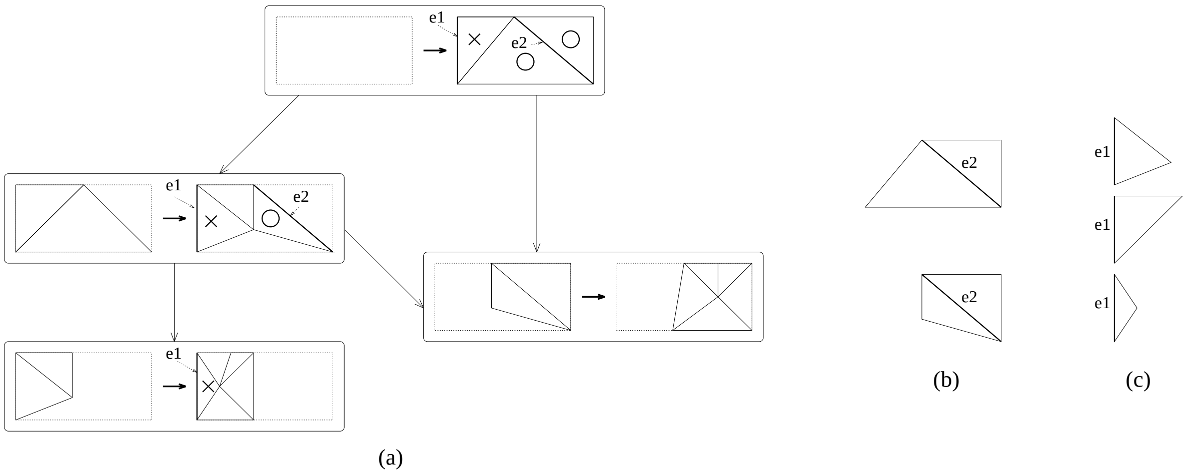


Figure 6.18: (a) A two-dimensional MC; e_1 is a boundary edge (its incident triangles are marked with a cross); e_2 is not a boundary edge (its incident triangles are marked with a circle); (b) Two pairs of mutually co-adjacent triangles incident at e_2; (c) All triangles incident at e_1 are mutually counter-adjacent.

We define the *boundary MC* of $\mathcal{M}$ as a one-dimensional MC providing a multiresolution representation of the boundary of the global combined cell complexes of $\mathcal{M}$.

Definition 6.1 *Let $\mathcal{M}$ be a two-dimensional simplicial MC. An edge e of $\mathcal{M}$ is a* global boundary edge *of $\mathcal{M}$ if, denoted with σ any triangle of $\mathcal{M}$ incident at e, σ has no co-adjacent triangles along e.*

The above definition identifies edges lying on the boundary of some global combined cell complex of $\mathcal{M}$, i.e., those edges e such that the triangles incident to e lie all on the same side with respect to e (see Figure 6.18).

The *boundary MC* of $\mathcal{M}$ is a graph obtained as follows:

1. Consider the set $\mathcal{C}$ of all the nodes C of $\mathcal{M}$, such that the set of global boundary edges of $\Pi_1(C)$ is different from the set of global boundary edges of $\Pi_2(C)$ (i.e., the nodes C, such that the application of the modification pattern $(\Pi_1(C), \Pi_2(C))$ changes the the boundary of the carrier. In Figure 6.19, set $\mathcal{C}$ contains nodes 0, 1 and 4; in Figure 6.20, set $\mathcal{C}$ contains nodes $0'$, $2'$ and $3'$.

2. For each $C \in \mathcal{C}$, delete all the cells of $\Pi_1(C)$ and $\Pi_2(C)$ with the exception of the global boundary edges and of their endpoints; let C' be the resulting one-dimensional modification pattern.

3. C' may not be minimal. If so, split it into minimal modification patterns. In Figure 6.19, this applies to the one-dimensional pattern corresponding to node 4.

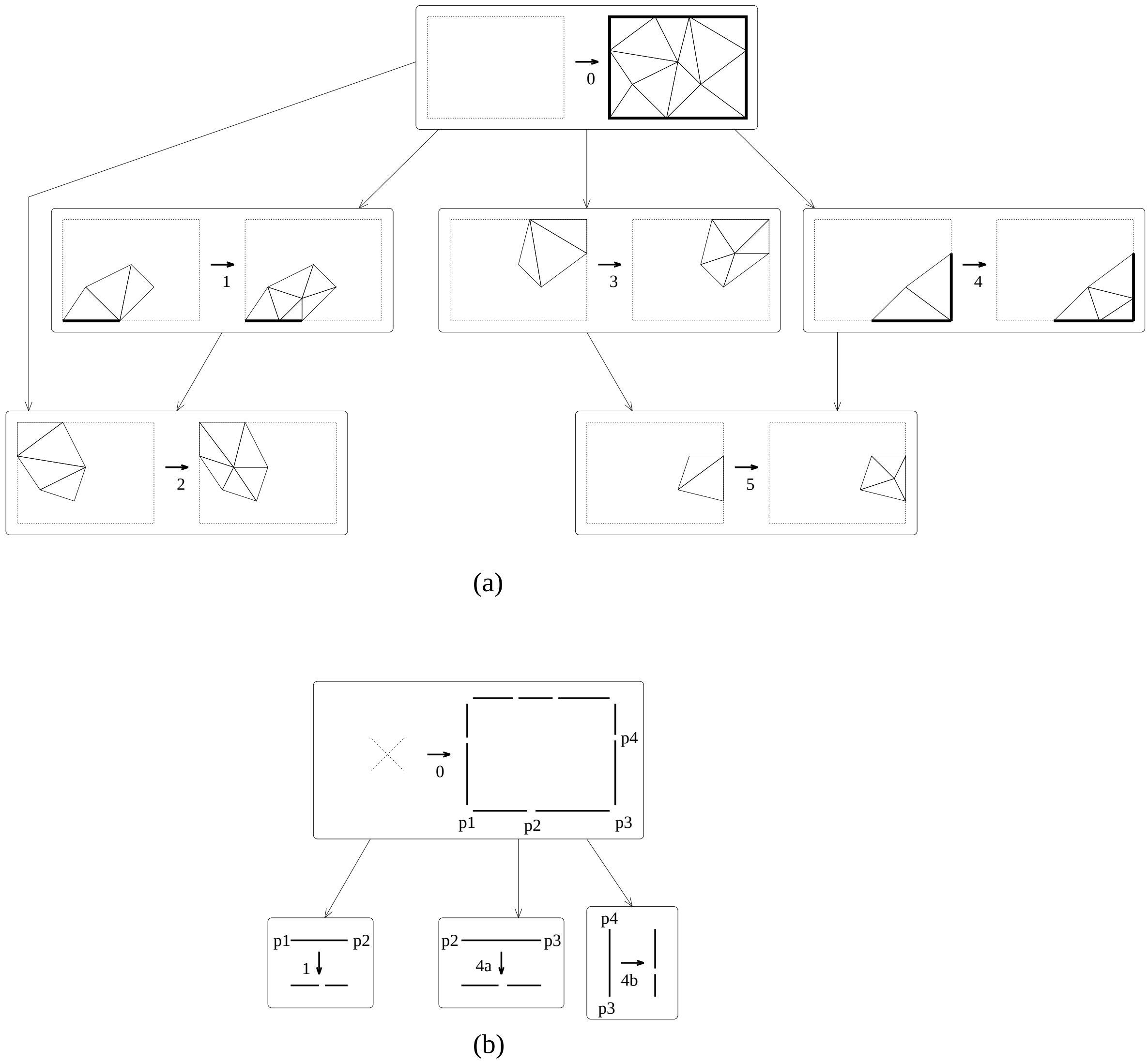


Figure 6.19: (a) A two-dimensional MC, and (b) its boundary MC.

4. Build a DAG from the resulting set of one-dimensional modification patterns, by creating an arc (C'_1, C'_2) for each pair of one-dimensional nodes such that C'_2 directly depends on C'_1.

Provided that the set of one-dimensional modification patterns resulting from Step 3 is non-redundant, the boundary MC is a one-dimensional MC (i.e., the DAG has no cycles). If we build the individual MCs with the method described in Section 8.2.1.2, this is always the case. Otherwise, in Step 4 we create an arc (C'_1, C'_2) between a pair of nodes such that C'_2 directly depends on C'_1 *only if*, denoted with C_i the node of $\mathcal{M}$ from which C'_i has been obtained (for $i = 1, 2$), we have that C_1 is an ancestor of C_2 in $\mathcal{M}$, and we consider two occurrences of the

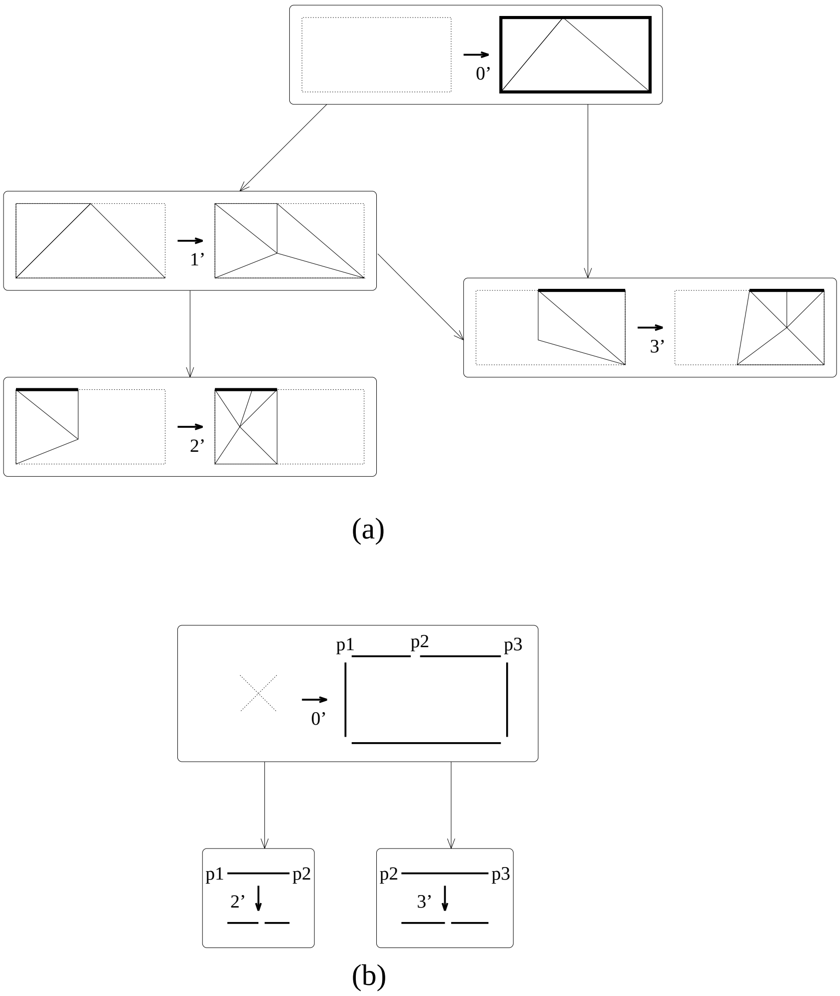


Figure 6.20: (a) A two-dimensional MC matching with the one of Figure 6.19, and (b) its boundary MC.

same edge in two different one-dimensional patterns as two distinct edges.

Given two boundary MCs $\mathcal{M}'_1$ and $\mathcal{M}'_2$ of two different two-dimensional Multi-Complexes, we say that a node C'_1 of $\mathcal{M}'_1$ and a node C'_2 of $\mathcal{M}'_2$ are *congruent* if they are both the root, or they are labelled with the same modification pattern.

We say that two two-dimensional MCs $\mathcal{M}_1$ and $\mathcal{M}_2$ *match at their boundaries* if, denoted with $\mathcal{M}'_1$ and $\mathcal{M}'_2$ the boundaries MCs of $\mathcal{M}_1$ and $\mathcal{M}_2$, the following conditions are verified:

1. for every edge e of $\mathcal{M}'_i$, either e does not appear in $\mathcal{M}_{3-i}$, or e appears in $\mathcal{M}'_{3-i}$ as well;
2. for every edge e common to $\mathcal{M}'_1$ and $\mathcal{M}'_2$, denoted with C'_i the node of $\mathcal{M}'_i$ such that $e \in \Pi_2(C'_i)$ (for $i = 1, 2$), C'_1 and C'_2 are congruent;

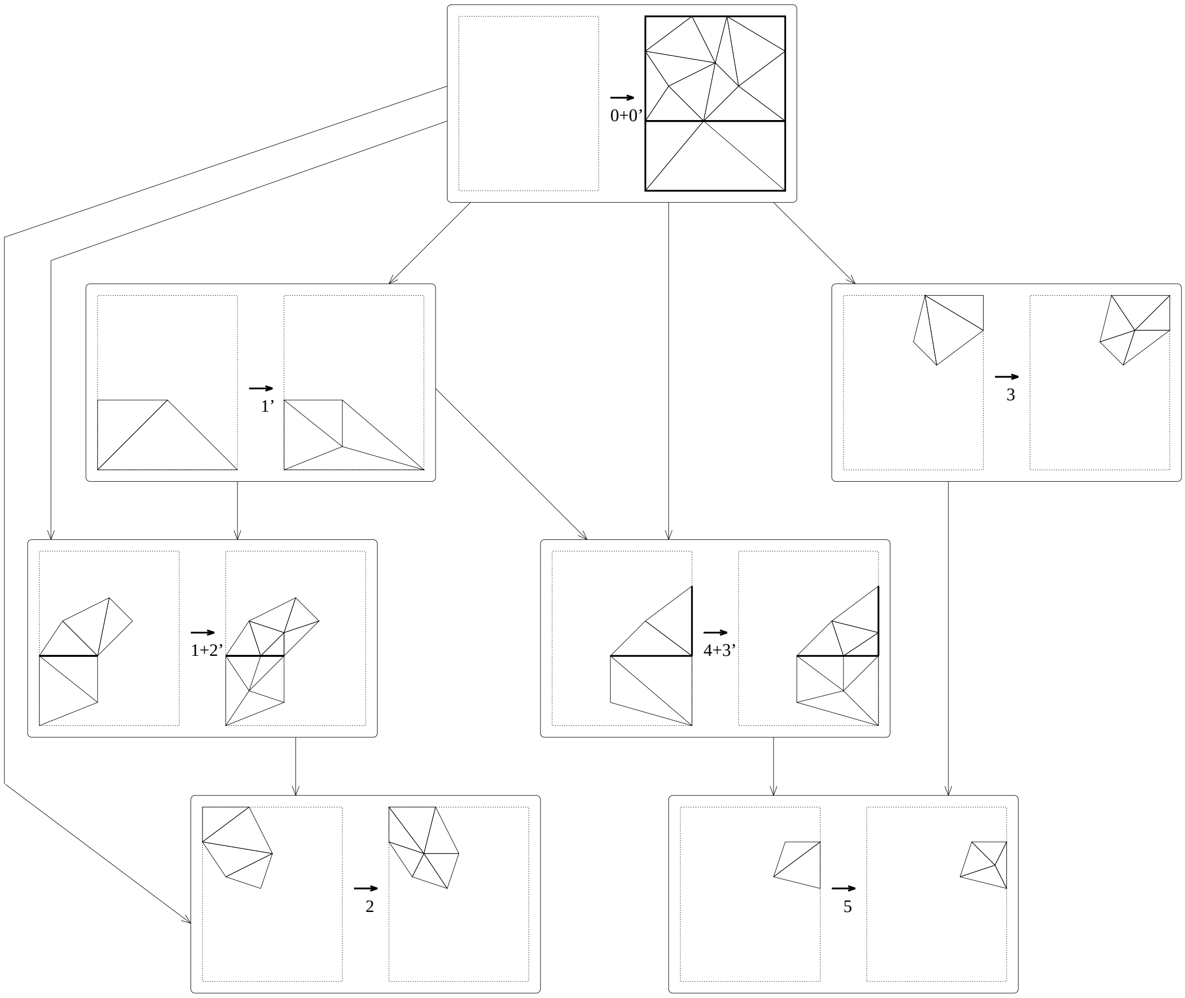


Figure 6.21: The two-dimensional MC resulting from merging the two matching MCs of Figures 6.19 and 6.20.

3. for every edge e common to $\mathcal{M}'_1$ and $\mathcal{M}'_2$, denoted with C'_i the node of $\mathcal{M}'_i$ such that $e \in \Pi_1(C'_i)$ (for $i = 1, 2$), either C'_1 and C'_2 do not exist, or they are congruent;

4. for every pair of congruent nodes C'_1 of $\mathcal{M}'_1$, and C'_2 of $\mathcal{M}'_2$, their parents and children which appear also in the opposite boundary MC are congruent;

5. for every pair of nodes C'_1 of $\mathcal{M}'_1$, and C'_2 of $\mathcal{M}'_2$, such that C'_1 is not congruent with any node of $\mathcal{M}'_2$, and C'_2 is not congruent with any node of $\mathcal{M}'_1$, C'_1 and C'_2 are not in conflict.

6. for every common edge of $\mathcal{M}'_1$ and $\mathcal{M}'_2$, for every pair of triangles σ_1 of $\mathcal{M}_1$ and σ_2 of $\mathcal{M}_2$ incident at e, the interiors of σ_1 and σ_2 are disjoint.

Conditions 2, 3 and 4 ensure that the common boundary portions of $\mathcal{M}_1$ and $\mathcal{M}_2$ evolve in the same way while increasing resolution. Condition 5 ensures that non-common boundary portions do not "disturb" each other. Condition 6 ensures that the carriers of combined cell complexes of $\mathcal{M}_1$ and $\mathcal{M}_2$ lie on opposite sides with respect to the common boundary edges.

Given two MCs $\mathcal{M}_1$ and $\mathcal{M}_2$ that match at their boundaries, we can construct their *union* in this way:

1. for every common global boundary edge e of $\mathcal{M}_1$ and $\mathcal{M}_2$, let C_i the node nearest to the root in $\mathcal{M}_i$ such that e appears in $\Pi_2(C_i)$ (for $i = 1, 2$);
2. we merge C_1 and C_2 into a single node which inherits incoming and outgoing arcs from both; in Figure 6.21, the pairs of merged nodes are 0 and $0'$, 1 and $2'$, 4 and $3'$;
3. possible arcs with the same source and destination created in the merge process are replaced with a single arc (for instance, in Figure 6.21 merged nodes 4 and $3'$ have both an arc from the root).

The result of the union is still an MC.

In the following subsections, we define a data structure for maintaining a collection of matching MCs on disk, and a technique for applying local query algorithms on such a collection.

6.5.2 Maintaining a Collection of Matching MCs

Let $\mathcal{U}$ be a collection of MCs such that every pair of MCs C_1 and C_2 in $\mathcal{U}$ are matching at their boundaries. Every boundary edge e is shared by at most two MCs of the collection. We assume that each MC in $\mathcal{U}$ is identified by a unique index. We store:

1. *each individual MC* of $\mathcal{U}$ on disk in a separate file;
2. *boundary information* which describe the shared boundary edges of each pair of MCs in $\mathcal{U}$ and permits merging the MCs of $\mathcal{U}$ pairwise;
3. a *main MC*, permanently resident in memory, which is used as a spatial index to decide which individual MCs must be loaded and merged to solve a given query.

Boundary information is organized as follows:

- we store the set of all boundary edges that are shared by exactly two MCs of $\mathcal{U}$ into a search structure (e.g., a balanced binary tree) indexed on the pair (i_1, i_2) of the indexes of the two MCs sharing the edge;

- for each boundary edge e in the binary tree we store two references to the two nodes C_1 and C_2 of the two MCs sharing e, such that, for $i = 1, 2$, C_i is the nearest node to the root containing e in $\Pi_2(C_i)$; these are the two nodes which must be glued at e in order to construct the union of the two MCs.

It is possible to reduce the size of the search structure by using the convention that, if more edges have the same pair of MC indexes and the same pair of node indexes, then only one of them is stored.

The *main MC* is an additional MC $\mathcal{M}_0$ whose front cell complex Σ_0 is the union of the triangulations at the root of each individual MC of set $\mathcal{U}$.

Direct dependency links are defined between the nodes of $\mathcal{M}_0$ (in main memory) and the roots of the individual MCs of $\mathcal{U}$ (on disk). We say that the root $C_{(0,i)}$ of $\mathcal{M}_i$ *directly depends* on those nodes C of $\mathcal{M}_0$, such that C contains some triangle of $\Pi_2(C_{(0,i)})$. This duplicates the triangles γ of $\mathcal{M}_0$, which are in Σ_0, in such a way that a copy of γ (the one belonging to $\mathcal{M}_0$) is in memory, and the other copy (the one belonging to $\mathcal{M}_i$) is on disk. This is illustrated in Figure 6.22.

In a typical situation, the main MC $\mathcal{M}_0$ is in memory along with a proper (small) subset $\mathcal{U}'$ of the individual MCs of $\mathcal{U}$. The MCs of $\mathcal{U}'$ have been pairwise merged and have been connected to the main MC $\mathcal{M}_0$.

For storing $\mathcal{M}_0$ and $\mathcal{U}'$, we adopt a data structure based on the explicit structure described in Section 6.3.1. The main MC $\mathcal{M}_0$ and each MC $\mathcal{M}_i \in \mathcal{U}'$ are represented in an individual explicit structure; a top-level structure encapsulates and connects such individual data structures "simulating" a global MC.

The top-level structure is a *shallow MC*, representing the union of the MCs in $\mathcal{U}'$ under the "cap" of $\mathcal{M}_0$. Each node [arc] of the shallow MC corresponds either to a node [arc] of $\mathcal{M}_0$, or to a set of nodes [arcs] belonging to different individual MCs of $\mathcal{U}'$ which have been merged.

The data structure for the shallow MC encodes just the DAG (i.e., nodes and arcs), while cells are obtained by accessing the single MCs of the set. The explicit data structure for a shallow MC is organized as an ordinary explicit data structure, except that:

- No vertices and no triangles are maintained.
- Every arc A, instead of storing the set of triangles labelling it, stores references to the arcs of the individual MCs that are merged into A. Each reference is made up of two indexes: the index of an MC, and the index of an arc inside it.
- Triangles labelling an arc A of the shallow MC are obtained indirectly by accessing the set of arcs contributing to it, and accessing the triangles associated with such arcs in the corresponding individual MC.

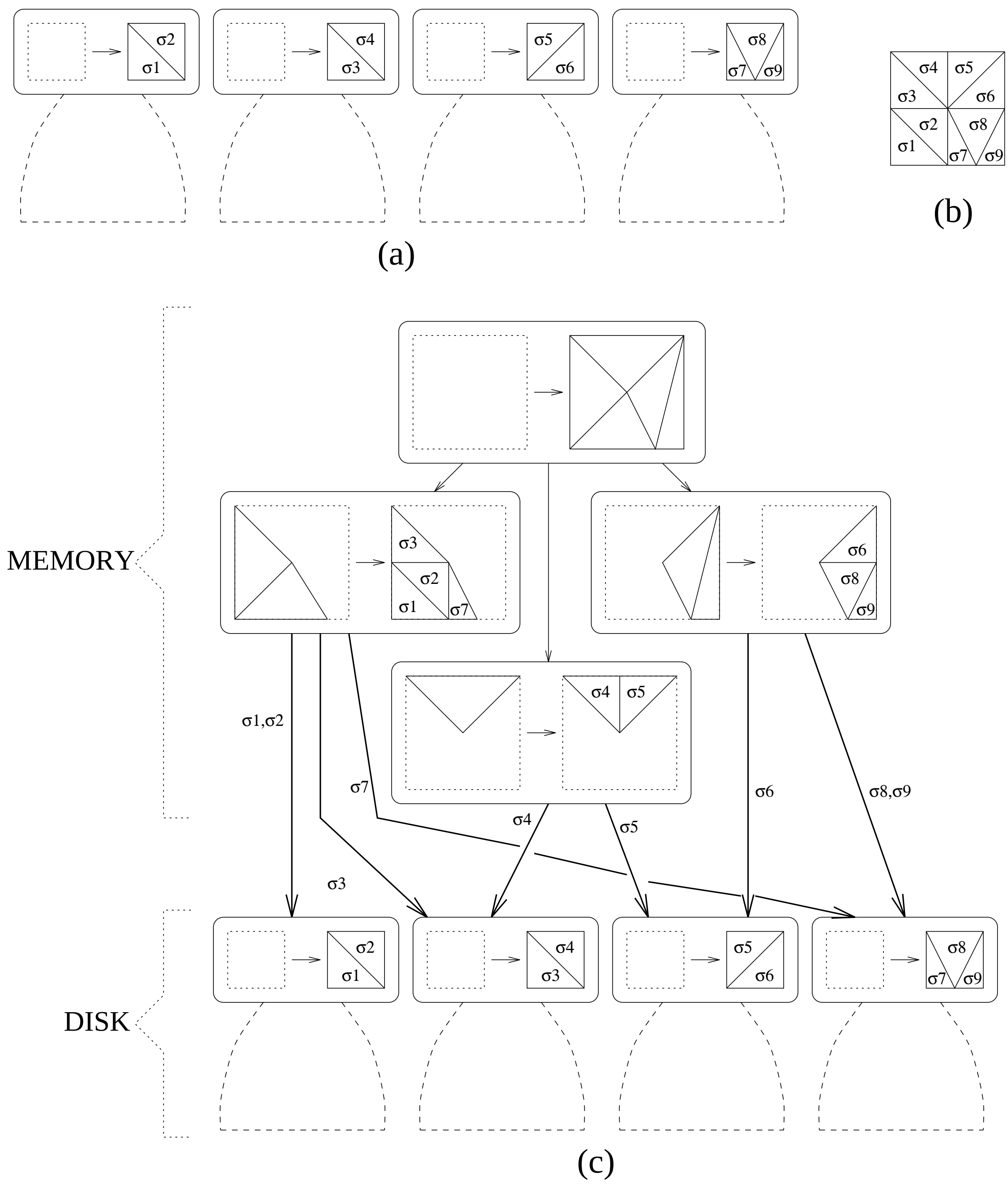


Figure 6.22: (a) A set of matching MCs (only their roots are shown); (b) the triangulation formed by the union of their roots; (c) the top-level MC; thick arcs represent the connection of the top-level MC with the individual MCs, and they go from main memory to disk.

6.5.3 Query Algorithms on a Collection of Matching MCs

The collection of matching MCs supports *local queries*, which are answered by loading into memory just the individual MCs revelant for the given focus set. Global queries do not make sense in this context because they would need to load all the MCs of the given collection $\mathcal{U}$.

At the beginning, the shallow MC is inizialized as a structure which encapsulates just the main MC $\mathcal{M}_0$. We assume that we can hold at most m individual MCs of $\mathcal{U}$ in main memory, where m is sufficient for managing all local queries occurring within our application. Depending on the query, more MCs of $\mathcal{U}$ are loaded and added to the current shallow MC. At the end of each query, the set $\mathcal{U}' \subset \mathcal{U}$ of MCs loaded can be discarded (or, better, moved to a buffer from which MCs can be resumed, if needed again in the next query, or discarded, if not needed and the buffer is full). The main MC $\mathcal{M}_0$ resides permanently in main memory (i.e., within the shallow MC), and is used as a spatial index to determine which individual MCs must be loaded for a given focus set.

Given a query, we start the local algorithm from the root of the shallow MC (which is the same as the root of $\mathcal{M}_0$). The first iterations of the algorithm will be within $\mathcal{M}_0$, and, thus, in memory. When we find that a node C_i must be included in the current pseudo-sub-MC, where C_i is the root of an MC $\mathcal{M}_i \in \mathcal{U}$ which has not been loaded yet, then we access the disk to load $\mathcal{M}_i$, merge it with the other MCs that have been loaded already, and continue with the query algorithm.

Now, we describe the algorithm used for merging a new MC $\mathcal{M}_i \in \mathcal{U}$ with the ones of the subset $\mathcal{U}' \subset \mathcal{U}$ already in memory. This operation reduces to connecting the nodes and arcs of $\mathcal{M}_i$ to the ones of the shallow MC.

From the balanced binary tree of boundary information, we retrieve all the boundary edges e corresponding to a pair of indexes (i_1, i_2) where i_1 is the index of $\mathcal{M}_i$ and i_2 is the index of an MC $\mathcal{M}_j \in \mathcal{U}'$ (or vice versa). At each of such edges, we find a pair of nodes to merge: one, C_i, belonging to $\mathcal{M}_i$ and the other one, C_j to $\mathcal{M}_j$. Node C_j is already present in the shallow MC; then, we attach C_i to the same node of the shallow MC that contains C_j.

6.6 Summary

In this chapter, we have considered three fundamental approaches to the representation of a simplicial Multi-Complex, namely as a DAG, as a sequence, or as a collection of triangles connected through adjacency relations. We have provided data structures for each approach that apply to MC in arbitrary dimensions.

The explicit representation scheme (i.e., the one which describes an MC as a DAG) seems the most promising since it provides the best balance between space complexity of the structure, and efficiency and simplicity of implementation of query algorithms working on it. Therefore, it has been adopted in our implementations (see Chapter 8).

We have proposed compressed structures for two-dimensional MCs which describe triangles in an implicit way, based on some known property of the modification patterns involved in the model. In particular, we have considered MC where modification patterns represent vertex insertions. Apart from the compressed structure for Delaunay triangulations, extending compressed structures to higher dimensions seems difficult. However, the necessity of compact encoding is stronger in higher dimensions since the number of simplices, for a set of vertices of the same cardinality is larger; thus, finding efficient compression schemes is an important open problem.

The problem of managing MCs on secondary storage is relevant because multiresolution deals with huge amounts of data, likely leading to structures too big to fit in memory. A typical example is in GISs, where often terrain datasets have such a size that a single-resolution terrain model built from such sets would be too big to be loaded into memory. The same problem is even more likely for an MC, which represents the terrain at a wide range of resolutions including the maximum one.

We have proposed a method to organize and manage a collection of modular MCs which are stored in different files and can be loaded and glued on request, depending on the current needs of the user. This approach has been developed in two dimensions; finding similar techniques in arbitrary dimensions is an interesting open issue, also due to the fact that the size of a model increases with its dimension.

We have started a preliminary study in the direction of data structures and algorithms for managing an MC resident on disk; in this case, kernel primitives involve disk accesses.

Chapter 7

Construction of a Multiresolution Cell Complex

7.1 Introduction

In this chapter, we propose and discuss methods for the construction of Multi-Complexes.

Here, we assume that an MC is built *off-line*; this is a realistic assumption for all applications dealing with large data sets which need multiresolution management. On-line interactive construction corresponds to multiresolution shape editing; this problem arises in some specific applications, and is not addressed here.

The construction of a Multi-Complex can be conceptually subdivided in two distinct subproblems:

1. generation of a *initial cell complex* plus a *sequence of modification patterns* which incrementally refine such cell complex;
2. determination of the *partial order* defined by the dependency relation from the sequence of modification patterns and construction of the DAG.

Problem 1 involves the use of application-dependent criteria and procedures in order to build and modify a complex starting from the data that represent the available information for an object; it does not involve the Multi-Complex directly. Problem 2 consists of organizing the initial cell complex and the modification patterns into a DAG according to their mutual dependencies. Unlike problem 1, problem 2 is only concerned with the Multi-Complex and is independent of the application.

In this chapter, we first consider problem 2, i.e., the construction of an MC starting from a given cell complex and a given sequence of modification patterns. We define construction operations for performing such task, which are part of the application-independent interface of the MC data type.

The techniques used for solving problem 1, i.e., to generate the initial complex and the sequence of modification, have an impact on the quality of the resulting MC. They affect the shape parameters (width, height, growth) defined in Section 4.6 and the compression power of the MC, i.e., its capability of providing small cell complexes for a given error threshold. In the second part of this chapter we analyze the impact of different construction techniques on the quality of the resulting MC, and we show experimental results from implementations in the specific case-study of two-dimensional scalar fields, which extend those presented in [DFMP97a].

This chapter is organized as follows:

- In Section 7.2, we introduce the concept of *history* (i.e., an initial cell complex plus a sequence of modification patterns) which is the input for MC construction, and we state the necessary properties that a history must satisfy in order to be used for building an MC.
- In Section 7.3, we define algorithms to build an MC starting from a history.
- In Section 7.4, we provide a specification of operations for building an MC, including low-level primitives acting on the basic data structure that represent an MC and user-level construction operations.
- In Section 7.5, we discuss the influence of the set of modification patterns (deriving from the specific algorithm used for history generation) on the efficiency of an MC. We present both a theoretical analysis, and experimental results from a set of algorithms that we have implemented.

7.2 Histories and History Generators

In Section 3.2, we have reviewed iterative algorithms for the construction of approximated cell complexes. Such algorithms are a natural basis for the creation of an initial complex and a sequence of modifications. We recall the classification of iterative methods into:

- *top-down*, or *refinement* methods, which start from a coarse cell complex and iteratively apply refinement patterns;
- *bottom-up*, or *coarsening* methods, which start from a cell complex at the maximum resolution and iteratively apply coarsening patterns.

Since the Multi-Complex deals with manifold complexes, the initial cell complex must be a manifold and every subsequent modification performed by the algorithm must preserve this property. In the remainder of this chapter, we restrict our discussion to iterative methods that satisfy these requirements.

We introduce the following notion of *history* of an iterative process:

Definition 7.1 *A* history *is a pair* $(\Gamma_0, [C_1, C_2, \ldots, C_n])$*, where* Γ_0 *is a regular and manifold* k*-dimensional cell complex, and* $C_1, C_2, \ldots, C_n$ *is a sequence of* k*-dimensional modification patterns such that:*

- *for every* $1 \leq i \leq n$*,* C_i *is minimal;*
- *either all modification patterns* C_i *are refinement patterns, and* $\Pi_1(C_i) \neq \emptyset$*; or all modification patterns* C_i *are coarsening patterns, and* $\Pi_2(C_i) \neq \emptyset$*.*
- *sequence* $C_0, C_1, C_2, \ldots, C_n$ *is non-redundant, where* $C_0 = (\emptyset, \Gamma_0)$*;*

A history containing refinement patterns is called a *refinement history*; conversely, a history containing coarsening patterns is called a *coarsening history*. The sequence of coarsening patterns contained in a coarsening history can be reversed to obtain a sequence of refinement patterns. Thus, every history provides a sequence of refinement patterns of a cell complex.

Most iterative algorithms modify their current cell complexes by applying *minimal* modification patterns. Non-minimal modification patterns occur in coarsening methods that perform several independent and conflict-free modifications (e.g., vertex removals, edge collapses, triangle collapses) simultaneously [dBD95, EDRD+95, XV96, XESV97, GHJ+97]. However, such non-minimal modification patterns can be easily split into minimal ones. All existing methods generate *non-redundant* sequences because they are based on modification operators such as vertex insertion or deletion, edge or triangle collapse, vertex-pair collapse. Non-redundancy is easy to see for coarsening methods: in all cases listed above, the top-cells deleted at each step are those incident in at least one of the disappearing vertice(s); thus, they cannot be recreated later. The non-redundancy of a vertex insertion history can be seen by considering the reverse sequence of vertex deletions.

Thus, all existing methods can produce histories. An iterative algorithm capable of producing a history is called a *history generator*.

Note that, in Definition 7.1, the sequence of modification patterns in a history does not necessarily satisfy all the conditions required in Definition 4.12 (the definition of a Multi-Complex). Definition 4.12 requires, in addition, that sequence $C_1, C_2, \ldots, C_n$ is *valid* for Γ_0, and *conflict free*.

Definition 7.2 *A history* $(\Gamma_0, [C_1, C_2, \ldots, C_n])$ *is* valid *if* $C_1, C_2, \ldots, C_n$ *is a valid sequence of modification patterns for* Γ_0*.*

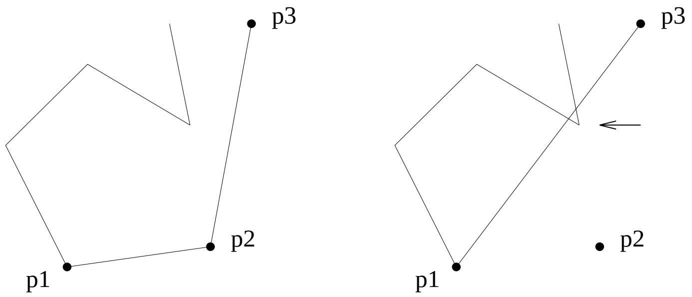


Figure 7.1: A non-valid modification of a one-dimensional complex, where the new 1-cell p_1p_3 intersects 1-cells that are not neighbors of the disappearing ones, and thus cannot be found in a local test. This situation generalizes to higher dimensions.

Definition 7.3 *A history* $(\Gamma_0, [C_1, C_2, \ldots, C_n])$ *is* conflict-free *if the set of modification patterns* $\{C_i \mid 1 \leq i \leq n\}$ *is conflict-free. Otherwise, the history is* conflicting.

Surprisingly enough, the algorithms proposed in the literature do not always perform *valid* sequences of modifications. Let us consider one step of the algorithm in which the current complex Γ is modified by replacing a subset Γ_{old} of its cells with a new set Γ_{new}. Almost all methods check that the cells Γ_{new} do not intersect the cells of $\Gamma \setminus \Gamma_{old}$ lying in the neighborhood of the updated area. In general, they do not perform a global intersection test with respect to the whole set $\Gamma \setminus \Gamma_{old}$, because of the high computational complexity involved. In practice, intersections are very unlikely, but still possible, as shown in the example of Figure 7.1. In this case, some of the intermediate sets of cells produced by the algorithm (including, possibly, the final one) may not be cell complexes. Among triangle mesh simplification algorithms, the *simplification envelope* method [CVM$^+$96] is the only method that guarantees the global consistency of the final result (but still not of the intermediate ones).

A valid history is not necessarily *conflict-free*. Figure 7.2 shows that, even starting from a valid history (i.e., a valid sequence of modifications), proper intersections may occur between cells involved in (independent) modifications applied in different parts for the object, when they are performed in a different order than the one of the original sequence.

If a DAG $\mathcal{M}$, structured as an MC, is built from a conflicting history, then conflicts between independent MC-nodes (i.e., nodes not connected through a directed path) cause that, for some sub-MC $\mathcal{M}'$ of $\mathcal{M}$, front($\mathcal{M}'$) is not a cell complex since some of its cells intersect in their interiors. From the point of view of the MC, conflicting histories cause the same problems, independently of whether the sequence of modification patterns is valid or not.

Valid and conflict-free histories can be guaranteed in the case of *conservative* complexes (see Section 4.4.2). Note that such models must have the same dimension of the space embedding them (like complexes used to approximate the domain of a field or of an explicit surface). In the case of cells complexes of lower dimensions embedded in a higher dimensional space, the use

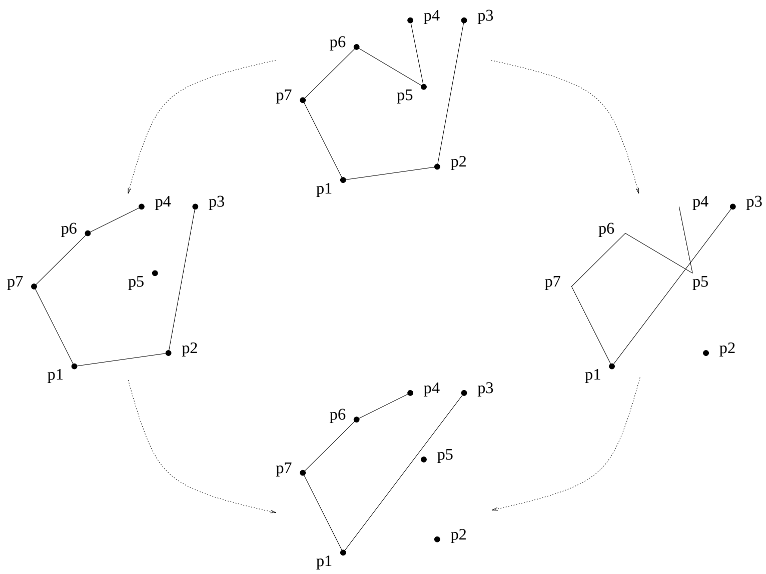


Figure 7.2: The sequence removing first p_5, then p_2 is valid, but not conflict-free, since the permuted sequence removing first p_2 and then p_5 is not valid. This situation generalizes to higher dimensions.

of histories which are not conflict-free may lead to self-intersections in the surfaces extracted from a multiresolution model. This problem is common to all existing multiresolution models dealing with surfaces (i.e., lower-dimensional entities in a higher-dimensional space).

In order to build an MC, it is necessary to start from a valid and conflict-free history. In the following sections, we will always assume to deal with valid and conflict-free histories.

7.3 From a History to a Multi-Complex

The MC built from a refinement history contains the given modification patterns in its nodes, plus an extra modification pattern corresponding to the creation of the initial cell complex. A coarsening history is the reverse of a refinement history, and the corresponding MC represents in its nodes the *reverse* of each given modification pattern, plus an extra modification pattern representing the reverse of the deletion of the final complex of the coarsening process.

Given a history $(\Gamma_0, [C_1, C_2, \ldots, C_n])$, complex Γ_0 is used to initialize a Multi-Complex consisting of just one node. Next, for each modification pattern C_i, for i increasing from 1 to n, the current MC is updated by adding a new node.

In the case of a refinement history, an MC is built top-down by starting from the root and adding a leaf at each step. After having processed the i-th modification pattern C_i of the sequence, the front cell complex of the current MC is the same as $\Gamma_0[C_1]\cdots[C_i]$. The next

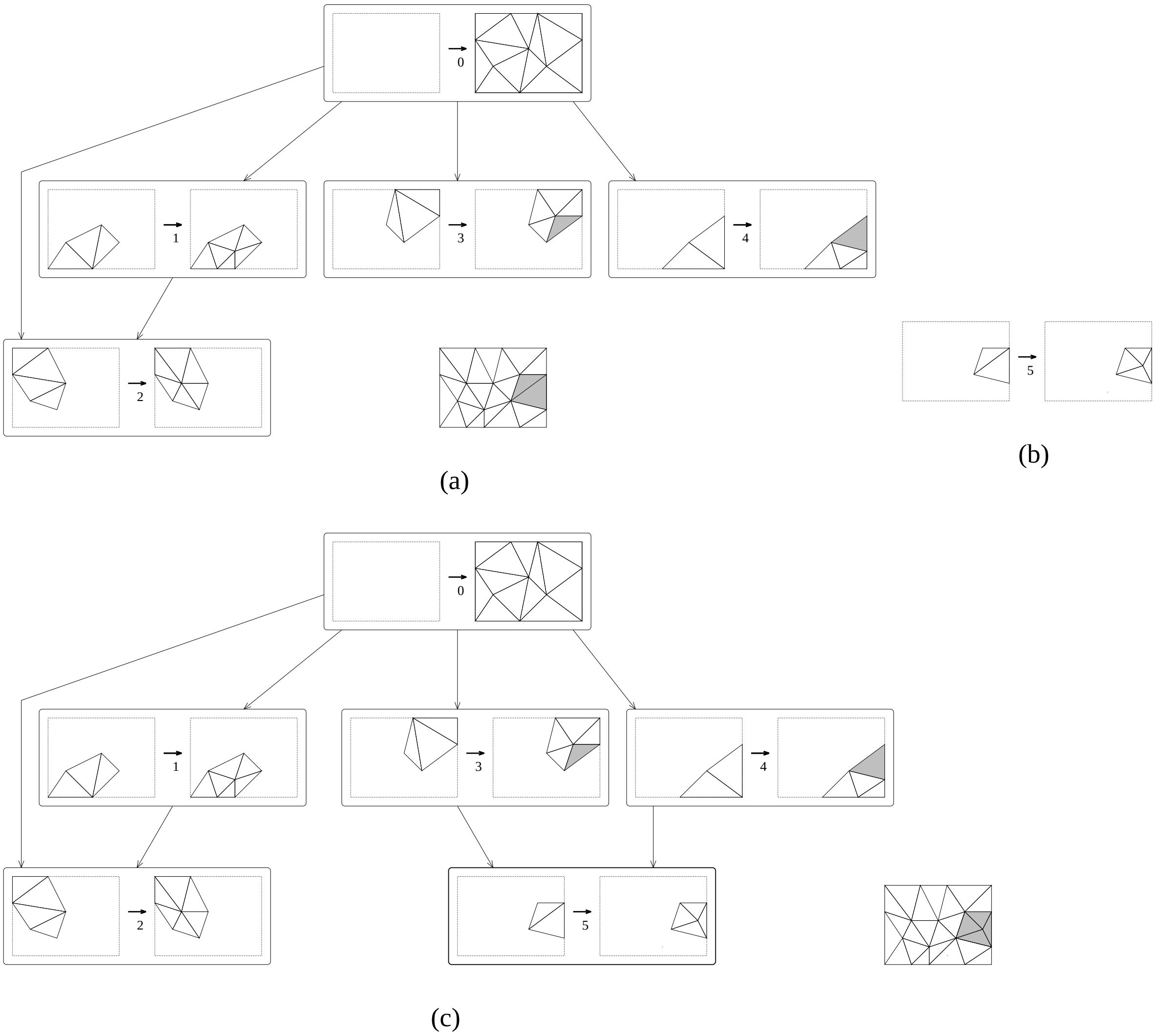
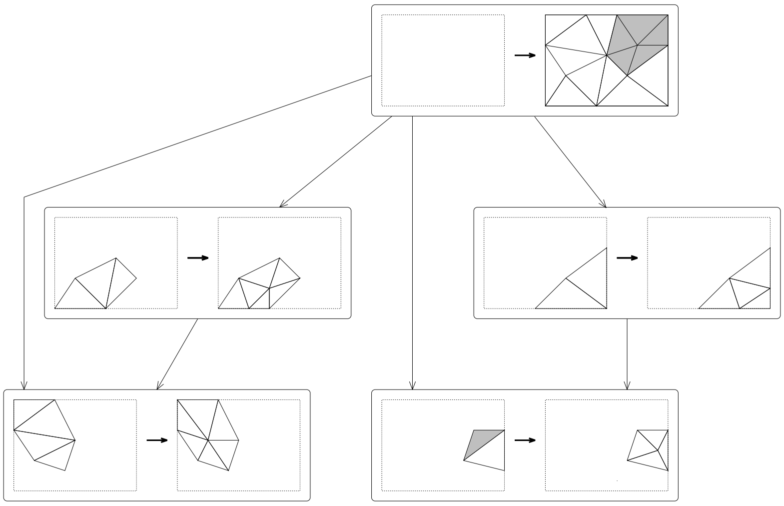


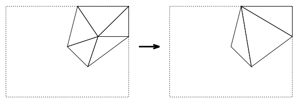
Figure 7.3: Adding a new node in the top-down construction of an MC: (a) the current MC and its front complex; (b) the next refinement pattern valid for the front cell complex; (c) the updated MC, where the new refinement pattern has been inserted as a leaf, and its front complex. In the DAG, the triangles shared by the pre-existing MC and the new modification pattern are shown shaded. In the front cell complex, the cells modified by adding the new node are shown shaded.

modification pattern C_{i+1} becomes a new MC-node that is inserted into the MC as a leaf by connecting it to the existing nodes that directly block C_{i+1}. The new front cell complex becomes $\Gamma_0[C_1]\cdots[C_i][C_{i+1}]$ (see Figure 7.3).

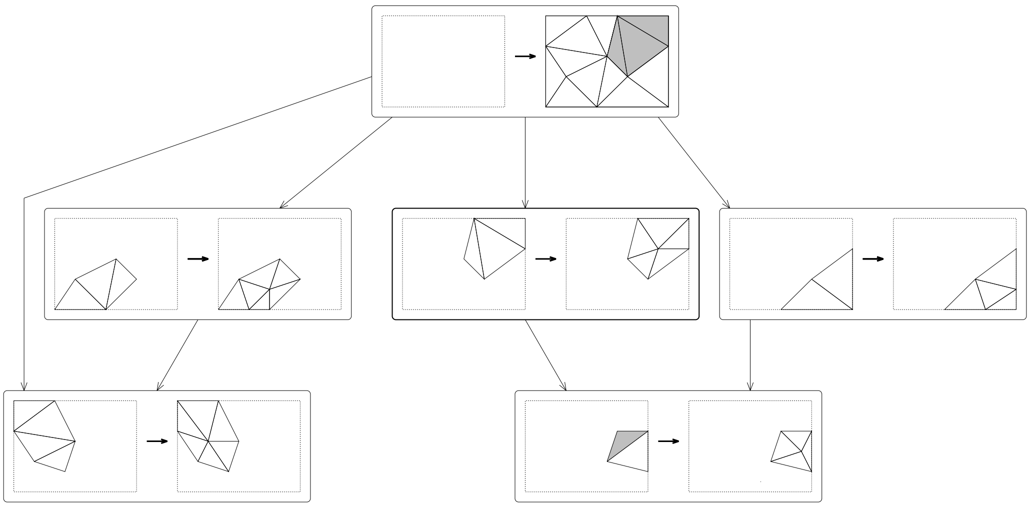
In the case of a coarsening history, an MC is built bottom-up by inserting at each step a new node as a child of the root. Let C_0' be the root. After having processed the i-th coarsening pattern C_i of the sequence, $\Pi_2(C_0')$ is the same as $\Gamma_0[C_1]\cdots[C_i]$. Given the next coarsening



(a)



(b)



(c)

Figure 7.4: Adding a new node in the bottom-up construction of an MC: (a) the current MC; (b) the next coarsening pattern valid for the cell complex at the root; (c) the updated MC, where the new coarsening pattern has been inserted as a child of the root and the complex at the root has been modified. In (a), the triangles shared by the pre-existing MC and the new modification pattern are shown shaded. In (c), the triangles that determine dependency links for the new node are shown shaded.

pattern C_{i+1} of the history, the reverse C'_{i+1} of C_{i+1} becomes a new MC-node that is inserted into the MC as a child of C'_0, while $\Pi_2(C'_0)$ itself is modified into $\Gamma_0[C_1]\cdots[C_{i+1}]$ by applying C_{i+1} to it (see Figure 7.4). Some children C'_j of the root (namely, those that directly depend on C'_{i+1} since $\Pi_1(C'_j) \cap \Pi_2(C'_{i+1})$, i.e., $\Pi_1(C'_j) \cap \Pi_1(C_{i+1})$ contains some top-cell) are attached as children to the new node; if C'_j does no longer depend on the root (because all the common top-cells of $\Pi_1(C'_j)$ and $\Gamma_0[C_1]\cdots[C_i]$ are in $\Pi_1(C_{i+1})$), then C'_j is detached from the root as well.

In the following subsections, we define algorithms for transforming a (valid and conflict-free) history into an MC.

7.3.1 Algorithms for MC Construction

In order to build an MC from a history, we need algorithms for initializing a Multi-Complex and for adding a new node to it either as a leaf or as a child of the root.

The following algorithm initializes a Multi-Complex with the initial complex contained in a history:

Algorithm 7.1
Input: a regular and manifold k-dimensional cell complex Γ_0.
Output: an MC $\mathcal{M}$.

> create $\mathcal{M}$ as an MC containing just one node, representing a modification pattern of the form $C_0 = (\emptyset, \Gamma_0)$, and no arcs.

It is easy to verify that the result of Algorithm 7.1 is an MC.

Now, we define the algorithms for adding a new node to an MC, given the next refinement or a coarsening modification pattern in a history, and the corresponding algorithms for building an MC from the entire history.

Algorithm 7.2
Input: a k-dimensional Multi-Complex $\mathcal{M}$, and a k-dimensional *refinement pattern* $C = (\Gamma_{old}, \Gamma_{new})$ (in order to ensure the correctness of the result, C must satisfy some conditions as specified next in Theorem 7.1).
Output: $\mathcal{M}$ after modifications.

1. add C as a new node to $\mathcal{M}$;
2. add a new arc (C_i, C) for each node C_i such that C directly depends on C_i.

The correctness of the algorithm is expressed in the following lemma:

Lemma 7.1 *If the input parameters $\mathcal{M}$ and $C = (\Gamma_{old}, \Gamma_{new})$ of Algorithm 7.2 satisfy the following conditions:*

1. *C is minimal, it is valid for the front cell complex Γ of $\mathcal{M}$, and Γ_{old} is not empty;*
2. *C is non-redundant with respect to the set of modification patterns that are the nodes of $\mathcal{M}$;*
3. *for each node C_i of $\mathcal{M}$ such that C does not depend on C_i, we have that C_i and C_i are not in conflict;*

then, the output $\mathcal{M}$ of Algorithm 7.2 is a Multi-Complex, and its front cell complex is $\Gamma[C]$.

Proof The hypothesis on C guarantees that, given any consistent sequence $\mathcal{S}$ of the nodes of $\mathcal{M}$, the sequence obtained by appending C to the end of $\mathcal{S}$ satisfies Conditions 1, 2, 3, and 4 of Definition 4.12. Moreover, the arcs entering C created by the algorithms capture all existing direct dependencies for C (note that C can only depend on nodes of $\mathcal{M}$; C cannot block nodes of $\mathcal{M}$ because of non-redundancy). Thus, the result is an MC. It is straight-forward to see that its front cell complex is $\Gamma[C]$. $\square$

MC construction through refinement from a cell complex Γ_0 and a sequence $C_1, \ldots, C_n$ of refinement patterns reduces to initializing an MC with Γ_0, and then iteratively refining it with C, for each i in increasing sequence from 1 to n.

Algorithm 7.3 (Top-down MC construction)
Input: a *refinement history* $(\Gamma_0, [C_1, C_2, \ldots, C_n])$.
Output: an MC $\mathcal{M}$.

- initialize $\mathcal{M}$ by applying Algorithm 7.1 with input parameter Γ_0;
- for every i from 1 to n, modify $\mathcal{M}$ by applying Algorithm 7.2 with input parameter C_i.

The correctness of the algorithm is expressed in the following theorem:

Theorem 7.1 *Given a valid and conflict-free refinement history $(\Gamma_0, [C_1, C_2, \ldots, C_n])$, Algorithm 7.3 outputs a Multi-Complex such that the nodes of $\mathcal{M}$ are the modification patterns of set $\{C_0, C_1, C_2, \ldots, C_n\}$, where $C_0 = (\emptyset, \Gamma_0)$.*

The proof of the theorem directly follows from the correctness of Algorithm 7.2 by noting that, at each step, Algorithm 7.2 is called with input arguments that satisfy Conditions 1, 2, and 3 of Lemma 7.1.

The following algorithm modifies an MC given a new coarsening pattern:

Algorithm 7.4
Input: a k-dimensional Multi-Complex $\mathcal{M}$, and a k-dimensional *coarsening pattern* $C = (\Gamma_{old}, \Gamma_{new})$ (in order to guarantee the correctness of the result, C must satisfy some conditions as specified next in Theorem 7.2).
Output: $\mathcal{M}$ after modifications.

1. set $\Pi_2(C'_0) = \Gamma[C] \equiv (\Gamma \setminus \Gamma_{old}) \cup \Gamma_{new}$;
2. add C' as a new node;
3. add a new arc (C', C'_i) for each node C'_i such that C'_i directly depends on C'.

The correctness of the algorithm is expressed in the following lemma:

Lemma 7.2 *If the input parameters $\mathcal{M}$ and $C = (\Gamma_{old}, \Gamma_{new})$ of Algorithm 7.2 satisfy the following conditions:*

1. *C is minimal, it is valid for the cell complex $\Gamma = \Pi_2(C'_0)$, where C'_0 denotes the root of $\mathcal{M}$, and Γ_{new} is not empty;*
2. *$C' = (\Gamma_{new}, \Gamma_{old})$ is non-redundant with respect to the set of modification patterns in the nodes of $\mathcal{M}$;*
3. *for each node C'_i of $\mathcal{M}$, and such that C'_i does not depend from C', C'_i and C' are not in conflict;*

then, the output $\mathcal{M}$ of Algorithm 7.2 is a Multi-Complex, and $\Pi_2(C'_0) = \Gamma[C]$.

Proof The hypothesis on C guarantees that, given any consistent sequence $\mathcal{S}$ of the nodes of $\mathcal{M}$, the sequence obtained by modifying $C'_0 \equiv (\emptyset, \Gamma_0)$ into $(\emptyset, \Gamma_0[C])$, and inserting C' just after C'_0 in $\mathcal{S}$ satisfies Conditions 1, 2, 3, and 4 of Definition 4.12. Moreover, the arcs created for C' capture all existing direct dependencies for C', and those deleted correspond to links of direct dependency that do not longer exist. Thus, the result is an MC. It is straight-forward to see that $\Pi_2(C'_0) = \Gamma[C]$. $\square$

MC Construction through coarsening from a cell complex Γ_0 and a sequence $C_1, \ldots, C_n$ of coarsening patterns reduces to initializing an MC with Γ_0, and then iteratively coarsening it with C, for each i in increasing sequence from 1 to n.

Algorithm 7.5 (Bottom-up MC construction)
Input: a *coarsening history* $(\Gamma_0, [C_1, C_2, \ldots, C_n])$.
Output: an MC $\mathcal{M}$.

- initialize $\mathcal{M}$ by applying Algorithm 7.1 with input parameter Γ_0;
- for every i from 1 to n, modify $\mathcal{M}$ by applying Algorithm 7.4 with input parameter C_i.

Theorem 7.2 *Given a valid and conflict-free coarsening history* $(\Gamma_0, [C_1, C_2, \ldots, C_n])$*, Algorithm 7.5 outputs a Multi-Complex such that the nodes of* $\mathcal{M}$ *are the modification patterns of set* $\{C'_0, C'_1, C'_2, \ldots, C'_n\}$*, where, for every* $1 \leq i \leq n$*,* C'_i *is the reverse of* C_i*, and* $C'_0 \equiv (\emptyset, \Gamma_0[C_1] \cdots [C_n])$*.*

The proof of the theorem directly follows from the correctness of Algorithm 7.4 by noting that, at each step, Algorithm 7.4 is called with input arguments that satisfy Conditions 1, 2, and 3 of Lemma 7.2.

It is easy to see that the following property holds:

Property 7.1 *The result of MC construction from a coarsening history* $(\Gamma_0, [C_1, C_2, \ldots, C_n])$ *is the same as the result of MC construction from a refinement history of the form* $(\Gamma_0[C_0] \cdots [C_n], [C'_1, C'_2, \ldots, C'_n])$*, where, for all* $0 < i \leq n$*,* $C'_i \equiv (\Pi_2(C_i), \Pi_1(C_i))$*.*

7.3.2 History Generation and MC Construction

The generation of a history and the actual construction of an MC from such history may be performed either in sequence or in an interleaved way.

In a *sequential approach*, a history generator is first run to produce a history; later, such history is given as an input to another algorithm which builds the corresponding Multi-Complex by using Algorithm 7.3 or Algorithm 7.5.

In a *interleaved approach*, an MC is incrementally built while the history is being created, by using Algorithm 7.1 and Algorithm 7.2 or 7.4. In this case, the construction of an MC is implemented within the same algorithm which performs incremental refinement or coarsening.

We think that the sequential approach is better at least for the two reasons listed below:

- *Modularity*: since history generation and MC construction are implemented as two independent programs, the program for MC construction can be used with different history generators. In addition, writing a history does not require any knowledge about Multi-Complexes.
- *Efficiency*: in an interleaved approach, a current MC has to be stored along with the current mesh and with the internal state of the refinement or coarsening process; thus, the overall space requirements are much higher.

7.4 A Specification of Operations for the Construction of an MC

After having defined the operations for MC construction through refinement and through coarsening, in this section we provide a specification of their implementation.

We use the same object-oriented style used for specifying the implementation of query operations in Section 5.8. We refer to the same basic classes introduced in Section 5.8, i.e., the class `MC` of Multi-Complexes, and the classes `Node`, `Arc` and `Cell` of nodes, arcs and top-cells of an MC. We recall that we are acting of the *enriched MC*, as in Section 5.8. For consistency with Section 5.8, we restrict our attention to simplicial MCs, although all algorithms presented here are independent of the types of cells.

We divide the relevant operations into three levels:

1. *Kernel operations*, i.e., basic primitive operations acting on the structure of an MC.

   Operations in this level perform elementary modifications on an MC (such as adding a node, an arc, a cell). Their purpose is hiding the data structure used to represent the MC.

2. *Basic construction operations*, which allow initializing an MC and modifying it according to a new refinement or coarsening pattern.

   These operations are defined on top of the kernel level. In an interleaved approach, they are used within a history generator to incrementally build an MC; they are also used as subcomponents of the operations at the upper level.

3. *History-oriented construction operations*, which directly build an MC starting from a given history.

   These operations are used in the sequential approach to MC construction.

Operations in levels 2 and 3 are available to the user. Figure 7.5 shows the relations between the three levels of operations.

As an implementation remark, we note that sample data structures presented in Chapter 6 are not efficient for construction operations since they are static (array-based). Construction operations are better supported by dynamic data structures using pointers, although such structures require more storage space. Since construction and use of an MC are two distinct tasks, it is still possible to build an MC into a dynamic structure, then write it to disk and reload it into a static structure for query.

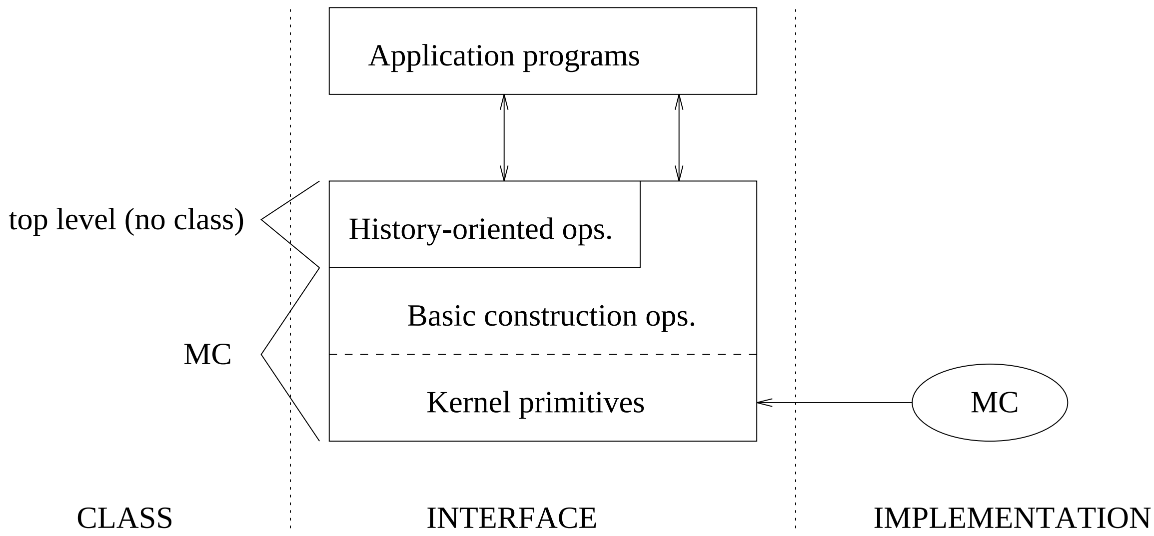


Figure 7.5: The three levels of construction operations in the MC data type.

7.4.1 Kernel Primitives for MC Construction

Kernel primitives allow modifying the internal structure of an MC at the lowest level (i.e., a DAG with sets of triangles associated with its nodes and arcs). They do not guarantee that the DAG they manipulate remains an MC. It is responsibility of the operations at the upper level to use kernel operations in such a way to maintain the underlying DAG an MC.

Kernel operations are private members of class `MC`. The basic kernel primitives are listed below:

- `AddNode()` $\longrightarrow$ `Node`.

  $\mathcal{M}$.`AddNode()`: adds a new node to $\mathcal{M}$, and returns the newly created node; the new node corresponds to a modification pattern C where $\Pi_1(C) = \Pi_2(C) = \emptyset$.

- `AddArc(Node, Node)` $\longrightarrow$ `Arc`.

  $\mathcal{M}$.`AddArc`(C_1, C_2): adds a new arc to $\mathcal{M}$ having C_1 as its source and C_2 as its destination, and and returns the newly created arc; the new arc is labelled with an empty set of cells.

- `DelArc(Arc)`.

  $\mathcal{M}$.`DelArc`(A): removes the given arc from $\mathcal{M}$.

- $\texttt{AddCell}_1$`(Cell, Node)`.

  $\mathcal{M}.\texttt{AddCell}_1(\gamma, C)$: adds cell γ to $\Pi_1(C)$.

- $\texttt{AddCell}_2$`(Cell, Node)`.

  $\mathcal{M}.\texttt{AddCell}_2(\gamma, C)$: adds cell γ to $\Pi_2(C)$.

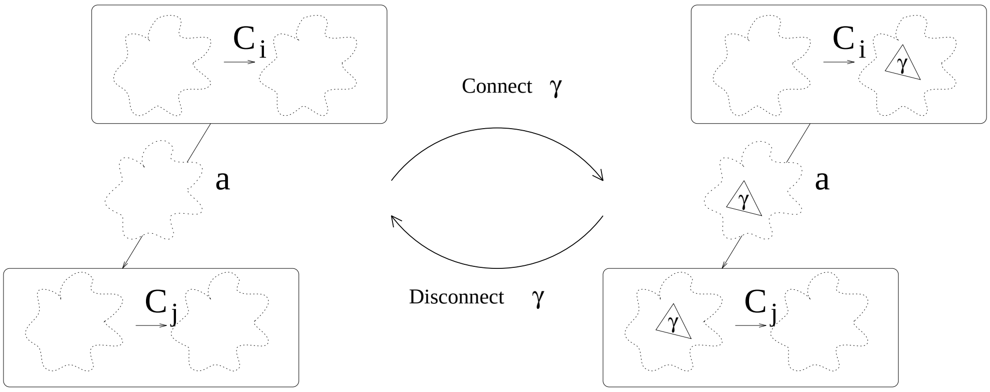


Figure 7.6: Effect of primitive `ConnectCell` and `DisconnectCell` on an MC (only the elements involved in the operation are shown).

- $\texttt{DelCell}_1(\texttt{Cell}, \texttt{Node})$.

  $\mathcal{M}.\texttt{DelCell}_1(\gamma, C)$: deletes cell γ from $\Pi_1(C)$.

- $\texttt{DelCell}_2(\texttt{Cell}, \texttt{Node})$.

  $\mathcal{M}.\texttt{DelCell}_2(\gamma, C)$: deletes cell γ from $\Pi_2(C)$.

- $\texttt{AddArcCell}(\texttt{Cell}, \texttt{Arc})$.

  $\mathcal{M}.\texttt{AddArcCell}(\gamma, A)$: adds cell γ to the set of cells associated with arc A.

- $\texttt{DelArcCell}(\texttt{Cell}, \texttt{Arc})$.

  $\mathcal{M}.\texttt{DelArcCell}(\gamma, A)$: deletes cell γ from the set of cells associated with arc A.

For convenience, we define two derived operations, each corresponding to a sequence of calls to the primitives above. Such derived operations are still private. They maintain the invariant property that two nodes C_i and C_j are connected by an arc if and only if $\Pi_2(C_i) \cap \Pi_1(C_j)$ contains at least a top cell:

- $\texttt{ConnectCell}(\texttt{Cell}, \texttt{Node}, \texttt{Node})$.

  $\mathcal{M}.\texttt{ConnectCell}(\gamma, C_i, C_j)$: adds a new cell γ inside an MC, having node C_i as its source and node C_j as its destination. It adds γ to $\Pi_2(C_i)$ and to $\Pi_1(C_j)$, creates an arc (C_i, C_j) (if not present), and associates γ with such an arc (see Figure 7.6).

  It corresponds to the following sequence of operations:

  > if $C_j \notin \mathcal{M}.\texttt{Children}(C_i)$ then $A := \mathcal{M}.\texttt{AddArc}(C_i, C_j)$;
  > else $A :=$ the arc in $C_i.\texttt{OutArcs}()$ such that $A.\texttt{Dest}() := C_j$;
  > $\mathcal{M}.\texttt{AddCell}_1(\gamma, C_j)$;

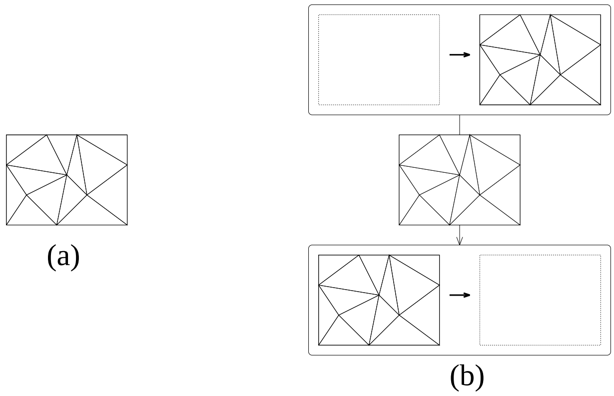


Figure 7.7: Initialization of a Multi-Complex: (a) the initial cell complex of the history, (b) the resulting enriched MC.

$\mathcal{M}$.AddCell$_2(\gamma, C_i)$;
$\mathcal{M}$.AddArcCell(γ, A);

- DisconnectCell(Cell).

  $\mathcal{M}$.DisconnectCell(γ): removes an existing cell γ from an $\mathcal{M}$. It disconnects γ from its source C_i, from its destination C_j, and from arc (C_i, C_j); it removes arc (C_i, C_j) if γ is the only cell associated with such an arc (see Figure 7.6).

  It corresponds to the following sequence of operations:

  $C_i := \gamma$.CellSource();
  $C_j := \gamma$.CellDest();
  $A := \gamma$.CellArc();
  $\mathcal{M}$.DelCell$_2(\gamma, C_i)$;
  $\mathcal{M}$.DelCell$_1(\gamma, C_j)$;
  $\mathcal{M}$.DelArcCell(γ, A);
  if A.ArcCells() $= \emptyset$ then $\mathcal{M}$.DelArc(A);

7.4.2 Basic Construction Operations

Operations in this layer permit to initialize and iteratively add nodes to an MC. They correspond to Algorithms 7.1, 7.2 and 7.4, described in Section 7.3.

Before presenting such operations and their implementation based on kernel primitives, we introduce some considerations about how the initialization and update of a Multi-Complex act on the corresponding enriched MC.

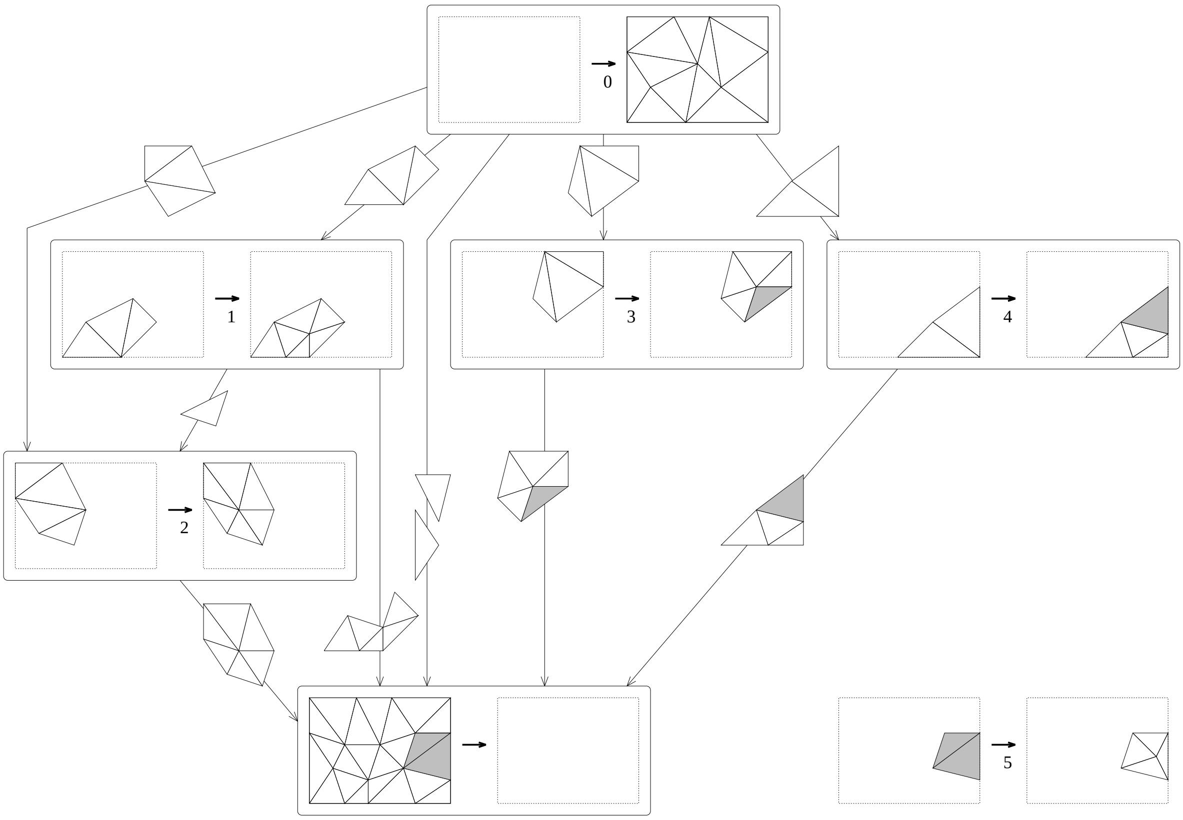


Figure 7.8: One step of the top-down construction of an MC: the current enriched MC, and the next refinement pattern, valid for the front cell complex. The triangles of the MC shared by the refinement pattern are shaded; the resulting MC is shown in Figure 7.9.

Denoted with Γ_0 the initial cell complex of the history, we start with an (enriched) MC having just the root and the drain, and one arc connecting them, labelled with Γ_0 (see Figure 7.7). Then, we create a new MC-node for each modification pattern C_i of the history. In the construction through refinement, we progressively refine the front cell complex of the MC being built. In the construction through coarsening, we progressively coarsen the cell complex at the root of the current MC, while the front cell complex remains unchanged.

In a refinement history, every modification pattern C_i of the history becomes an MC-node, and a parent of the drain. When we encounter C_i in the history sequence, we have to insert the top-cells of $\Pi_2(C_i)$ into the MC. Let γ be a top-cell in $\Pi_2(C_i)$: its source is known (it is the new MC-node just created for representing C_i), but its destination is still unknown (since we do not know whether some modification pattern C_j exists later in the sequence such that $\gamma \in \Pi_1(C_j)$). Thus, the destination of γ is temporarily set to the drain (i.e., γ becomes a cell of the front complex), and it will be possibly changed in a subsequent step (see Figure 7.8 and 7.9).

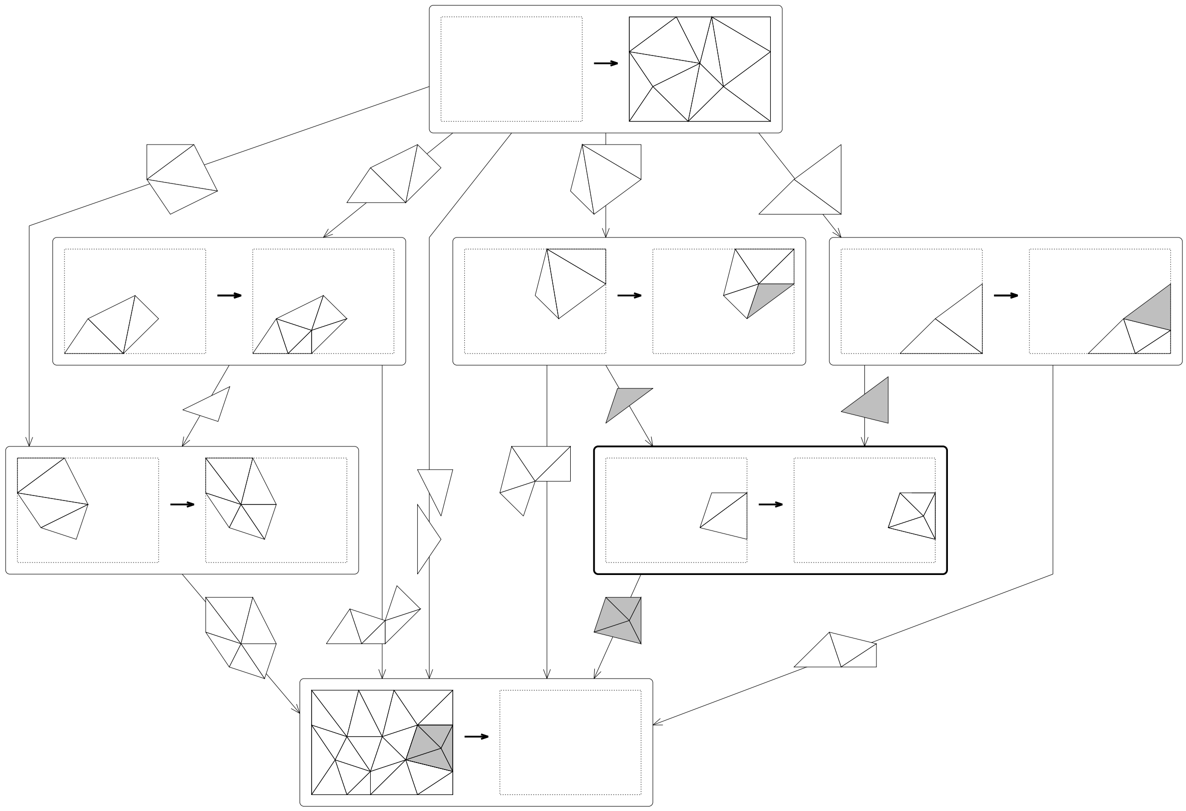


Figure 7.9: One step of the top-down construction of an MC: the resulting enriched MC. The original enriched MC and the coarsening pattern inserted are shown in Figure 7.8. The triangles defining direct dependency links for the new node are shaded.

In a coarsening history, for each modification pattern C_i in the history, the reverse C_i' of C_i becomes an MC-node. When processing C_i, we also have to insert all the top-cells of $\Pi_2(C_i) \equiv \Pi_1(C_i')$ into the MC. Let γ be a top-cell in $\Pi_2(C_i)$: its destination is known (it is the new MC-node just created for C_i'), but its source is still unknown. Thus, the source of γ is temporarily set to the root (i.e., γ becomes a cell in $\Pi_2(C_0')$, where C_0' denotes the root), and it will be possibly changed in a subsequent step (see Figure 7.10 and 7.11).

Note that the two methods act on the enriched MC symmetrically, by adding a node immediately before the drain, and immediately after the root, respectively.

Now, we present the implementation of the three operations based on kernel primitives. Such operations are public in class `MC` in order to be used within a history generator in an interleaved approach. In the specification below, type `Parts(Cell)` denotes the type of cell complexes, and a pair of objects of this type represents a modification pattern:

- `InitMC(Parts(Cell))` $\longrightarrow$ `MC`.

Figure 7.10: One step of the bottom-up construction of an MC: the current enriched MC and the next coarsening pattern, valid for the mesh at the root. The triangles of the MC shared by the coarsening pattern are shaded; the resulting MC is shown in Figure 7.11.

`InitMC`(Γ_0) accepts a cell complex Γ_0 (the initial cell complex of the history) and returns a DAG consisting of just the root node, labelled with a modification pattern $C_0 = (\emptyset, \Gamma_0)$, the drain node, and an arc connecting the root to the drain. It is implemented by applying a modification pattern of the form $(\emptyset, \Gamma_0)$ to a DAG containing just the drain node, and no arcs (such modification pattern can be thought as creation of cell complex Γ_0 from scratch):

$\mathcal{M}$:= empty DAG;
$C_D := \mathcal{M}$.`AddNode`();
$\mathcal{M}$.`RefineMC`($\emptyset, \Gamma_0$);
return $\mathcal{M}$;

- `RefineMC(Parts(Cell), Parts(Cell))`.

  $\mathcal{M}$.`RefineMC`($\Gamma_{old}, \Gamma_{new}$) accepts an MC and a pair $C \equiv (\Gamma_{old}, \Gamma_{new})$ which is a refinement pattern for the front cell complex of $\mathcal{M}$. It adds a new MC-node representing C and connects it to the drain and to the nodes that directly block it, possibly detaching some of them from the drain. The operation is implemented based on kernel primitives as follows:

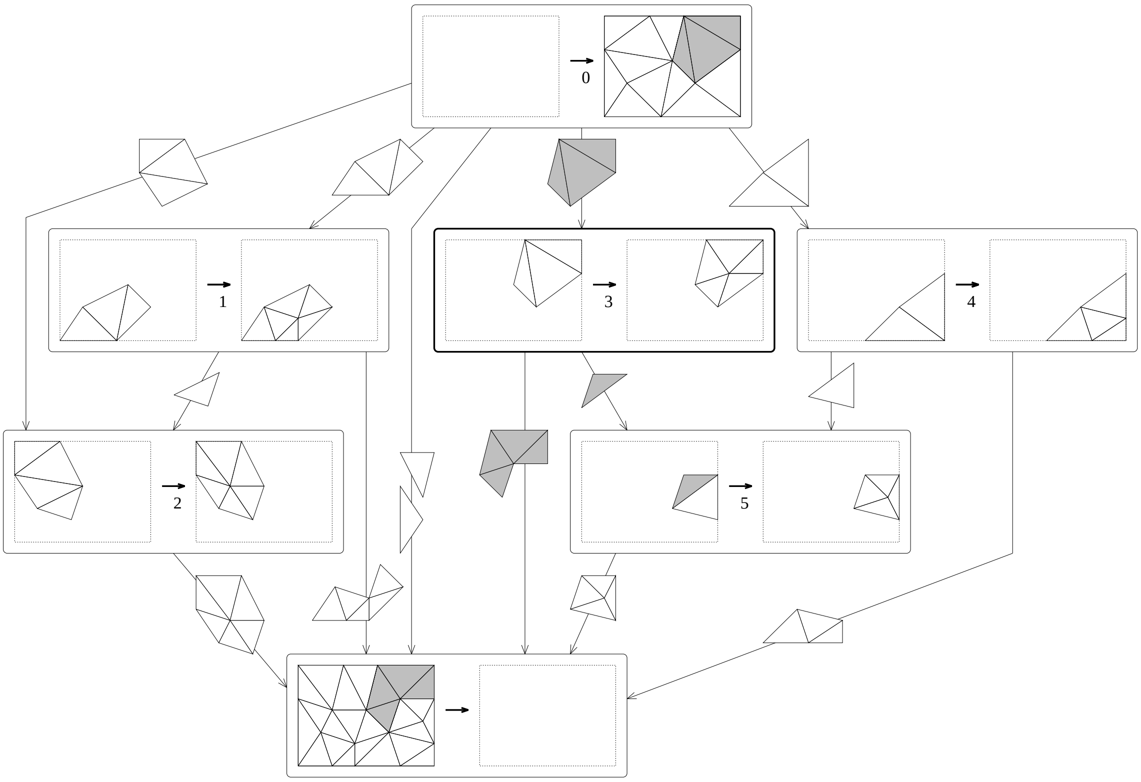


Figure 7.11: One step of the bottom-up construction of an MC: the resulting enriched MC. The original enriched MC and the coarsening pattern inserted are shown in Figure 7.10. The triangles defining direct dependency links for the new node are shaded.

$C := \mathcal{M}.$`AddNode`$()$;
for each cell $\gamma \in \Gamma_{old}$ do
  $C_i := \mathcal{M}.$`CellSource`(γ);
  $\mathcal{M}.$`DisconnectCell`(γ);
  $\mathcal{M}.$`ConnectCell`(γ, C_i, C);
for each cell $\gamma \in \Gamma_{new}$ do $\mathcal{M}.$`ConnectCell`$(\gamma, C, \mathcal{M}.$`Drain`$())$;

- `CoarsenMC(Parts(Cell), Parts(Cell))`.

  $\mathcal{M}.$`CoarsenMC`$(\Gamma_{old}, \Gamma_{new})$ accepts a pair $C \equiv (\Gamma_{old}, \Gamma_{new})$ which is a coarsening pattern for the cell complex contained in the root of $\mathcal{M}$. It adds a new MC-node representing $C' \equiv (\Gamma_{new}, \Gamma_{old})$ (i.e., the reverse of C), updates the root mesh by applying C to it, and connects the new node to the root and to the nodes that directly depend from it, possibly detaching some of them from the root. It is implemented as follows:

  $C' := \mathcal{M}.$`AddNode`$()$;
  for each cell $\gamma \in \Gamma_{old}$ do

$C'_j := \mathcal{M}.\mathtt{CellDest}(\gamma)$;
$\mathcal{M}.\mathtt{DisconnectCell}(\gamma)$;
$\mathcal{M}.\mathtt{ConnectCell}(\gamma, C', C'_j)$;
for each cell $\gamma \in \Gamma_{new}$ do $\mathcal{M}.\mathtt{ConnectCell}(\gamma, \mathcal{M}.\mathtt{Root}(), C')$;

7.4.3 History-Oriented Construction Operations

Operations in this level accept a history, and directly build the corresponding Multi-Complex. We define two operations, corresponding to the construction of an MC top-down (from a refinement history) and bottom-up (from a coarsening history), respectively, i.e., to Algorithms 7.3 and 7.5. These operations are defined at top level, outside class `MC`, and are implemented on top of the basic construction operations of the MC class:

- $\mathtt{TopDownMC}(\mathtt{Parts}(\mathtt{Cell}), \mathtt{Seq}(\mathtt{Parts}(\mathtt{Cell}) \times \mathtt{Parts}(\mathtt{Cell}))) \longrightarrow \mathtt{MC}$.

  $\mathtt{TopDownMC}(\Gamma_0, C_1, \ldots, C_n)$: builds an MC from a refinement history $(\Gamma_0, C_1, \ldots, C_n)$. Its implementation reduces to one call to `InitMC` followed by a sequence of calls to `RefineMC`:

  $\mathcal{M} := \mathtt{InitMC}(\Gamma_0)$;
  for $i = 1, \ldots, n$ do $\mathcal{M}.\mathtt{RefineMC}(\Pi_1(C_i), \Pi_2(C_i))$;
  return $\mathcal{M}$;

- $\mathtt{BottomUpMC}(\mathtt{Parts}(\mathtt{Cell}), \mathtt{Seq}(\mathtt{Parts}(\mathtt{Cell}) \times \mathtt{Parts}(\mathtt{Cell}))) \longrightarrow \mathtt{MC}$.

  $\mathtt{BottomUpMC}(\Gamma_0, C_1, \ldots, C_n)$: builds an MC from a coarsening history $(\Gamma_0, C_1, \ldots, C_n)$. Its implementation reduces to one call to `InitMC` followed by a sequence of calls to `CoarsenMC`:

  $\mathcal{M} := \mathtt{InitMC}(\Gamma_0)$;
  for $i = 1, \ldots, n$ do
  $\mathcal{M}.\mathtt{CoarsenMC}(\Pi_1(C_i), \Pi_2(C_i))$;
  return $\mathcal{M}$;

  Alternatively, operation `BottomUpMC` could be implemented through a call to `TopDownMC` by first reversing each modification pattern of the given history, and reversing their sequential order, as follows:

  $\Gamma := \Gamma_0$;
  for $i = 1, \ldots, n$ do
  $C'_i := (\Pi_2(C_i), \Pi_1(C_i))$;
  $\Gamma := \Gamma[C_i]$;
  return $\mathtt{TopDownMC}(\Gamma, C'_n, \ldots, C'_1)$;

7.5 Influence of the History Generator on Quality of a Multi-Complex

In this section, we present and analyze various history generators in order to evaluate the impact of different criteria used for refining / coarsening a cell complex on the properties of the resulting Multi-Complex. To our knowledge, a comparison of this kind is done here for the first time, and it is made possible thanks to the fact that the MC is a general model. A preliminary version of this work appeared in [DFMP97a].

We measure the *quality* of a Multi-Complex based on the shape parameters (width, height, growth) introduced in Section 4.6, plus an additional parameter, called the *compression power*:

Definition 7.4 *The* compression power *of a Multi-Complex $\mathcal{M}$ for a certain resolution filter R is the ratio of the size of the front cell complex of $\mathcal{M}$ over the size of the cell complex resulting from a global query on $\mathcal{M}$ with R.*

More precisely, good history generators are the ones for which the corresponding MC has:

- *High compression power*, i.e., the necessary size for an extracted cell complex to achieve a certain accuracy of representation must be relatively small.
- *Small growth*, i.e., the number of cells traversed by a query algorithm in order to extract a given complex must be relatively small compared with the size of the complex at variable resolution obtained as the result of the query. This parameter measures the overhead introduced by the use of a multiresolution model with respect to directly querying a complex at variable resolution; for instance, if an MC has linear growth, then the overhead is linear.
- *Small width*, i.e., a single MC-node must contain a small number of top cells. The smaller the width, the lower the cost to traverse a directed path in the DAG (e.g., for point location through a local query). In particular, we would like a bounded width, where the bounding constant is relatively small.
- Small *height*, i.e., a path on the DAG must be relatively short; for instance, a logarithmic height.

Growth, width and height are structural properties of an MC, independent of the object represented in it. The compression power is a property of an MC related to the represented object, since it depends on the accuracy of cell complexes extracted from the model.

The values of the quality parameters for an MC depend on the properties of the sequence of modification patterns contained in it, which, in turn, depend on the history generator used.

We can devise two opposite perspectives in history generation:

- Heuristics aimed at obtaining *high compression power*, which do not guarantee a provably good shape for the resulting DAG. Such methods operate in a greedy way. In a refinement process, they try to decrease the approximation error as rapidly as possible; in a coarsening process, they try to increase it as slowly as possible.
- Theoretically well-founded criteria that guarantee *good shape parameters*, but may need considerably more cells to achieve a given accuracy.

Here, we explore a number of methods, trying to find a good balance of these opposite issues.

We consider refinement and coarsening approaches based on iterative vertex insertion and removal, respectively. Although the strategies presented are more general, we illustrate their implementation in the case of two-dimensional scalar fields. We have chosen two-dimensional scalar fields because:

- we can adopt a Delaunay triangulation of the domain, for which we have developed efficient vertex insertion and vertex removal algorithms;
- we compute approximation errors according to the metric defined for fields, which is easy to evaluate;
- it is possible to experiment with both the refinement and the coarsening approaches, whereas for other entities (e.g., free-form surfaces) only coarsening methods apply;
- we are guaranteed to obtain conflict-free histories.

We assume that the input data consists of a finite set S of n points in $\mathbb{E}^2$ at which the field value is known. Approximation errors are estimated by using the discrete field distance introduced in Section 2.4.3: we consider, for each triangle σ, the maximum vertical distance between a data point p, whose xy-projection falls within σ, and the plane through the three data points at the vertices of σ. The front triangulation of the constructed MC will be a (Delaunay) triangulation of S; thus, its size will be $O(n)$.

For each algorithm described, we evaluate its time complexity and the theoretical characteristics of the resulting MCs. An experimental evaluation of the resulting MCs is provided in Section 7.5.3. Note that the time complexity of a history generator is not a critical issue, as long as it remains within reasonable bounds, since the construction of a Multi-Complex is done off-line. On the contrary, the characteristics of the resulting structure, in terms of growth, width, height, and size necessary to achieve a given accuracy, may have great relevance in the applications.

7.5.1 Refinement History Generation

The general scheme for a refinement history generator is the following:

1. Compute a Delaunay triangulation Σ_0 of the vertices on the convex hull of S; complex Σ_0 is the initial cell complex of the history.

2. While there are points of S which are not vertices of the current triangulation Σ:

   - select a point p to be inserted as a new vertex;
   - insert p and update Σ according to the Delaunay criterion; add the corresponding refinement pattern to the history.

Point insertion is done by first locating the triangle σ of the existing triangulation that contains the new point p; then, p is inserted into the current triangulation by splitting σ; finally, the triangulation is optimized according to the Delaunay criterion by iteratively applying edge flips. The portion of the triangulation modified when inserting a point p is exactly the one covered by the triangles incident at p after the optimization. Thus, inserting a point (except the location of the triangle σ) costs $O(k)$, where k is the number of its incident triangles after insertion. In the worst case, k can be $O(n)$ (where n is the number of vertices in the triangulation containing p), but it is constant on average [GKS92].

Variants of the refinement technique depend on how point p is selected at each iteration, and have an influence on the resulting MC. Here, we describe two variants.

7.5.1.1 Variant TD1.

In this variant, a random point is selected for insertion at each iteration.

The resulting MC has none of the desired properties (linear growth, bounded width, and logarithmic height). Since each insertion of a point may modify $O(n)$ triangles in the worst case, $O(n^2)$ different triangles may be generated during the process; thus, the growth of the MC may be quadratic. In the worst case, $O(n)$ point insertions may modify the same location, thus yielding to a sequence of $O(n)$ modification patterns, each depending on the previous one; thus, the height of the MC may be up to $O(n)$. Finally, there is no way to bound the number of triangles created when inserting a point; thus, the MC has not a bounded width.

However, the resulting MC is expected to behave well in the average case. According to the randomized analysis given in [GKS90], the overall number of intermediate triangles created during the refinement process is expected to be linear in the output size; thus, the resulting MC has linear growth with high probability. The average number of triangles created during the process which interfere over the same location is expected to be logarithmic in the output size; thus, the average length of a path in resulting MC is logarithmic.

Since the selection of points to be inserted does not take into account the field error, approximation errors for triangles can be evaluated in a postprocessing step. For this purpose, we can use the search structure of [GKS90], which can be built during the refinement process.

Such a structure can support point location queries and can provide, in expected logarithmic time, all the triangles that contained a point $p \in S$ during the evolution of the mesh. Point p contributes to the approximation errors of exactly such triangles, and thus error evaluation for all the triangles of the MC is done in expected $O(n \log n)$ time.

According to the randomized analysis given in [GKS90], the refinement algorithm runs in expected $O(n \log n)$ time. The worst-case time complexity is $O(n^2)$.

Since the algorithm does not take into account the field values, the approximation error may decrease very slowly during refinement; thus, the resulting Multi-Complex will show poor compression power (i.e., a lot of points may need to be inserted for achieving a given accuracy).

7.5.1.2 Variant TD2.

In this variant, the point causing the maximum error is selected for insertion at each iteration.

This approach privileges a greedy, error-based strategy; thus, the approximation error decreases rapidly while inserting points, and the resulting MCs have better compression power than those generated through variant TD1.

The resulting MCs may have none of the desired characteristics in terms of growth, height, width, for the same reasons explained for variant TD1. Moreover, here the randomized analysis of [GKS90] does not apply.

Unlike variant TD1, approximation errors must be computed during refinement. To this aim, we adopt a bucketing technique similar to that used in [KMS91] for dynamic triangulations. We associate with each triangle σ in the current triangulation the list of points of S (not yet vertices) whose vertical projections lie inside σ. From such points, we compute the approximation error of σ and record the point p corresponding to such error.

All triangles are kept in a priority queue based on their errors; at each step, the triangle of maximum error is selected and the point p associated with it is inserted in the current triangulation. After inserting a vertex, the points that were associated with the disappearing triangles must be relocated among the new triangles and their errors must be recomputed.

An efficient implementation of this variant, described in detail in [DFMP96b], achieves $O(n^2 \log n)$ time complexity in the worst case.

7.5.2 Coarsening History Generation

The general scheme of a coarsening history generator is the following:

1. Start from a Delaunay triangulation Σ_0 having the points of S as its vertices; Σ_0 is the initial cell complex of the history.

2. While vertices can be removed from the current triangulation Σ:

   - select a set V of vertices to be removed, consisting either of one vertex, or of more vertices which are *mutually independent* (i.e., there is not a pair of vertices in V that are the two endpoints of the same edge);
   - for each vertex $v \in V$, remove v and update Σ according to the Delaunay criterion; each removed vertex provides a new coarsening pattern that is added to the history.

Removing a vertex v from a triangulation is done by deleting all the triangles incident at v, creating a triangulation of the "hole" resulting from its deletion, and, finally, optimizing the triangulation through edge-flipping, based on the Delaunay criterion. The optimization stage is guaranteed to involve just the temporary triangles created to fill the hole. Thus, deleting a point costs $O(k)$, where k is the number of triangles incident at v. In the worst case, k can be $O(n)$ (where n is the number of vertices in the triangulation containing p), but it is constant in the average case [GKS92].

A set V of independent vertices can be removed simultaneously since their sets of incident triangles are disjoint; the corresponding modification patterns are independent and can be added to the history in an arbitrary order.

Note that, while refinement methods always insert a single point, coarsening methods can remove a set of vertices simultaneously. This happens because it is easy to select a set of independent vertices from a triangulation, but it is difficult to select a set of data points that will become independent vertices once inserted in a triangulation. Also, in coarsening strategies we can easily guarantee MCs with a bounded width just by removing vertices with a bounded degree. Therefore, coarsening algorithms are able to produce MCs which theoretically guarantee better shape parameters.

In the following, we describe and analyze four variants of a coarsening strategy.

7.5.2.1 Variant BU1.

This variant has the good shape of an MC as its only goal. It is inspired by the algorithm proposed by de Berg and Dobrindt [dBD95] to build their pyramid of triangulations, which is a special case of an MC (see Sections 3.3.4.1 and 4.7.3.2).

At each iteration, the set V of removed vertices is an arbitrary maximal set of independent vertices of bounded degree (at most b). A set V of independent vertices is *maximal* if every vertex of Σ that is not in V is on the same edge as a vertex already in V.

The resulting MC has bounded width because only vertices of degree less than b are deleted, and, therefore, it has linear growth because of theorem 4.6. It has a logarithmic height, due to the following result of Kirkpatrick [Kir83]: given a triangulation with n vertices, the cardinality of any maximal set of independent vertices is $\frac{n}{c}$ for a constant $c > 1$. Thus, the current triangulation reduces to a triangulation of the convex hull of S in a logarithmic number of steps, and the height of the MC is at most $O(\log n)$.

The computational complexity of the algorithm (apart from computation of the initial triangulation) is $O(n)$, but a further $O(n \log n)$ time is needed to evaluate the approximation errors of all triangles in the MC.

In order to compute approximation errors, we have to determine, for each triangle σ in the structure, the set of data points falling inside σ. Since the DAG has a logarithmic height, for each point $p \in S$, at most $O(\log n)$ triangles contain p. All such triangles lie on the same path in the DAG and, due to the bounded width, they can be traversed in $O(\log n)$ time. Since the points in S are n, it follows that the time complexity for computation of approximation errors is $O(n \log n)$ in the worst case. Note that here, as in variant TD1, errors can be computed a-posteriori just because the selection mechanism for removing vertices does not rely on an error-based criterion.

Since the algorithm does not take into account of approximation errors, large errors may be introduced from the very first stages of the process, and the resulting Multi-Complex will show poor compression power.

7.5.2.2 Variant BU2.

Similarily to Variant BU1, this variant eliminates a maximal set of independent vertices of bounded degree at each step, but, here, set V is built by taking into account approximation errors, with a greedy technique that works as follows:

- initialize set V with the vertex whose elimination causes the smallest increase in error among those that have a degree smaller than b;
- while V is not maximal, add to V the vertex causing the smallest error, among those that have a degree smaller than b, and are independent of the vertices already in V.

Since V is a maximal independent set of vertices with bounded degree, as in Variant BU1, the resulting MC has the same characteristics, (i.e., bounded width, linear growth, and logarithmic height). Moreover, the additional heuristics, which makes the selection depend on the error caused by removing a vertex, helps decreasing the accuracy of the model more slowly than in variant BU1. Thus, the compression power of the resulting MC will be better.

The greedy selection of vertices requires that the error caused by removing a vertex of the current triangulation is known at each iteration.

Similarly to Variant TD2, in order to compute approximation errors of triangles we associate with each triangle σ in the current triangulation the set of vertices already deleted whose vertical projections fall in σ.

The error introduced by removing a vertex v is evaluated by simulating the deletion of v: the hole left by v is retriangulated and the error is measured by locating v and all points lying inside its incident triangles with respect to the new triangles that fill the hole, computing the errors of such triangles, and choosing the maximum error.

The error of each vertex v is stored together with v, and it is updated whenever a vertex adjacent to v is removed. Note that it is sufficient to evaluate the error only at vertices whose degree is smaller than b, because they are the only candidates for removal. The triangulation of a hole, necessary to evaluate the error, is performed only at vertices of bounded degree, hence it takes constant time at each vertex.

Vertices of bounded degree are maintained in a priority queue based on their errors; such a queue is examined while building set V and is updated each time a vertex is deleted. Moreover, at each deletion, error must be re-evaluated only for a bounded number of vertices.

Location of points inside a given triangle σ occurs at most four times, namely for evaluating the cost of removing each of its three vertices, and when one of such vertices is actually removed (when σ is removed as well). During the whole coarsening process, a point $p \in S$ may fall in at most $O(\log n)$ different triangles; thus, p must be relocated at most $O(\log n)$ times. Relocating p costs $O(1)$ since the number of triangles which must be tested for point inclusion is bounded by b. It follows that the total cost for computing approximation errors is $O(n \log n)$ in the worst case.

Inquiring and updating the priority queue has a cost of $O(\log n)$ per operation, and a constant number of operations are performed for each vertex removed. Therefore, the total time complexity of building an MC through Variant BU2 is $O(n \log n)$, i.e., the same as with Variant BU1.

7.5.2.3 Variant BU3.

In this variant, at each iteration, set V consists of just one vertex, namely the vertex that causes the smallest error increase, among those having degree smaller than b.

This method can provide MCs with better compression power than Variant BU2 because selection is performed on the basis of a criterion that is only related to error increase caused by each deletion. In Variant BU2, the need of having a maximal independent set may force the algorithm to delete points that cause a high error increase at early stages of the construction.

The resulting MC still has bounded width because only vertices of bounded degree are removed, and linear growth for Theorem 4.6, while it may not have logarithmic height.

We adopt the same tools described for variant BU2 for maintaining the approximation errors of triangles and evaluating the error increase caused by removing a vertex. Unlike in variant

BU2, here a point $p \in S$ may fall in $O(n)$ different triangles during the coarsening process. Thus the total cost for error evaluation if $O(n^2)$, which dominates the time complexity of the algorithm.

7.5.2.4 Variant BU4.

This variant works in the same way as Variant BU3, except for the vertex causing the minimum error increase which is selected, regardless of its degree: this corresponds to the criterion adopted in [CCMS97].

In this variant, the selection strategy is completely error-driven, therefore the greedy heuristic approach, i.e., trying to discard less important points at early stages, is stronger than in any other previous variant. The resulting MC in the worst case may have none of the desired properties in terms of growth, height, and width. The reasons are similar to those explained for variant TD2 (indeed, the two variants are based on symmetric criteria). As in Variant TD2, the growth of the resulting MC might be quadratic.

Each vertex removal may change up to $O(n)$ triangles; thus, the time complexity for updating the current triangulation after all deletions is $O(n^2)$ in the worst case.

Approximation errors of triangles and vertices are computed and maintained as in variants BU2 and BU3. Note that, here, errors must be maintained for all vertices, regardless of the degree. When removing, or simulating the removal of a vertex v, a polygon with $O(n)$ edges must be triangulated and as many as $O(n)$ points must be located with respect to $O(n)$ triangles. The Delaunay triangulation of the polygon can be computed in time $O(n \log n)$, while building a spatial index [Kir83] that supports point location in $O(\log n)$ time. Therefore, triangulation and location of points may require $O(n \log n)$ time at each iteration, yielding a global cost of $O(n^2 \log n)$, which is dominant over the remaining operations performed by the algorithm.

7.5.3 Experimental Results

All the variants of algorithms for history generation presented in Sections 7.5.1 and 7.5.2 have been implemented and experimented with. We have compared the Multi-Complexes resulting from applying the six algorithms presented on the same datasets S.

We show results obtained on three terrain datasets made available from the U.S. Geological Survey:

- *San Bernardino* is a regular grid of 128×128 elevation data;
- *Mount Marcy* is a regular grid of 128×128 elevation data;
- *Devil Peak* is a regular grid of 165×231 elevation data.

The comparison includes a static evaluation of the shape parameters of the MC (size, width, height, growth), and a run-time evaluation of the performance of query operations on it (size of the extracted meshes, compression power, number of triangles traversed). Variants BU1-2-3 have been applied by using a bound $b = 11$ for the maximum degree of vertices that can be eliminated at each step.

In order to evaluate the compression power, we run global queries corresponding to the extraction of a triangulation at uniform accuracy on the whole domain. Such queries have been answered by using the static algorithm for the min-size version of the query (Algorithm 5.2). The ratio between the number of triangles traversed and the output size gives an estimate of the MC growth.

In order to evaluate the efficiency of the model as a spatial index (which depends on its average height and width), we run local queries corresponding to point location (the focus set is a point) at maximum accuracy. Such queries have been answered by using Algorithm 5.5 for conservative MCs. The number of traversed triangles is related to the MC height and width, since it is the sum of the widths of the nodes forming the traversed path.

Table 7.1 summarizes the characteristics of the MCs built through the six construction algorithms. For each variant, the size of the front triangulation ("front tri" column) is the same and corresponds to the triangulation obtained by subdividing each square of the input grid into two right triangles. We show the size of the resulting MC ("MC tri" column), its growth, computed as the ratio of "MC tri" over "front tri", its height, the maximum and average width of a node. We note that all variants produce MCs with small width, including TD1, TD2, and BU4. In particular, BU3 and BU4 have almost the same characteristics since the bounded degree constraint has very little influence. As expected, BU1 and BU2 give the best height, followed by TD1. The heights of other approaches are comparable. Among coarsening variants, BU1 and BU2 give the smallest MC size, but, surprisingly enough, refinement variants give even better results.

Now, we present the results of extracting a triangulation at uniform accuracy according to an increasing sequence of constant error thresholds functions. Figures 7.12, 7.13, and 7.14 show the size of the output triangulation (which is proportional to the inverse of the compression power since the size of the front mesh is fixed). Figures 7.15, 7.16, and 7.17 show the number of visited triangles (i.e., the size of the sub-MC providing the result). Figures 7.18, 7.19, and 7.20 show the growth, measured as the ratio between the number of triangles visited by the query algorithm and the number of triangles in the output. Here, the best variant is the one for which the curves of output size and number of visited triangles are the nearest to the coordinate axes and the growth curve is the lowest (meaning a high compression power and a small growth). In terms of compression power, the best variant is TD2, and the worst one is TD1; coarsening variants removing sets of independent vertices give a worse compression power than those removing one vertex at a time. Again, we note that variants BU3 and BU4 are almost equivalent. The growth is almost linear for the four coarsening variants, and increases

Mount Marcy

Variant	front tri	MC tri	growth	height	max width	avg width
TD1	32258	82574	2.559	81	16	5.041
TD2	32258	80270	2.488	137	10	4.900
BU1	32258	91918	2.849	31	10	5.611
BU2	32258	91482	2.835	32	11	5.584
BU3	32258	96620	2.995	146	11	5.898
BU4	32258	96621	2.995	146	12	5.898

San Bernardino

Variant	front tri	MC tri	growth	height	max width	avg width
TD1	32258	82470	2.556	75	11	5.034
TD2	32258	81202	2.517	138	11	4.957
BU1	32258	91913	2.849	32	10	5.611
BU2	32258	91878	2.848	31	11	5.609
BU3	32258	97547	3.023	145	11	5.955
BU4	32258	97575	3.024	146	14	5.956

Devil Peak

Variant	front tri	MC tri	growth	height	max width	avg width
TD1	75440	193174	2.560	86	13	5.068
TD2	75440	190538	2.525	298	11	4.999
BU1	75440	215193	2.852	34	11	5.646
BU2	75440	214321	2.840	35	11	5.623
BU3	75440	227670	3.017	267	11	5.973
BU4	75440	227683	3.018	367	13	5.974

Table 7.1: Parameters of the MC built through the six variants.

more rapidly for the refinement variants.

Results of point location queries are shown in Table 7.2, which summarize a set of 400 ramdom point location queries performed with a uniform distribution over the domain. Variant BU2 gives the best values. The worst results are given by variants BU3 and BU4. The other variants are not noticeably different. Variant BU1 does not perform much better than refinement variants (although it guarantees bounded width and linear growth in the worst case) because the corresponding MC contains more triangles.

7.6 Summary

In this chapter, we have defined the operations necessary for building a Multi-Complex. The construction of a Multi-Complex consists of two conceptually distinct steps, namely the generation of a history (i.e., of an initial cell complex and a sequence of modification patterns), and the assembly of the corresponding dependency DAG. Although these two steps can be interleaved, we recommend that they are implemented as two separate operations for the sake of modularity and efficiency. History generation relies on iterative algorithms for building approx-

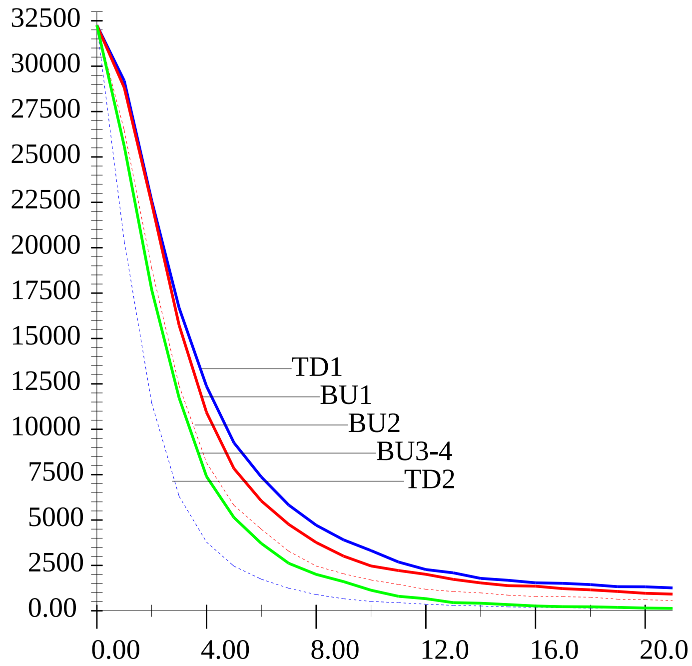


Figure 7.12: Number of triangles in the output TIN of global queries according to uniform error thresholds on Mt. Marcy. The x-axis corresponds to the threshold values, expressed as a percentage of the height range in the terrain data. Colors and line styles have the following meaning: blue for the refinement variants (TD1 solid, TD2 dashed), red for coarsening variants removing independent sets of vertices (BU1 solid, BU2 dashed), green for coarsening variants removing one vertex at a time (BU3 solid, BU4 dashed). Note that solid and dotted green lines are congruent since variants BU3 and BU4 have the same behavior.

imated cell complexes, reviewed in Section 3.2. In this section, the operation corresponding to the construction of the DAG from such data has been formally defined.

We have analyzed the influence of the algorithm used to create the initial complexes and of the sequence of modifications on the efficiency of a Multi-Complex. To this aim, we have presented and compared, both from a theoretical and from an experimental perspective, six approaches designed for the case of terrain modeling based on Delaunay triangulations. We have experimentally found that Variant TD2 (an error-driven refinent variant) achieves a good overall performance, although Variant BU2 (an error-driven coarsening variant removing an independent set of vertices of small degree) performs better than TD2 on point location queries.

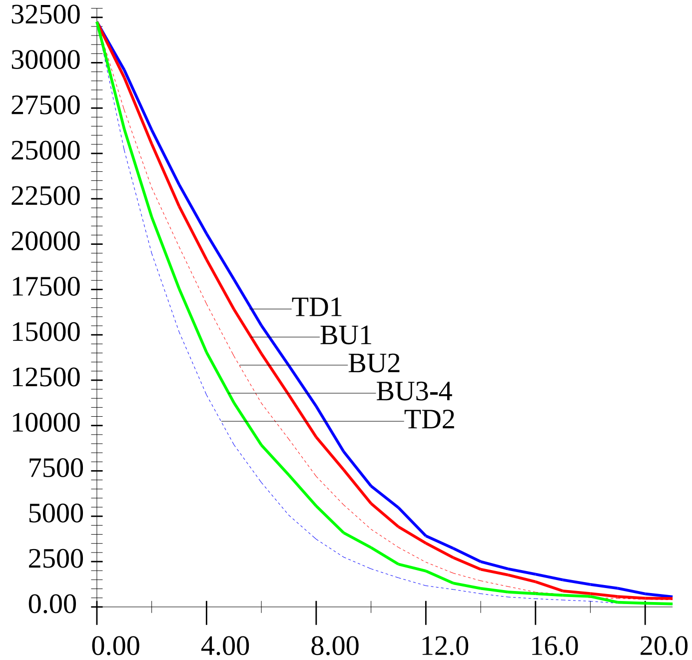


Figure 7.13: Number of triangles in the output TIN of global queries according to uniform error thresholds on San Bernardino. The same notation, colors, line styles, and labels are used as in Figure 7.12.

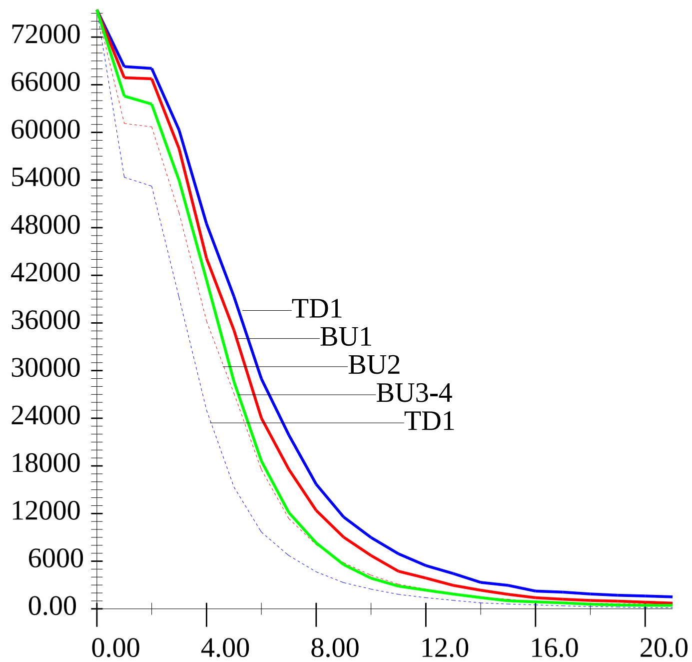


Figure 7.14: Number of triangles in the output TIN of global queries according to uniform error thresholds on Devil Peak. The same notation, colors, line styles, and labels are used as in Figure 7.12.

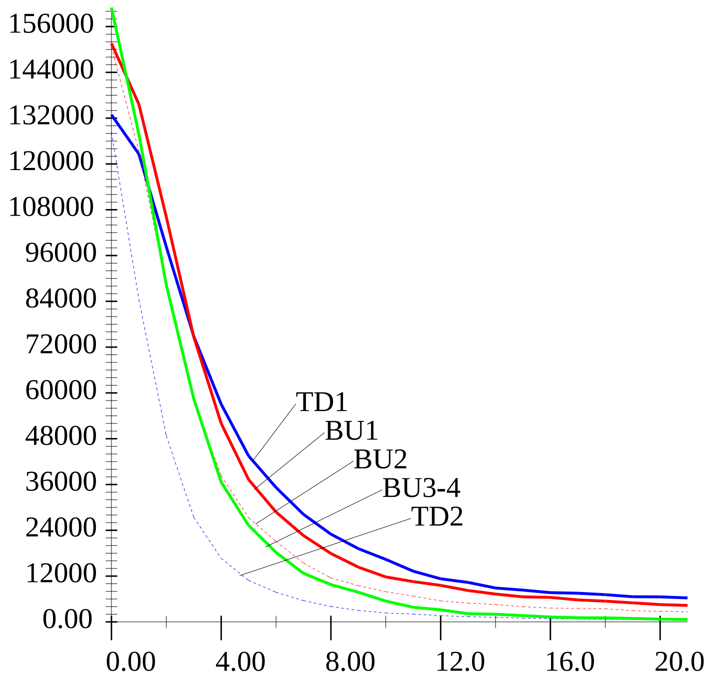


Figure 7.15: Number of visited triangles of global queries according to uniform error thresholds on Mt. Marcy. The same notation, colors, line styles, and labels are used as in Figure 7.12.

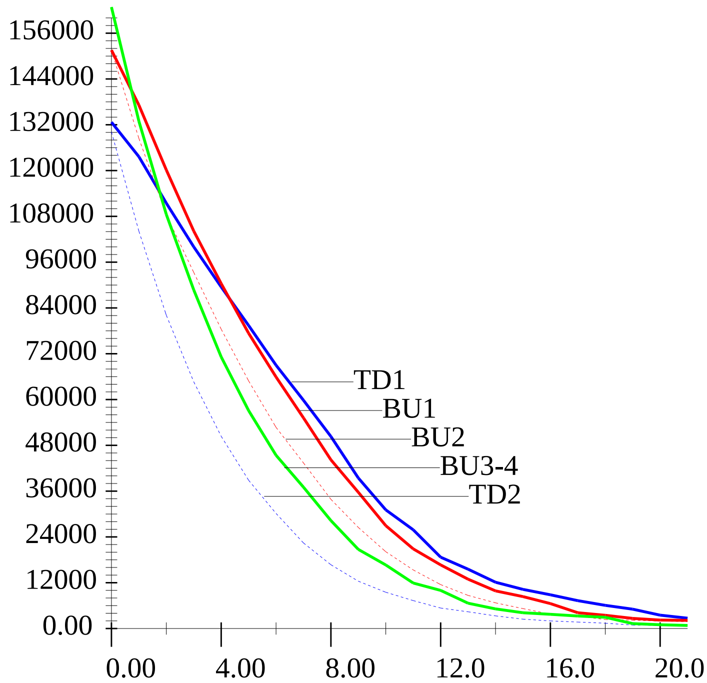


Figure 7.16: Number of visited triangles of global queries according to uniform error thresholds on San Bernardino. The same notation, colors, line styles, and labels are used as in Figure 7.12.

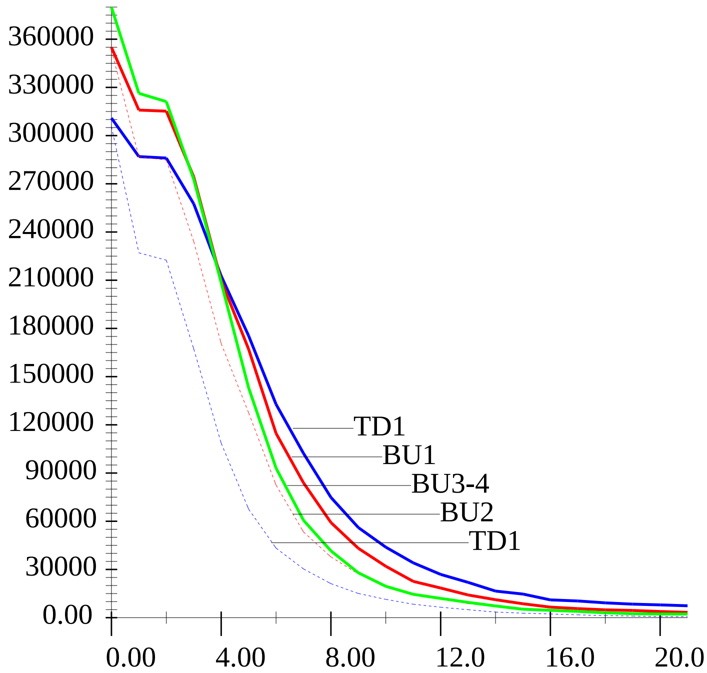


Figure 7.17: Number of visited triangles of global queries according to uniform error thresholds on Devil Peak. The same notation, colors, line styles, and labels are used as in Figure 7.12.

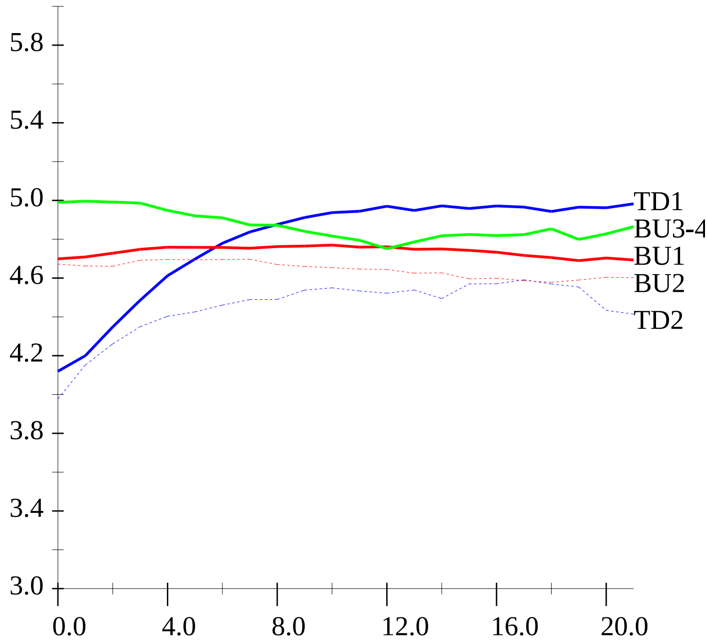


Figure 7.18: Ratio between the number of traversed triangles and the output size (growth) of global queries according to uniform error thresholds on Mt. Marcy. The same notation, colors, line styles, and labels are used as in Figure 7.12.

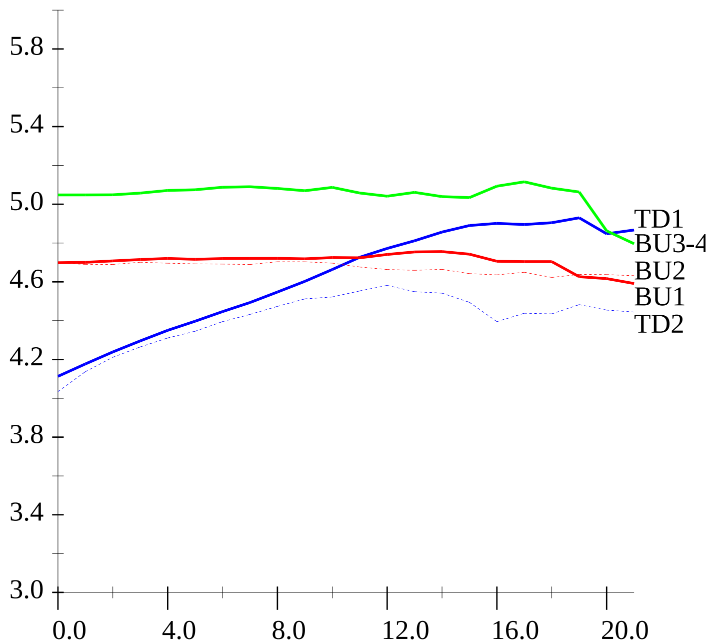


Figure 7.19: Ratio between the number of traversed triangles and the output size (growth) of global queries according to uniform error thresholds on San Bernardino. The same notation, colors, line styles, and labels are used as in Figure 7.12.

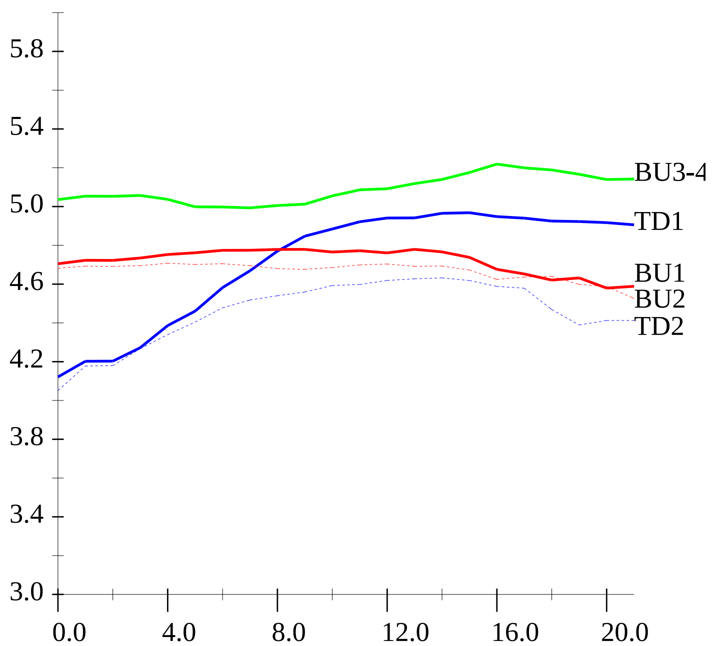


Figure 7.20: Ratio between the number of traversed triangles and the output size (growth) of global queries according to uniform error thresholds on Devil Peak. The same notation, colors, line styles, and labels are used as in Figure 7.12.

Mt. Marcy

Variant	visited tri			overhead		
	avg	max	min	avg	max	min
TD1	96.394	236	28	0.002988	0.007316	0.000868
TD2	84.879	199	15	0.002631	0.006169	0.000465
BU1	91.791	232	44	0.002846	0.007192	0.001364
BU2	88.664	154	37	0.002749	0.004774	0.001147
BU3	115.297	288	9	0.003574	0.008928	0.000279
BU4	115.297	288	9	0.003574	0.008928	0.000279

San Bernardino

Variant	visited tri			overhead		
	avg	max	min	avg	max	min
TD1	92.392	236	27	0.002864	0.007316	0.000837
TD2	109.718	313	37	0.003401	0.009703	0.001147
BU1	87.730	237	25	0.002720	0.007347	0.000775
BU2	91.083	194	41	0.001483	0.006014	0.000031
BU3	134.244	287	45	0.004162	0.008897	0.001395
BU4	134.639	283	45	0.004174	0.008773	0.001395

Devil Peak

Variant	visited tri			overhead		
	avg	max	min	avg	max	min
TD1	105.378	210	42	0.001397	0.002784	0.000557
TD2	118.530	358	36	0.001571	0.004745	0.000477
BU1	101.602	262	42	0.001347	0.003473	0.000557
BU2	106.311	221	55	0.001409	0.002929	0.000729
BU3	153.254	405	43	0.002031	0.005369	0.000570
BU4	153.354	405	43	0.002033	0.005369	0.000570

Table 7.2: Results of point location queries at full resolution. We report the number of triangles traversed and the overhead, computed as the ratio between the number of traversed triangles and the size of the triangulation at full resolution.

Chapter 8

Applications of the Multi-Complex

8.1 Introduction

In this chapter, we show applications of the Multi-Complex within the case-study topics listed in Section 2.7. We provide implementation details and show experimental results where already available.

A Multi-Complex is instantiated for a specific application by:

- identifying a *class of MCs* suitable to represent the objects within the application domain;
- providing *construction algorithms* that permit to build an MC for an object starting from available data;
- defining *operations of the application* based on suitable instances of the general query operations of an MC.

For all applications we use simplicial Multi-Complexes of the appropriate dimension.

According to the directions outlined in Section 7.2, defining construction algorithms reduces to defining history generators, since operations for building an MC from a history are part of the standard interface of the Multi-Complex data type. It is necessary to find iterative methods for the incremental refinement or coarsening of a cell complex representing an object in the application domain. In some cases, such algorithms already exist in the literature (see Section 8.3.1 and 8.5). In other cases, we have developed our own methods based on techniques designed before we developed the MC (see, for instance, the methods described in Section 7.5 for two-dimensional scalar fields). In some cases, finally, defining history generators has required to develop new techniques (see Section 8.3.2 and 8.4).

The simplest application-dependent operations reduce to queries on a Multi-Complex and, thus, the only concern is identifying the appropriate type of query (either global or local), the appropriate query parameters (i.e., the resolution filter and the focus set), and the most convenient approach for the solution of the query (e.g., for a global query, a basic or a dynamic algorithm). More complex application-dependent operations also involve post-processing on the result of a query.

For each application, we provide history generators and application-dependent operations that have been developed and either have been implemented or are under implementation. We briefly discuss other operations that are of interest for the application and mention some open problems.

Our final objective has been to develop a software library based on the Multi-Complex which encapsulates the application-independent features of Multi-Complexes. Such a library provides data structures for encoding the MC, operations for constructing an MC from a history, and the basic query operations. Using the basic functions provided in the library, applications are able to construct and store their models and implement application-specific operations on top of the query operations of the library.

This chapter is organized as follows:

- In Section 8.2, we consider the application of the MC to represent, display and analyze *terrains* in GISs.
- In Section 8.3, we consider the application of the MC to represent *surfaces in three-dimensional space*. We provide methods for building an MC for a surface given either as a dense triangle mesh (as surfaces acquired through scanning equipments), or as a boundary representation (as surfaces produced by CAD modelers).
- In Section 8.4, we develop an MC for the *volumetric representation of solids*. We propose an MC that encompasses a collection of nested shapes approximating an object, where coarser shapes properly contains finer ones.
- In Section 8.5, we consider the application of the MC to represent and visualize *volume data* (three-dimensional scalar fields).
- In Section 8.6, we apply the MC to the *compression and progressive transmission* of triangle meshes.
- In Section 8.7, we describe the architecture of the software library that we have developed based on the MC.

8.2 Terrains in Geographic Information Systems

A terrain is a special case of a two-dimensional scalar field $\phi : \Omega \subseteq \mathbb{E}^2 \longrightarrow \mathbb{E}$ where the field value means elevation. We consider terrain modeling based on triangulations.

The Multi-Complexes considered for terrain representation are two-dimensional simplicial MCs embedded in $\mathbb{E}^2$, where each vertex p has an associated height value $\phi(p)$ and each triangle σ can be mapped into $\mathbb{E}^3$ by adding the field value as a third coordinate to its three vertices.

Thus, vertex heights define an embedding of the MC into three-dimensional space. In the reminder of this section, we consider a Multi-Complex representing a terrain as embedded either in $\mathbb{E}^2$ or in $\mathbb{E}^3$, depending on the context. If $\mathcal{M}$ denotes the MC in $\mathbb{E}^2$, then its embedding in $\mathbb{E}^3$ is denoted with $\hat{\mathcal{M}}$. In general for a triangle σ, a vertex p, and an edge e of $\mathcal{M}$, the corresponding entity in $\hat{\mathcal{M}}$ is denoted with $\hat{\sigma}$, $\hat{p}$, $\hat{e}$, respectively. An entity in $\mathbb{E}^2$ / $\mathbb{E}^3$ will be denoted with o / $\hat{o}$; if o and $\hat{o}$ are used in the same context, we mean that o is the projection of $\hat{o}$ onto the xy-plane.

Our implementations use the explicit data structure described in Section 6.3.1 in the variant without adjacency information. The global static, global dynamic, and local static algorithms have been implemented. Adjacency relations for the output triangle mesh (if necessary for the specific operation) are determined through incremental reconstruction during the DAG traversal, as explained in Section 6.2.3. We have also implemented the compressed version of the MC structure for Delaunay triangulations (see Section 6.4.2.1).

8.2.1 Construction of MCs for Terrains

In order to support the construction of Multi-Complexes for terrains, we have designed and developed a number of history generators based on Delaunay triangulations. These methods have been described in Section 7.5, where experimental results have been discussed as well.

When representing terrain, it is often desirable to maintain certain known surface characteristics (e.g., lineal features such as rivers, ridges, isolines, etc.) in such a way that they are represented at different resolutions in the Multi-Complex. In Section 8.2.1.1, we outline a method that allows that.

In Section 8.2.1.2, we describe an algorithm that builds a set of MCs matching along their boundaries; such algorithm is the basis for the construction of modular sets of MCs which can be encoded on disk into the data structure presented in Section 6.5.

8.2.1.1 Construction of an MC that Preserves Lineal Features

We describe a method which permits to build a terrain MC that incorporates a multiresolution description of lineal features. For such a purpose, we modify coarsening history generators based on vertex removal (described in Section 7.5) according to the following guidelines.

We start from a triangulation Σ of the domain conforming to the lineal features. Edges representing (parts of) lineal features will be called *feature edges* in the following.

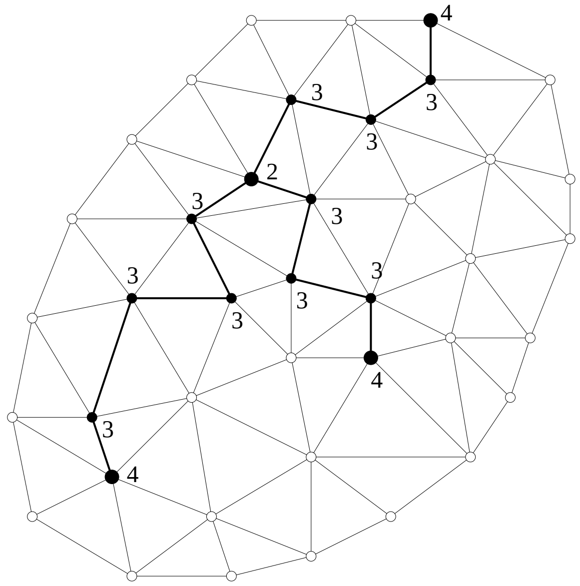


Figure 8.1: A triangulation with lineal features; feature edges are thick. Vertices of type 1 are shown in white; vertices of types 2, 3 and 4 are shown in black and labelled with the corresponding type.

The vertices of the current triangulation can be subdivided into the following classes (see Figure 8.1):

1. vertices that are not endpoints of any feature edge;

2. vertices that are endpoints of three or more feature edges.

3. vertices that are endpoints of exactly two feature edges;

4. vertices that are endpoints of exactly one feature edge;

Vertices of type 1 are treated as in the standard case. Vertices of type 2 cannot be eliminated. When removing a vertex p of type 3, the new triangulation *must* contain the segment p_1p_2, joining the other endpoints p_1 and p_2 of the two feature edges incident at p, and p_1p_2 becomes a feature edge. It is possible that p_1p_2 intersects some other feature edge: in this case, v cannot be removed. When removing a vertex p of type 4, simply one feature edge (namely, the one incident at p) disappears. In this case, a feature line is shortened.

For vertices of type 3 and 4, error evaluation (in variants which remove vertices based on error criteria) must take into account both the field error and the horizontal error (warping) introduced with respect to the original shape of the feature line.

Horizontal approximation errors are estimated as follows. Let L be a lineal feature in the initial triangulation at full resolution, and let L' be the current coarsened approximation of L. Let

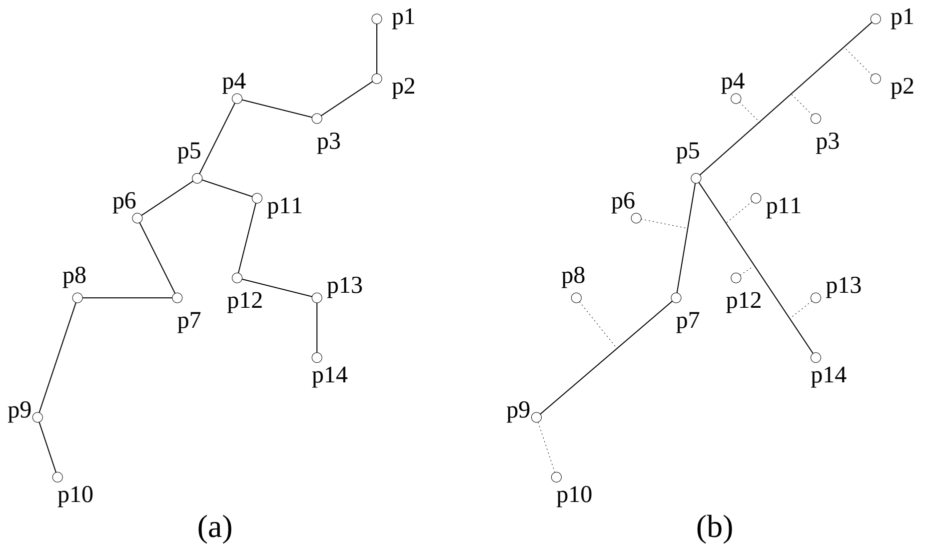


Figure 8.2: (a) A lineal feature, and (b) an approximation of it, with the removed points associated with each segment for the estimation of horizontal error.

p_1p_2 be a segment of L'. Intuitively, segment p_1p_2 approximates the polyline of L containing all the removed vertices of L between p_1 and p_2, and all the removed vertices of L beyond p_i (for $i = 1, 2$), if p_i is an endpoint of L' and not of L. We denote with Q the above set of removed points. The horizontal error of edge p_1p_2 is equal to the maximum distance of a point $q \in Q$ from p_1p_2 (see Figure 8.2).

During the iterative process of vertex removal, we keep track of the set Q associated with each segment of the current chain L' and use such sets to update approximation errors. When a vertex p of type 3 is removed, the new feature edge inherits the union of the two sets associated with the two disappearing feature edges incident at p, plus p. When removing a vertex p of type 4, the set associated with the disappearing segment, plus p, is added to the feature edge incident in the disappearing segment.

The condition that avoids removing a vertex of type 2 when the new feature edge intersects some existing feature edge limits the possibility of coarsening the TIN: it is possible to arrive at a situation where no vertex can be eliminated without getting some intersection. A possible solution is associating a priority with each lineal feature. At certain fixed stages during the coarsening process, all feature edges belonging to lineal features with the lowest priority become standard, non-feature edges. In cartography there are precise conventions which relate the approximation error and the density of lineal features in a terrain map, which can be applied in this context.

The method described here is under implementation.

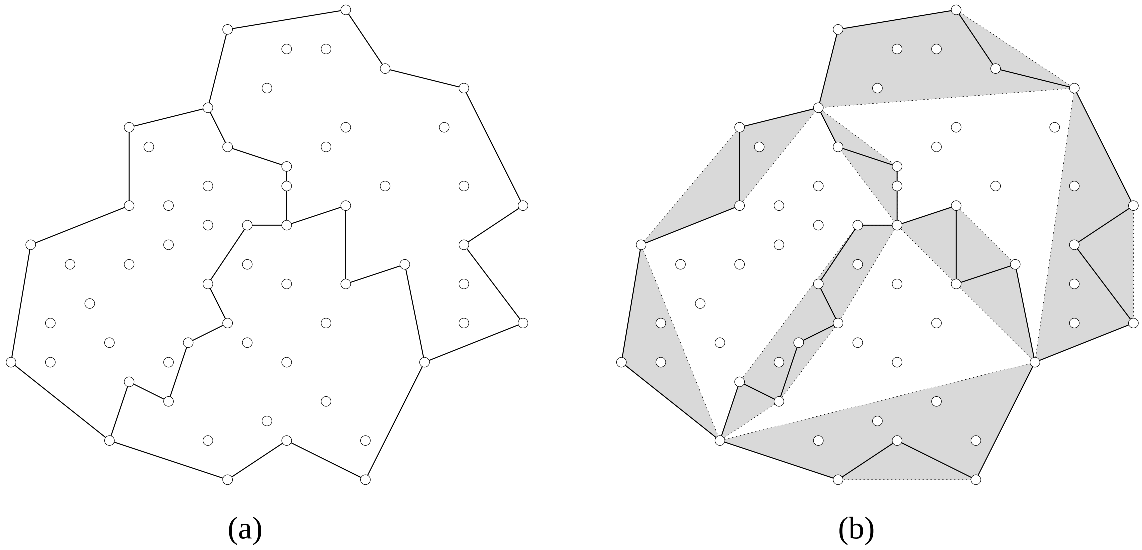


Figure 8.3: (a) An admissible data partition (circles are the points of set S). (b) A collection of chains (the convex hull of each chain is shown).

8.2.1.2 Building a Collection of Matching MCs

We describe a method for constructing a set of matching MCs suitable to be managed with the disk-based data structure described in Section 6.5. This method has been fully developed (see [BM98]) and we have started its implementation.

We assume that our initial data are a set S of scattered points in $\mathbb{E}^2$, with associated elevations, plus a *partition of the data* into areas called *regions*. For each region an individual MC will be built. We operate in two phases:

1. first, we build one-dimensional MCs for the boundaries of regions;
2. next, we build the individual two-dimensional MCs independently for each region by using such boundaries as constraints.

Before describing phases 1 and 2, we state the properties that the given partition of the data must satisfy. We assume that such partition is given; therefore, here we do not address the problem of computing an admissible partition.

Such partition is a two-dimensional complex Γ with vertices at a subset S' of S. We call *regions* the 2-cells of Γ; each region is a simple polygon.

Each region contains a subset of S in its interior. We impose that each point of S must be either a vertex of Γ or it must lie in the interior of a region of Γ. This prevents a point of S from lying in the interior of an edge of Γ or outside the carrier of Γ. Figure 8.3 (a) shows an

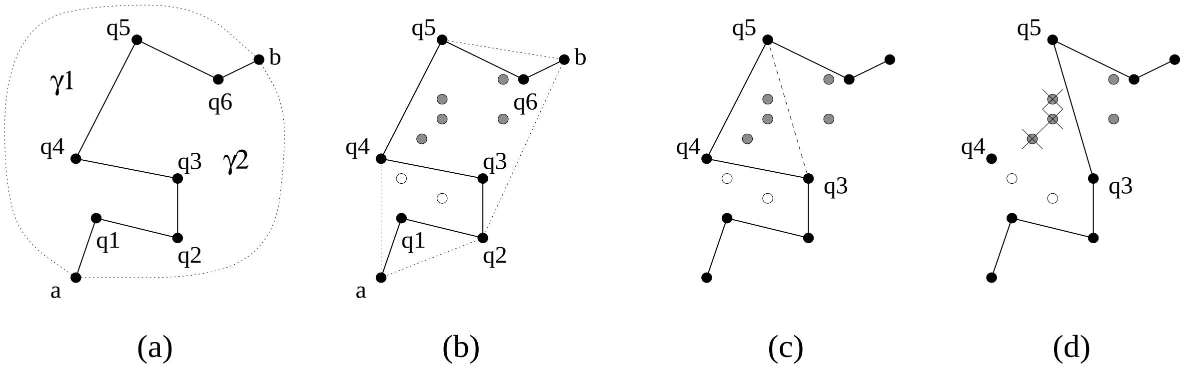


Figure 8.4: (a) The chain to be coarsened. (b) The two sets of points lying in its convex hull (S_1 shown in white and S_2 in grey). (c) When removing vertex q_4, the three points in triangle $q_3q_4q_5$ are captured. (d) The resulting chain after having removed q_4 (captured points are marked with crosses).

admissible partition for a data set S.

Generation of the one-dimensional MCs for the boundary We group the edges of Γ into *chains* satisfying the following constraints:

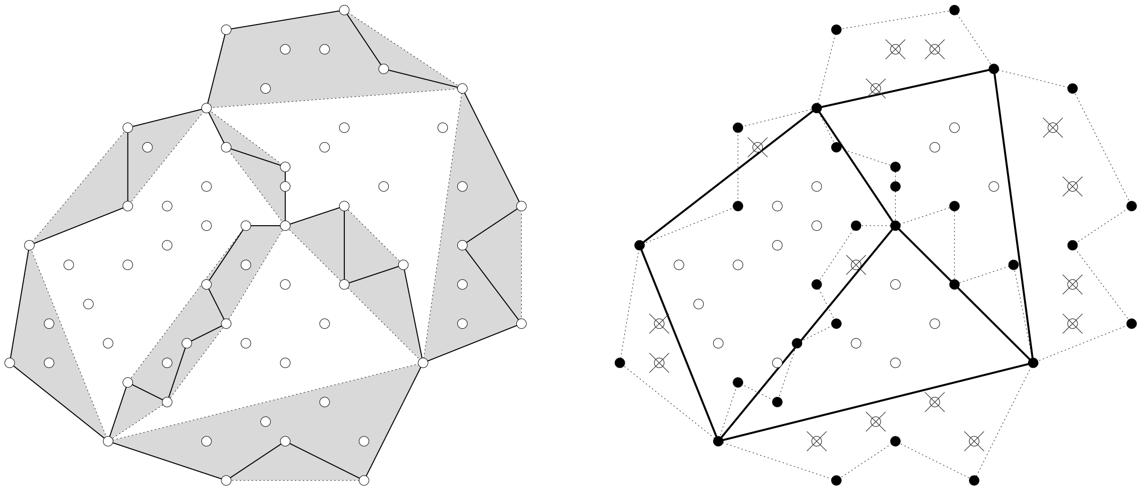
1. each chain is a connected sequence of edges;
2. two edges e_1 and e_2 belong to the same chain only if the two regions incident in e_1 are the same as the two regions incident in e_2;
3. no vertex of a chain is inside the convex hull of the vertices of another chain;
4. the boundary of each region consists of at least three chains.

Figure 8.3 (b) shows a possible way of grouping the edges of the partition of Figure 8.3 (a) into chains.

We build a one-dimensional MC for each chain. All the MCs built for the chains bounding a region γ will be used as constraints when building the two-dimensional MC for γ.

The input of this phase is a chain L, shared by two regions γ_1 and γ_2 of Γ, and the two subsets S_1 and S_2 of points of S which lie inside the convex hull of the vertices of L and belong to the interior of γ_1 and γ_2, respectively (see Figure 8.4 (b)).

We build the one-dimensional MC for L through a coarsening process that iteratively removes all the interior vertices of the chain, leaving just its two endpoints. The coarsening process is driven by the error of the current chain with respect to the initial one. At each step, the vertex causing the minimum error increase is eliminated.



(a) (b)

Figure 8.5: (a) The chains of an admissible data partition, with their convex hulls shown. (b) The final situation of boundary coarsening, where every chain has become a single segment; black circles denote the vertices of the original chains, crosses denote captured points. Note that each captured point lies on the opposite side of the coarsened chain to that of the full-resolution chain in (a).

Approximation errors are estimated as described for lineal terrain features in Section 8.2.1.1. Note that, unlike the endpoints of a lineal feature, the endpoints of a chain are never removed; thus, the set Q of points associated with a segment q_1q_2 is the set of removed points lying between q_1 and q_2 in the original chain.

Associated with each interior vertex q of a chain (i.e., with each vertex q candidate for removal), we maintain the error that would be introduced by removing q, computed by simulating the removal of q, and considering the error of the resulting segment.

Since each step of the chain coarsening algorithm replaces two segments with one segment, then the corresponding (reversed) refinement pattern C in the resulting MC contains one segment in $\Pi_1(C)$, and two segments in $\Pi_2(C)$. Thus, C will have exactly one parent and two children, and the resulting one-dimensional MC will have the structure of a binary tree. To avoid confusion with the two-dimensional MCs that will be constructed in phase 2, we call *binary trees* the one-dimensional MCs constructed in this phase.

While coarsening chain L, the points of S_1 and S_2 may change their relative position with respect to the chain: some points on one side of the chain may switch to the opposite side. We keep track of such points, since they must receive a special treatment when building the two-dimensional MCs for γ_1 and γ_2.

Let us consider a generic step of the coarsening process, in which a vertex q_i is removed and its two incident segments $q_{i-1}q_i$ and q_iq_{i+1} are replaced by segment $q_{i-1}q_{i+1}$. Then, all the points internal to triangle $q_{i-1}q_iq_{i+1}$ switch sides with respect to the chain. We say that such points are *captured* by the modification pattern corresponding to the removal of q_i (see Figure 8.4 (d), where $i = 4$). Since the vertices of a chain cannot be inside the convex hull of other chains, only interior points of γ_1 and γ_2 (more precisely, points of S_1 and S_2) may be captured while coarsening L; points lying inside other regions, or vertices of other chains, cannot be captured.

The output of the coarsening process for chain L is a binary tree, where the root represents the creation of the segment joining the two endpoints of L, and each other node represents the insertion of a vertex in the interior of the chain. Moreover, each node C stores an approximation error, equal to the error of the segment forming $\Pi_1(C)$, and a (possibly empty) list of captured points with the identifier of either γ_1 or γ_2 (depending which of the two regions owns the captured points).

The process of history generation for the binary tree for chain L can be summarized as follows:

1. for every interior vertex q of L, compute the error that would be introduced by removing q;
2. while the current chain has more than two vertices:
   a. choose the vertex q corresponding to the minimum error;
   b. check whether removing q causes a self-intersection of the chain; if this happens, do not remove q: choose the next vertex of minimum error, and resume from 2.b;
   c. compute the set of data points captured by the modification corresponding to the removal of q;
   d. record the modification pattern defined by the removal of q, the set of captured points, and the identifier of the region (among the two regions shared by the chain) to which the captured points belong; remove q;
   e. recompute the errors for the two old neighbors of q.

When a vertex q cannot be removed in step 2.b, the removal of q is not forbidden but just delayed: q will be reconsidered again after having removed some other vertex of the chain.

Figure 8.5 shows the final situation after having coarsened all chains to a single segment.

Generation of the MCs for each region Inside each region γ of Γ, we build a two-dimensional MC through a refinement technique. For convenience, we use a Delaunay triangulation (but other criteria could be used as well). We progressively refine both the interior of γ and its boundary, based on an error-driven criterion (an extension of Variant BU2). The points to be inserted for boundary refinement are the ones pre-computed and stored in the binary trees of the chains bounding γ. Since horizontal errors of boundary points and field errors of interior points are not homogeneous quantities, an appropriate factor of conversion is used to compare them; without loss of generality, here we assume that the conversion factor is equal to one.

Let S_γ be the subset of data points internal to γ, $L_1, \ldots, L_n$ be the chains forming its boundary, and, for each i, let $\mathcal{M}'_i$ be the binary tree built for L_i. Each of $\mathcal{M}'_i$ has in its root a modification pattern of the form $(\emptyset, \{e_i\})$, where e_i is the straight-line segment joining the two endpoints of the corresponding chain L_i.

The algorithm performs the following steps:

1. Build a constrained triangulation Σ of the polygon bounded by the root segments of all $\mathcal{M}'_i$'s. Note that Σ contains at least one triangle since each region has at least three boundary chains.
2. Start with a set S_{int} of candidate interior points obtained by subtracting from S_γ the sets of points that are captured by the modification patterns of some binary tree among $\mathcal{M}'_1, \ldots \mathcal{M}'_n$, and with a set S_{bound} of candidate boundary points formed by the points inserted in the modification patterns which are children of the roots of the binary trees (see Figure 8.6 (c)).
3. Compute the field error at each candidate point of S_{int}; note that candidates lie inside the domain of the current triangulation Σ, and thus a triangle containing them (either inside, or on its boundary) always exists. The error of candidate points of S_{bound} is already known: it has been computed during phase 1 and stored in the corresponding tree node.
4. While at least one of sets S_{int} and S_{bound} is not empty:
   - if S_{int} is not empty, let p be the point of maximum error in S_{int}; if S_{bound} is not empty, q be the point of maximum error in S_{bound};
   - if q does not exist, or the error of p is larger than that of q, then update Σ by inserting point p;
   - otherwise, update Σ by inserting point q along its boundary as specified in the corresponding binary tree node C; if the captured points of C belong to γ, then add them to S_{int}; if C is not a leaf, then add the points corresponding to the two children of C to S_{bound} (see Figure 8.6 (d)).
   - (re)compute the approximation error of all candidate points that lie in the interior of a newly created triangle (such points include the freed captured points).

The constrained Delaunay triangulation is updated through the incremental algorithm of De Floriani and Puppo [DFP92a]. Updating Σ after the insertion of a new interior point p is a standard point insertion in a CDT. Updating Σ after the insertion of a new boundary point q

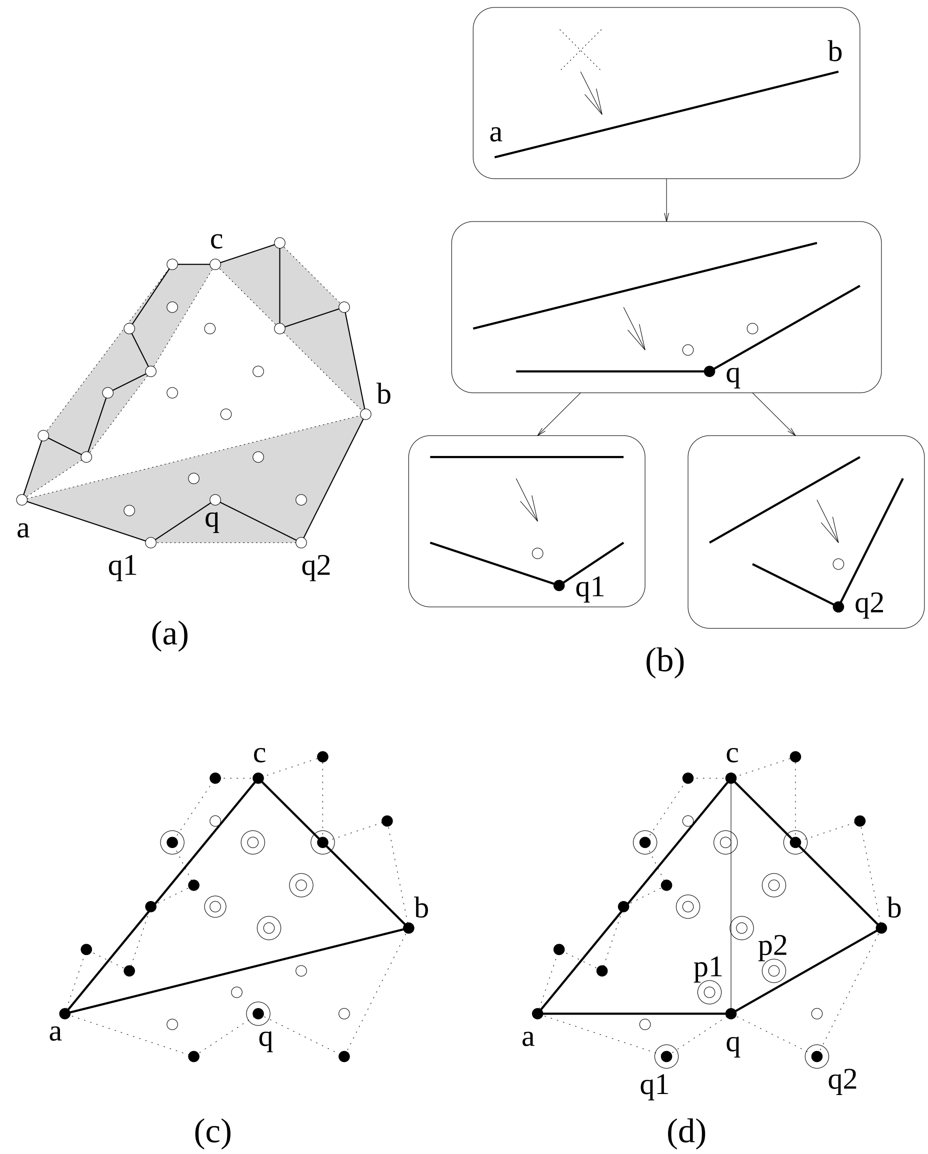


Figure 8.6: (a) One of the regions γ of the partition of Figure 8.5, its boundary chains and their convex hulls. (b) The one-dimensional MC of the bottom chain of γ; the points of γ captured by each modification pattern are shown in white. (c) The initial situation for building an MC inside γ: the interior candidate points of S_{int} and the boundary candidate points of S_{bound} are shown circled; other interior points cannot be used to refine γ yet, because they fall outside the current domain; the boundary candidate point for the bottom chain is q; the other two chains have one boundary candidate point each. (d) When the bottom chain is refined by inserting vertex q, points p_1 and p_2 are moved from $S_\gamma \setminus S_{int}$ to S_{int}, and the boundary candidate point q is replaced with q_1 and q_2.

involves removing one constraint segment e and replacing it with two new constraints e_1 and e_2. We start with the triangulation formed by Σ plus the triangle σ bounded by edges e, e_1 and e_2 and we optimize it to a Delaunay one through edge flipping.

During the main loop of the algorithm (Step 4), the set S_{bound} of interior candidate points may become empty. This is either because all points of S_γ have been inserted in the current triangulation or because all the remaining points of S_γ cannot be inserted yet, since they are captured by some one-dimensional modification pattern that has not yet been performed. In the latter case, while continuing boundary refinement, at some iteration we will perform a modification pattern having a non-empty set of captured points, thus getting some new interior candidates.

Comparison with Other Methods Other methods have been proposed for managing a multiresolution terrain model consisting of parts that are stored on disk separately and can be combined on-line [Hop98b, Paj98].

In [Hop98b], the high-level organization is a quadtree; each quadtree block contains a Progressive Mesh whose coarse mesh is the same as one quadrant of the full-resolution mesh of its parent. The whole model is built bottom-up by iteratively applying edge collapses to the full-resolution mesh of each block. In order to avoid the creation of dependency links between collapses performed in different blocks, the coarsening process inside each block is forced to preserve the boundary vertices of the mesh.

Pajarola [Paj98] imposes a grid over the domain and builds a hierarchy of right triangles inside each rectangle of the grid. Dependency links occurring across adjacent rectangles are tracked easily by exploiting the regular pattern of the hierarchy.

Our method is more general since we do not impose any specific shape for our regions; thus, we can generate partitions with a meaning for the application (e.g., regions for which individual MCs are built can be drainage basins, administrative units, etc.). We manage dependencies across regions in a general way, whereas the method used in [Paj98] is ad-hoc for fixed-pattern subdivisions and the one in [Hop98b] forces the delay of coarsening actions that would lead to inter-block dependencies. Moreover, our history generation strategy is completely error-driven and therefore is expected to achieve a better compression power.

8.2.2 Operations on Terrain MCs

In this section, we address the implementation of *terrain operations on the Multi-Complex.* We have considered the following operations as cases of study:

- *basic queries* for focusing the attention on a certain portion of the domain, e.g., inside a rectangular box (usually called a windowing query), along a line, or at a point (usually called a point location query);

- operations for the support of *interactive visualization* of a terrain;
- testing the *mutual visibility of two points*;
- computing the *viewshed* of a terrain, i.e., the portions of the surface visible from a given viewpoint;
- computation of *contour lines*.

Such operations are obtained on a Multi-Complex as instances either of the local query or of the global one with suitable parameters. In the following subsections, we describe the operations in detail and show experimental results where already available.

8.2.2.1 Queries in the Domain

We have implemented a number of basic queries on a terrain MC using focus sets in $\mathbb{E}^2$ on the Multi-Complex $\mathcal{M}$ embedded in $\mathbb{E}^2$. This is typically a conservative MC because in most terrain datasets the carrier is a rectangle (like a sheet map) and is the same at any resolution. After the result of the query has been computed as a plane triangulation Σ, it can be mapped into a triangle mesh in $\mathbb{E}^3$ by using the field values.

Basic queries permit to determine the configuration of elevation in a certain area of interest. Some examples of basic queries are:

- *point location*: the focus set is a point $p \in \mathbb{E}^2$; this query can be used to evaluate the terrain height at a given location.
- *windowing*: the focus set is a rectangular axis-parallel box in $\mathbb{E}^2$; this query can be used to select a TIN covering a sub-area of the whole;
- *range query*: this is a metric query asking for entities which are within distance d from a point p_0; we reduce it to an interference query with a focus set equal to the circle centered in p_0 with radius d; this query can be used to select a TIN covering the area around a given location.
- *line interference*: the focus set is a polygonal line in $\mathbb{E}^2$; this query can be used to find the terrain configuration along a road, a river, a land boundary, etc.

All above operations reduce to local queries on an MC. With each query, different resolution filters can be used. We have experimented with resolution filters induced by constant error threshold functions, and filters induced by error thresholds that increase with the distance from a fixed point (see also Section 8.2.2.2).

In point location and line interference queries, the active area is likely to be small with respect to the extension of the domain, thus a constant error threshold is reasonable.

Algorithm 5.5 (the local query algorithm) is appropriate if the user is not interested in getting a representation of the terrain outside the active area as well. Otherwise, the query can be answered by reducing it to a global query with a resolution filter restricted to the given focus set. In this case, the result is a TIN covering the whole domain and more refined in the part that interferes with the focus set. A dynamic algorithm may be appropriate if repeated queries are performed each with a slightly modified focus set (e.g., a windowing query with a moving window).

Figures 8.7 and 8.8 show the results of some queries answered as local and as global queries. Figure 8.9 shows the corresponding images. These results refer to an MC built on the same terrain dataset as that used in Section 6.4.4.

8.2.2.2 Interactive Terrain Visualization

For interactive terrain visualization, we consider a scenario in which the user can drive its viewpoint to fly over the terrain. In order to obtain an interactive frame rate, it is necessary to keep the overall number of triangles sent to a rendering device under a certain bound. At the same time, the visual appearance of the mesh that is rendered must not be perceptually dissimilar from the mesh at the maximum resolution.

The idea is extracting a TIN over the whole domain, but coarsening terrain portions which are either distant or outside the field of view. Thus, we adopt a global query with a resolution filter restricted to the view frustum; the basic (unrestricted) resolution filter is designed according to the following guidelines:

- resolution must be higher near the viewpoint and decrease with distance;
- resolution must be higher at the center of the field of view and decrease towards the sides.

While moving the viewpoint, the query parameters vary at each instant in time, and thus a dynamic algorithm is used.

As shown in Figure 8.10, let $\hat{v} = (x_v, y_v, z_v)$ be the current viewpoint, $\hat{t} = (x_t, y_t, z_t)$ be a point such that vector $\hat{t} - \hat{v}$ defines the current view direction, α and β be the horizontal and vertical view angles, respectively. The focus set of our query is the view frustum. Since it is an entity in $\mathbb{E}^3$, it applies to the MC $\hat{\mathcal{M}}$ embedded in $\mathbb{E}^3$, which is not conservative.

For the sake of simplicity, we assume that the observer flies near the terrain surface, the view direction is nearly-horizontal (i.e., $z_v \simeq z_t$), the vertical view angle β is large, and the near and far clipping planes are very close to and very far away from $\hat{v}$, respectively. Thus, instead of using a focus set equal to the view frustum in $\mathbb{E}^3$, we use a focus set in $\mathbb{E}^2$, equal to the

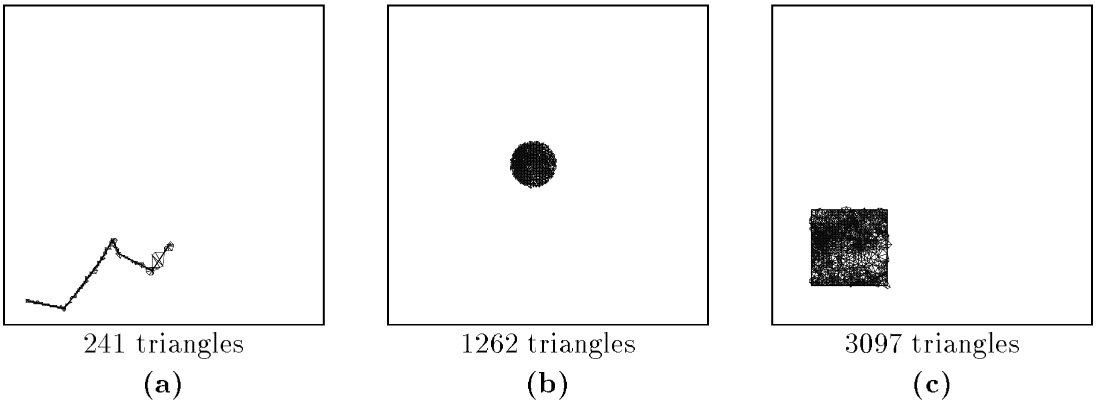


Figure 8.7: Results of (a) a line interference query, (b) a range query, and (c) a windowing query answered as local queries. The resolution filter is induced by a constant error threshold function. The local algorithm visits 3075 triangles in (a), 11160 in (b), and 5603 in (c).

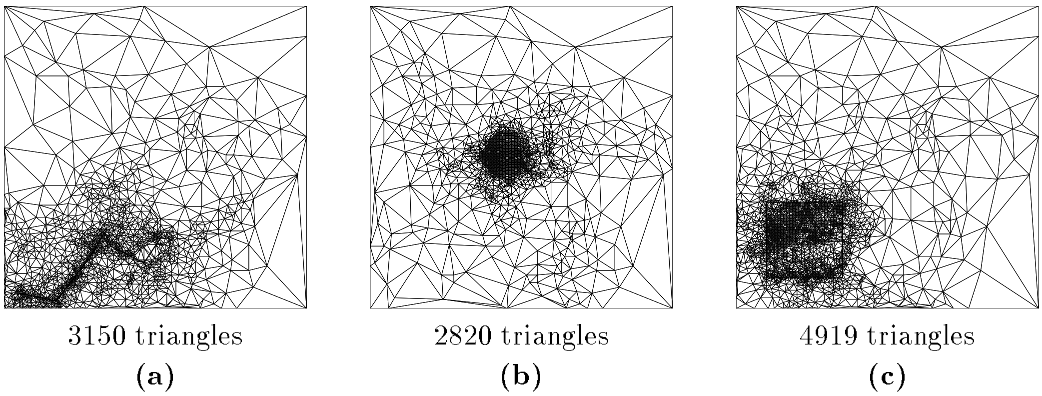


Figure 8.8: Results of (a) a line interference query, (b) a range query, and (c) a windowing query answered as global queries with a restricted resolution filter. The output TIN is the same as with the local algorithm (see Figure 8.7); the front TIN of the sub-MC visited by the basic algorithm is shown. The basic algorithm visits 14106 triangles in (a), 20845 triangles in (b), and 12036 triangles in (c).

two-dimensional sector with apex in v, central axis directed as vector $t - v$ and opening angle equal to α. Such two-dimensional focus set is applied to $\mathcal{M}$, which is a conservative MC. A triangle σ is active if an only if, for at least one vertex p_i of σ, the angle θ formed by p_i, v and t is smaller than α (see Figure 8.11 (a)).

Our basic resolution filter is induced by an error threshold function τ that has its minimum (equal to zero) in the viewpoint $\hat{v}$, and increases linearly with the distance from $\hat{v}$ and with the angular distance from the half-line outgoing from $\hat{v}$ in the view-direction. Again, for simplicity we define such threshold in $\mathbb{E}^2$, and apply it to the triangles of $\mathcal{M}$.

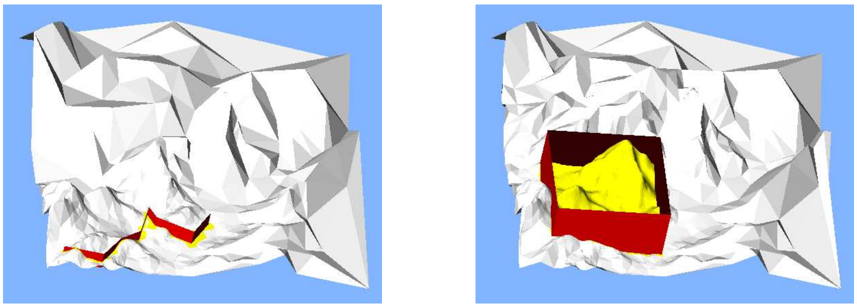


Figure 8.9: Three-dimensional views of the meshes shown in Figure 8.8 (a) and (c). Active triangles are yellow, non-active ones are white. The focus set (actually defined in $\mathbb{E}^2$) is shown here by extending vertical "walls" from it.

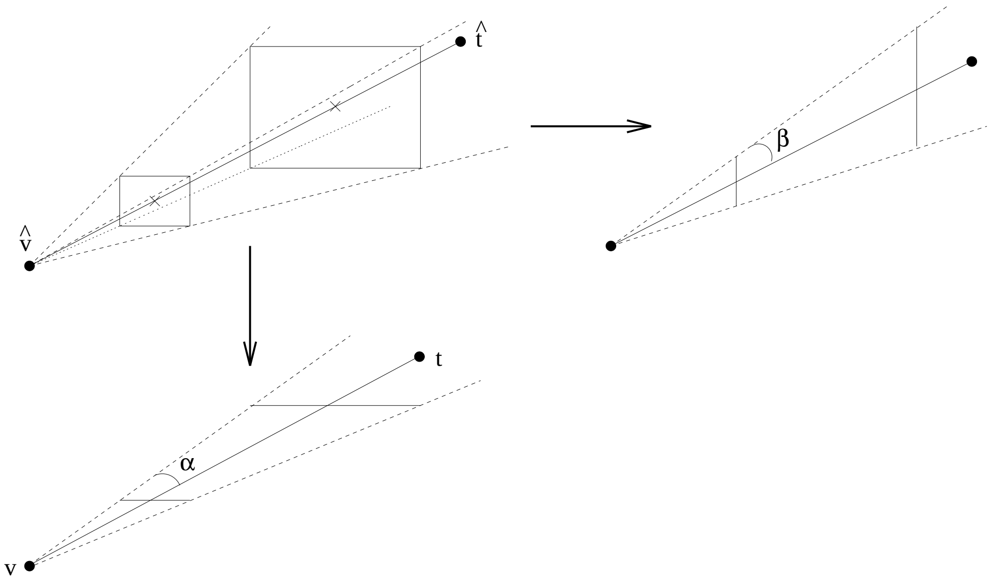


Figure 8.10: The view frustum and its projection onto a horizontal and a vertical plane.

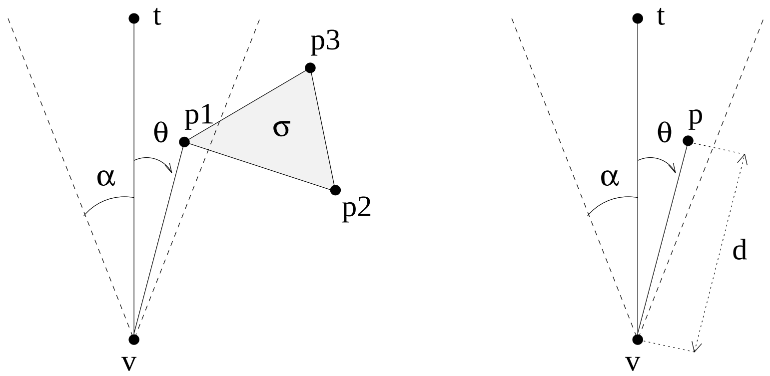
Given a point $p \in \mathbb{E}^2$, we denote with d the distance of p from v, and with θ the angle formed by p, v and t (see Figure 8.11 (b)). The error threshold function has the following expression:

$$\tau(p) = \begin{cases} c_{dist} d(1 - c_{ang}\theta) & \text{if } \theta < \alpha \\ +\infty & \text{otherwise} \end{cases}$$

where c_{dist} and c_{ang} are two positive constants.

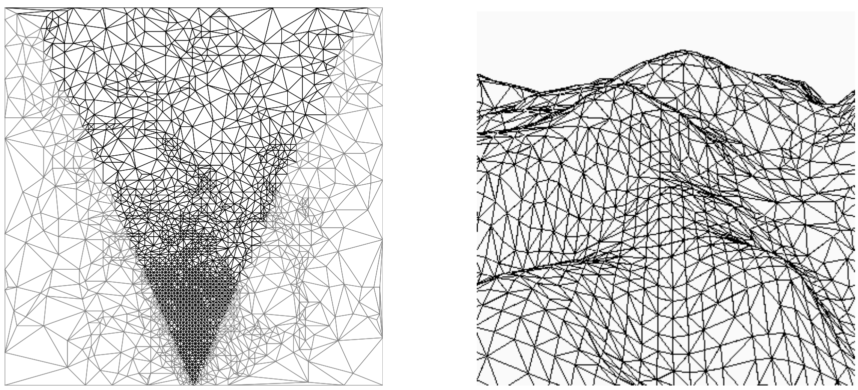
The resolution filter induced by τ is then restricted to an area of interest equal the focus set previously defined.

Experimental results have been shown in Section 5.7 in the context of the comparison of the



(a) **(b)**

Figure 8.11: (a) Testing whether triangle σ falls (at least partially) within the sector: in this case the answer is positive since the angle θ formed by vertex p_1 with the view direction is smaller than α. (b) Computing the error threshold at a point p: the distance d from v and the angle θ formed with the view direction are considered.



(a) **(b)**

Figure 8.12: (a) A TIN used for interactive terrain visualization, and (b) the corresponding view of the terrain.

global static, global dynamic and local extraction algorithms. Figure 8.12 shows a view of the terrain and the underlying TIN at variable resolution.

8.2.2.3 Point Visibility

Two points $\hat{p_1}, \hat{p_2} \in \mathbb{E}^3$ are mutually *visible* on a TIN $\hat{\Sigma}$ if and only if the interior of segment $p_1 p_2$ does not intersect or touch any triangle $\hat{\sigma}$ of $\hat{\Sigma}$ (see Figure 8.13).

The problem of testing a pair of points for mutual visibility on a TIN can be solved by reducing it to an interference query where the focus set is the straight-line segment $\hat{p_1}\hat{p_2}$. Such query is

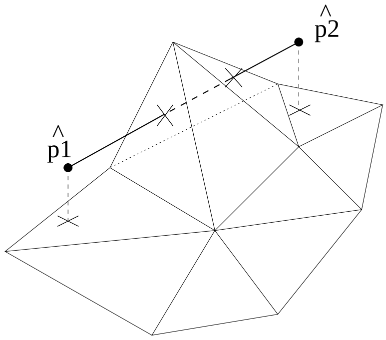


Figure 8.13: Points $\hat{p_1}$ and $\hat{p_2}$ are not mutually visible since the straight-line segment connecting them intersects the TIN.

applied to the triangle mesh $\hat{\Sigma}$ in $\mathbb{E}^3$ and its result is formed by the triangles $\hat{\sigma}$ that block the view from $\hat{p_1}$ to $\hat{p_2}$. The two points are mutually visible if and only if the result of the query is empty (i.e., segment $\hat{p_1}\hat{p_2}$ intersects no triangle).

For testing the visibility of two points on an MC, we perform a *local query* with the same focus set as above on the Multi-Complex $\hat{\mathcal{M}}$ in $\mathbb{E}^3$. In order to have the maximum precision in the answer, we use a resolution filter induced by a constant error threshold function identically equal to zero. The focus set applies to an MC which is not conservative since it is defined in three-dimensional space; thus, the query must take into account loosely active triangles. Note that the query is likely to select just few triangles on average. In this sense, the MC works as a spatial index. Results of experiments performed on the MC representing Mount Marcy, already used in Section 7.5, are reported in Figure 8.14.

8.2.2.4 Viewshed Analysis

Computing the *viewshed* of a given viewpoint $\hat{v}$ on a TIN $\hat{\Sigma}$ consists of determining the portions of each triangle of $\hat{\Sigma}$ that are visible from $\hat{v}$.

The problem of viewshed computation has been extensively studied for TINs, and for general three-dimensional scenes. Viewsheds can be computed from an MC by extracting first a TIN at the desired resolution, and then applying one of the existing viewshed algorithm for TINs [RS88, DFFJ$^+$89, KOS91, PJ92, OS92, BK92, DB93] (see also [DFM94b] for a survey).

The most popular viewshed algorithms traverse the TIN triangles in front-to-back order with respect to the given viewpoint and compute the visible and invisible portions of each triangle directly during such traversal. They exploit the fact that a triangle σ can only be obscured by triangles that precede it in the front-to-back order; thus, when σ is traversed, we already have all necessary information to determine the visibility situation of σ.

Viewshed computation is an expensive task since the worst-case space complexity of the

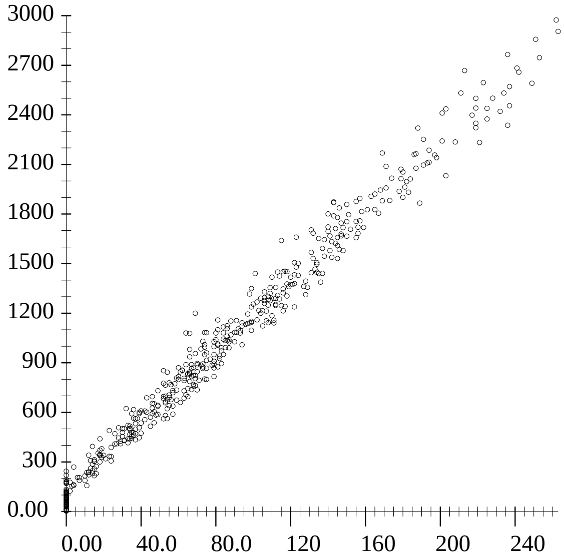


Figure 8.14: Result of 500 point visibility queries with pairs of points generated randomly. The horizontal axis reports the number of triangles blocking visibility (i.e., intersecting the straight-line segment which joins the two points in $\mathbb{E}^3$ and, thus, reported by the query); the vertical axis reports the number of triangles visited by the local algorithm. Note that the ratio of the two quantities is about constant.

viewshed can be as high as $\Omega(n^2)$ for a TIN with n vertices [CS89]. Thus, we can save time and computation resources by computing it on a terrain representation at a reduced resolution. In general, visibility computations are sensitive to elevation errors near the viewpoint, therefore it is recommended to ensure a sufficient accuracy near the viewpoint, and decrease it progressively with distance [FG90].

As in the case of interactive terrain visualization, we define our resolution filter in $\mathbb{E}^2$. Given a generic point $p \in \mathbb{E}^2$, we denote with d the distance of p from the viewpoint v. Our resolution filter is induced by an error threshold function increasing linearly with d, according to the following expression:

$$\tau(p) = c_{dist} d$$

where c_{dist} is a positive constant.

Since a representation of the whole terrain is needed, we use a global query. We use the basic algorithm because typically the viewshed is computed for a single viewpoint or for a discrete set of viewpoints at a certain distance from each other; thus, the extracted meshes will be typically different and a dynamic approach not convenient. Adjacency information for the extracted TIN should be generated because viewshed algorithms need them.

The same approach can be used to compute other visibility structures on a terrain, for instance, the *horizon* [CS89], which corresponds to the distal boundary of the visible area from a viewpoint in all directions.

A module for computing viewsheds from TINs extracted from a Multi-Complex is currently under implementation.

In our previous work [DFM94a, DFM97], we have extended front-to-back viewshed algorithms to hierarchical TINs (HTINs, see Section 3.3.2.4), yielding algorithms that compute the visible portions of every extracted triangle *while* extracting the mesh at the required accuracy from the multiresolution model. Such algorithms exploit the nested structure of an HTIN, since they are based on the fact that each branch of the tree is traversed independently and the front-to-back order of the triangles of a mesh at coarse level is consistent with the front-to-back order of the triangles of a more refined mesh. These properties do not hold in a generic MC; thus, a similar extension of viewshed algorithms does not seem to be possible.

8.2.2.5 Contour Lines

Given a scalar field $\phi : \Omega \longrightarrow \mathbb{E}$ and a value $h \in \mathbb{E}$, the *contour* of ϕ at value h is the intersection of the graph of ϕ in $\mathbb{E}^3$ with the plane of equation $z = h$. With the exception of special cases in which ϕ is constant over some two-dimensional subsets of Ω, in general, the contour consists of a collection of closed and open lines.

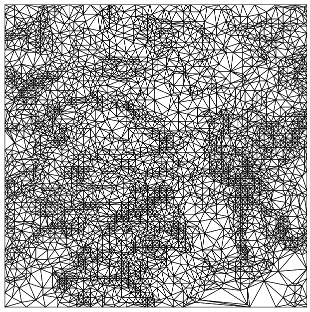
If ϕ is approximated by a triangulation of Ω and a piecewise linear function (as in the case of a TIN), then the graph of ϕ in $\mathbb{E}^3$ is approximated with a triangle mesh, and the contour consists of a set of closed or open polygonal chains. The projections of several contours (corresponding to a sequence of field values) onto the xy-plane form a *contour map*.

A Multi-Complex can help the generation of contour maps for a terrain in different ways. Given a Multi-complex, we can find the triangles that contribute to the contour map by performing a local query where the focus set is the set of planes in $\mathbb{E}^3$ at the given heights and the resolution filter is induced by a constant error threshold function, corresponding to the desired accuracy. In this sense, the MC plays the role of a spatial index to select just the relevant triangles.

Next, the contour lines are determined by computing the intersection segments of the triangles returned by the query with the horizontal planes at the given heights. It is important to generate adjacency relations among the triangles of the output mesh of the query since this information permits to find pairs of contour segments which must be connected into chains (adjacent contour segments belong to adjacent triangles).

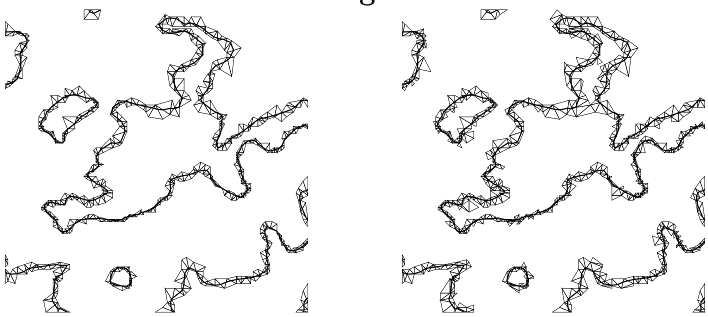
In the above approach, we have assumed that we are not interested in the triangles out of the contour lines (thus, we have used a local query). As an alternative, we can perform a global query with a resolution filter restricted to the area of interest defined by the planes at the given heights. In both the local and the global case, the queried MC is not conservative (since it is the one embedded in $\mathbb{E}^3$), and thus the query must be aswered by referring to loosely active

TIN at a uniform accuracy



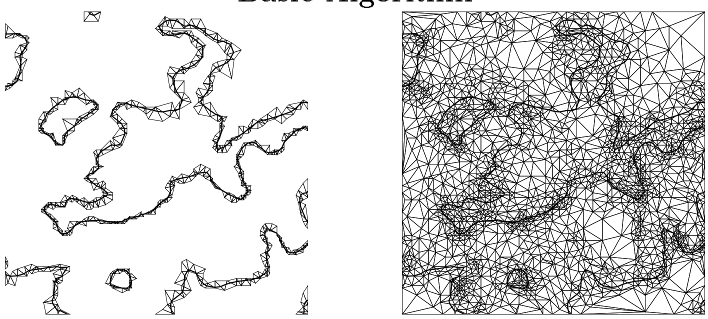
8280 triangles, 35396 triangles in the visited sub-MC.

Local Algorithm



6236 triangles in the visited pseudo-sub-MC, 940 triangles in its front TIN, 767 active.

Basic Algorithm



12903 triangles in the visited sub-MC, 4177 triangles in its front TIN, 769 active.

Figure 8.15: Isolines extracted from the MC for Mount Marcy for a constant error threshold equal to 5.

cells, and a loose restriction of the resolution filter, respectively. Results from both approaches are shown in Figure 8.15. Note that, in the result of the global query, triangles tend to have a bigger size outside areas intersected by the lines.

Another possible approach consists of building a one-dimensional Multi-Complex that represents contour maps at multiple resolutions for a given terrain, and then query such one-dimensional MC to obtain contour maps at any desired accuracy.

Such a one-dimensional MC can be obtained by applying an iterative process of line simplification to a contour map computed on the terrain at full resolution (the terrain model does not need to be an MC; it can be simply a TIN) for a dense sequence of heights. Here, the approximation errors are measured on the plane and refer to the shape of the contour lines.

While coarsening the lines, the algorithm must avoid intersections between different contour lines. Since such constraint may prevent coarsening the lines beyond a certain point, we can correlate the (horizontal) approximation error of the representation with the density of lines by saying that some contours (e.g., every odd contour) disappear when the approximation error becomes larger than a certain threshold; this enables further coarsening of the remaining lines.

Representations at variable resolutions can be extracted from the one-dimensional MC of the contour map by selecting the desired horizontal error and the desired heights among the available ones.

Note that the contour maps extracted from a one-dimensional MC will not, in general, coincide with those computed on a TIN extracted from the two-dimensional MC representing the same terrain. A possible way to overcome this problem, i.e., to make sure that contour lines extracted from the terrain MC at a lower resolution resemble the simplified contour lines obtained with a line simplification algorithm, consists of building a terrain MC that contains a multiresolution representation of the contour lines by using the approach outlined in Section 8.2.1.1.

8.3 Approximation of Surfaces

For representing surfaces in $\mathbb{E}^3$, we use two-dimensional simplicial Multi-Complexes in $\mathbb{E}^3$. Error evaluation is based on the Hausdorff distance (see Section 2.4.3).

We provide algorithms to build an MC for a surface $\mathcal{O}$ in the cases when an approximation of $\mathcal{O}$ is given as a *dense triangle mesh* (e.g., acquired from a three-dimensional scanner), and when an analytical description of $\mathcal{O}$ is given in the form of a *parametric surface* (i.e., a collection of parametric patches) such as the ones produced by CAD systems. Preliminary work appeared in [DFMP96a, DFMP97b, DFMP97a, DFMP97c]. The construction algorithms for the two cases will be described in Sections 8.3.1 and 8.3.2.

As far as operations are concerned, by now we have implemented some simple queries, using:

- focus sets (used either in local queries or for restricting resolution filters in global queries) consisting of axis-parallel boxes (as the one used in Section 5.7), or spheres;
- resolution filters induced by constant error threshold functions or by error threshold functions increasing with distance from a reference point; we have experimented error threshold functions increasing linearly, quadratically, and exponentially with distance.

- resolution filters based on the size of triangles, regardless of their approximation errors; our measure for the size of a triangle is the radius of its circumscribed circle.

Box and sphere focus sets are useful to select an area of interest, like a magic lens, and they are applied to MCs which are not conservative. The distance-dependent filter is useful to adapt accuracy according to the distance from a viewpoint, as explained for terrains. The triangle size filter allows getting a representation that corresponds to a uniform density of points on the surface.

8.3.1 MC Construction from Acquired Triangle Meshes

In order to build an MC for a surface given as a dense triangle mesh, it is sufficient to choose one of the mesh simplification algorithms existing in the literature and use it as a history generator. Although most existing algorithms do not guarantee a conflict-free or even a valid history (see Section 7.2), our experience confirms that this does not affect the consistency of the extracted meshes in practice.

We have implemented a history generator based on the coarsening algorithm developed by Ciampalini et al. [CCMS97] at the C.N.R. of Pisa (see Section 3.2.3). The MCs used to generate the experimental results for free-form surfaces shown in Section 5.7 have been built with such a generator.

This algorithm iteratively eliminates a vertex from a mesh and retriangulates the resulting hole. At each step, the vertex causing the least error increase is removed. This approach corresponds to Variant BU4 described for two-dimensional scalar fields in Section 7.5. The difference is that we evaluate the approximation error through the Hausdorff distance (rather than through the field distance), and we do not use the Delaunay criterion for retriangulating.

Variants of mesh coarsening corresponding to the other approaches presented for two-dimensional scalar fields in Section 7.5 (i.e., elimination of sets of independent vertices, elimination of vertices with a bounded degree) can be defined for surfaces as well. We have not yet implemented the versions for surfaces of Variants BU1-2-3, but we expect similar results about the quality of the Multi-Complexes produced as those shown in Section 7.5.

8.3.2 MC Construction from Parametric Boundary Models

In this subsection, we propose a new method for building a Multi-Complex from a parametric surface [DFMP96a, DFMP97c]. We recall from Section 2.2.3 that a parametric surface is a collection of parametric surface patches, such that any two patches may touch along their boundaries, but cannot intersect.

A triangle mesh approximating a parametric patch $\phi : \Omega \subseteq \mathbb{E}^2 \longrightarrow \mathbb{E}^3$ is obtained by defining first a triangulation Σ of a polygonal approximation of its domain Ω in parameter space, and then lifting the triangulation into physical space by replacing each vertex with its image through function ϕ (see Figure 2.21). In other words, we embed Σ into $\mathbb{E}^3$ through a piecewise-linear function f.

We measure the approximation error of a triangle mesh Σ approximating a parametric patch through an L_∞ *norm* (as done in [KS95]), exploiting the fact that the surface piecewise described by f over Ω and the surface described by ϕ over Ω can both be parametrized in the same space. The approximation error of a triangle $\sigma \in \Sigma$ takes into account the set P_σ of points $p \in \Omega$ such that $p \in \sigma$ and it is computed as the maximum distance of $\phi(p)$ from $f(p)$ over all $p \in P_\sigma$.

Since several parametric patches are glued together, a major issue is ensuring matching between the approximations of any pair of adjacent patches (e.g., avoiding cracks or unwanted intersections). This cannot be guaranteed if each patch is triangulated separately.

When constructing a single-resolution triangle mesh, we guarantee matching by imposing first boundary constraints and then using a constrained triangulation of the interior of the domain of each patch. In a *preprocessing phase*, we subdivide the trimming curves of each patch in such a way that each pair of neighboring patches share exactly the image of one or more trimming curves (see Figure 8.16 (a),(b),(c)). In other words, we transform the set of patches and their trimming curves into a cell complex. Next, the algorithm performs three steps:

1. create a polygonal approximation of the trimming curves shared by each pairs of neighboring patches in $\mathbb{E}^3$ (see Figure 8.16 (d));
2. back-project each polyline to the domain in $\mathbb{E}^2$ of the relevant patches and use the resulting straight-line segments to build a constrained triangulation of each parametric domain (see Figure 8.16 (e));
3. lift the individual triangulations into physical space, thus obtaining a triangle mesh for the whole surface (see Figure 8.16 (f)).

In an MC, matching between the discretizations of neighboring patches must be guaranteed at *any* resolution.

A straight-forward approach for the construction of an MC for a parametric surface consists of building first a triangle mesh representing it at a high accuracy, and then proceed as in the case of free-form surfaces given as dense triangle meshes (see Section 8.3.1). However, the need for high accuracy may lead to memory problems for carrying on the mesh simplification of all patches at the same time.

Here, we propose a method that processes the patches of a parametric surface one at a time (or, in some phases, in pairs). The method extends the method outlined above to a multiresolution

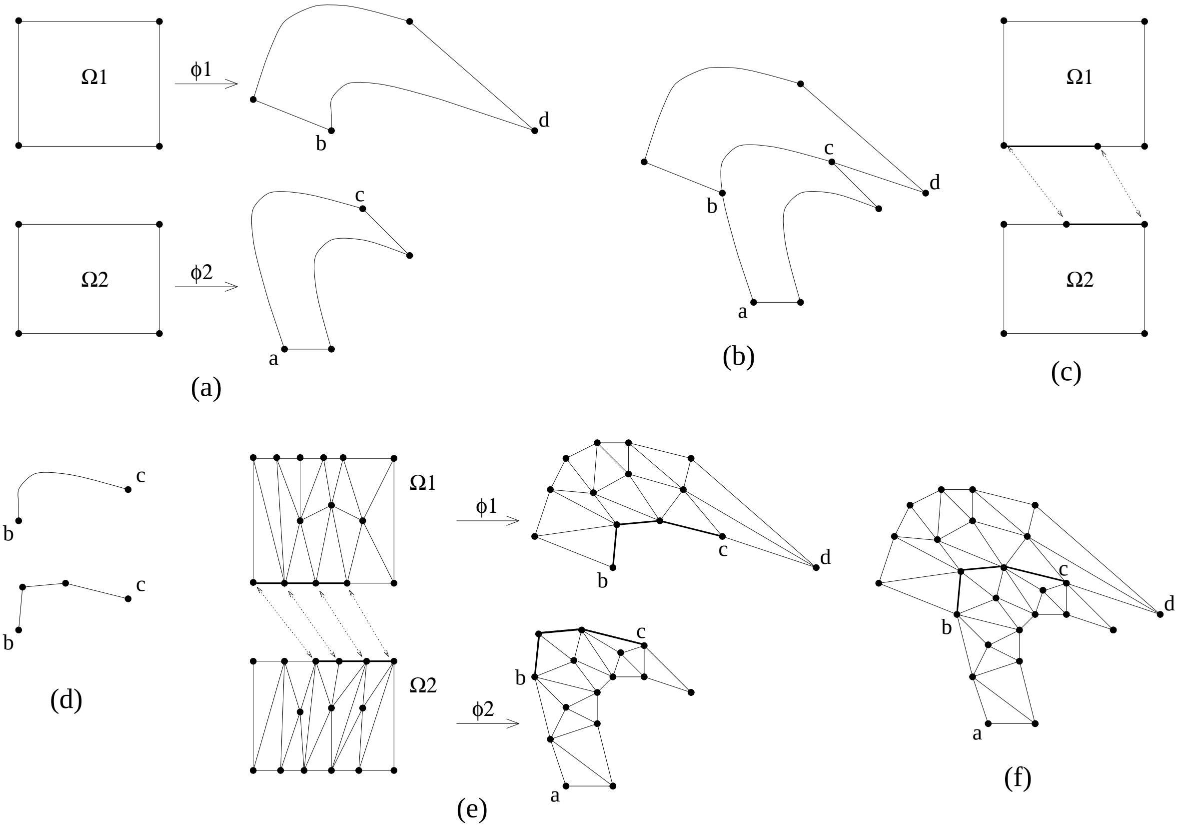


Figure 8.16: Construction of a triangle mesh for two neighboring parametric patches. (a) The two patches ϕ_1 and ϕ_2. (b) The images of the two patches touch along segment bc, which is the overlapping portion of two trimming curves ac and bd. (c) New points are inserted on the trimming curves so that bc is a trimming curve in both patches. (d) Construction of a polyline approximating bc. (e) Back projection of the polyline vertices in the parameter space of ϕ_1 and ϕ_2, and construction of a constrained triangulation of the two domains, which corresponds to a triangulation of the two surfaces. (f) The two triangulations match along the common polyline.

setting [DFMP97c]; the basic idea is similar to the one used for generating a matching set of terrain MCs (see Sections 6.5 and 8.2.1.2).

The preprocessing is the same as in the single-resolution case. The subsequent steps are:

1. build a one-dimensional MC for the trimming curves shared by each pair of neighboring patches in $\mathbb{E}^3$;

2. build a two-dimensional MC separately for the domain of each patch by using multiresolution representation of its trimming curves (back-projected into parametric space) as constraints;

3. project the resulting MC into physical space by replacing each vertex with its image and

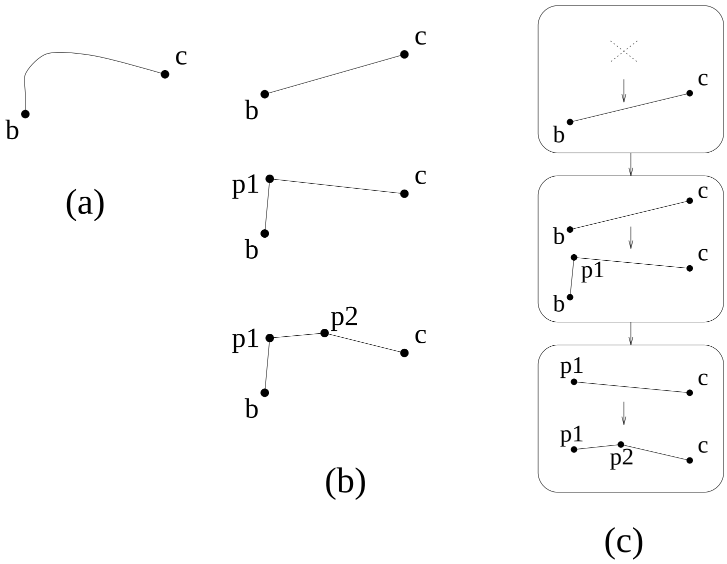


Figure 8.17: Construction of the one-dimensional MC for the common trimming curve of the two adjacent patches of Figure 8.16: (a) the given trimming curve in $\mathbb{E}^3$; (b) the successive steps of its approximation with a polyline; (c) the corresponding one-dimensional MC.

glue the individual MCs of neighboring patches at their common boundary edges, thus obtaining an MC for the whole surface.

Step 1 is shown in Figure 8.17. Given a trimming curve L in space, we start with an approximation of L consisting of one straight-line segment joining its two endpoints. Then, we iteratively select the point on L corresponding to the maximum error of the current piecewise-linear approximation and insert it. Let $p_1, \ldots, p_k$ be the current polygonal approximation. The approximation error of each segment $p_i p_{i+1}$ is estimated as the maximum distance from $p_i p_{i+1}$ of a point of L lying between p_i and p_j. Note that the error is measured in physical space. The insertion of a new point $q \in L$ lying between p_i and p_{i+1} replaces segment $p_i p_{i+1}$ by two segment $p_i q$ and $q p_{i+1}$. The resulting one-dimensional MC is a binary tree, as noticed for terrains in Section 8.2.1.2. Each node of the resulting binary tree (representing the split of one straight-line segment $p_i p_{i+1}$ by the insertion of a new vertex q) stores its approximation error, equal to the distance of q from $p_i p_{i+1}$.

Next, such binary tree is back-projected in the parameter spaces of the two patches sharing the trimming curve L. The binary tree of the trimming curves bounding a patch define an approximation of its domain with a polygonal region at multiple resolutions.

In Step 2, we define a two-dimensional simplicial MC for the domain of each patch separately while using the multiresolution representations of its trimming curves (built at Step 1) as constraints to ensure matching with adjacent patches. Such a two-dimensional MC is built through iterative vertex insertion based on an error-driven strategy.

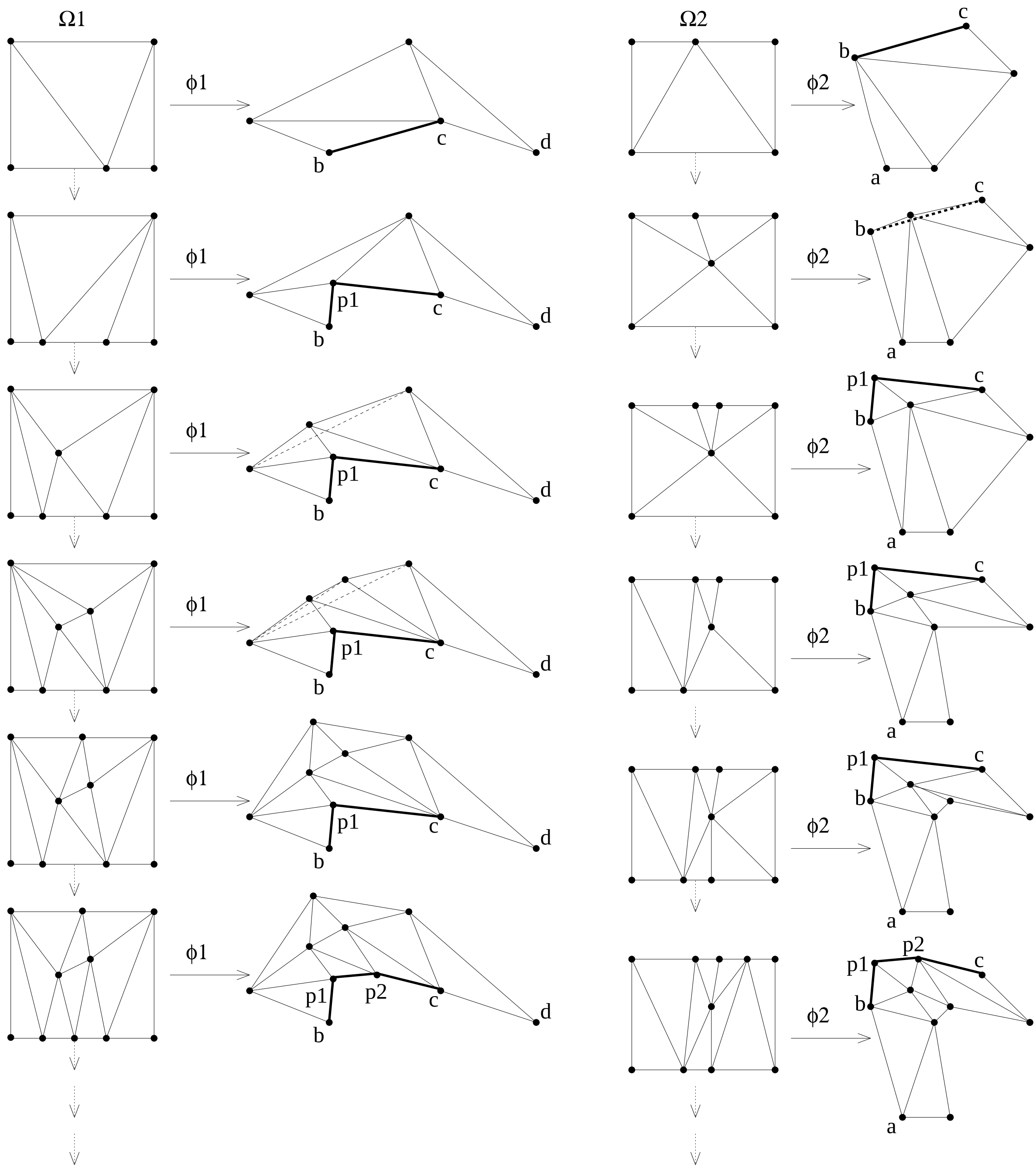


Figure 8.18: Iterative refinement of the interiors of the two patches ϕ_1 and ϕ_2 of Figure 8.16 using the one-dimensional MC of Figure 8.17 as constraints.

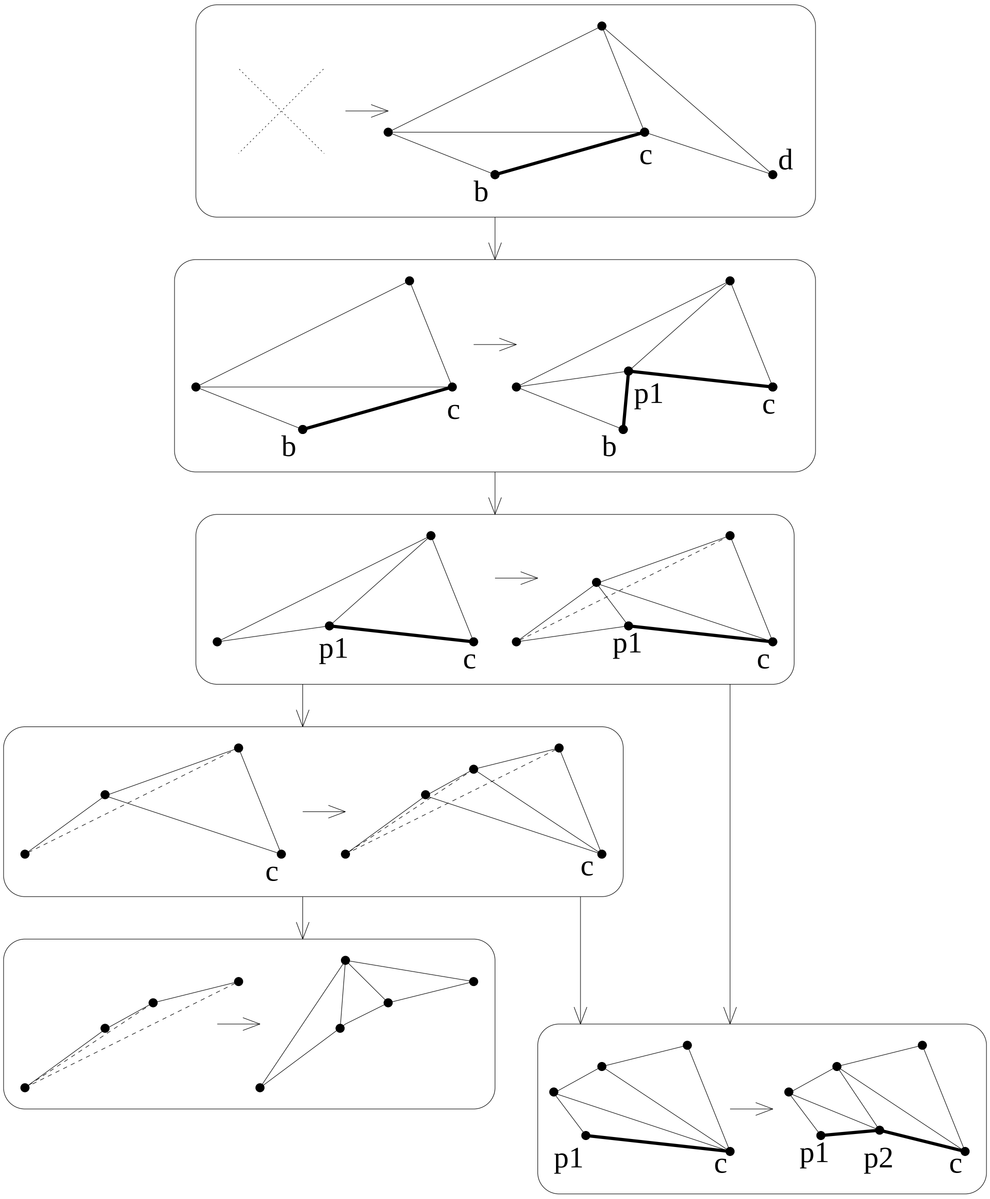


Figure 8.19: The MC for patch ϕ_1 of Figure 8.16 (a).

Step 2 is illustrated in Figure 8.18 for two adjacent patches; the corresponding local MCs are shown in Figures 8.19 and 8.20. We start with a polygonal approximation of Ω consisting of one straight-line segment for each trimming curve L, namely the segment contained in the root the binary tree representing L.

At each iteration, we refine the current constrained triangulation in one of the two following ways: either we insert a point which lies in the interior of the current domain discretization, or

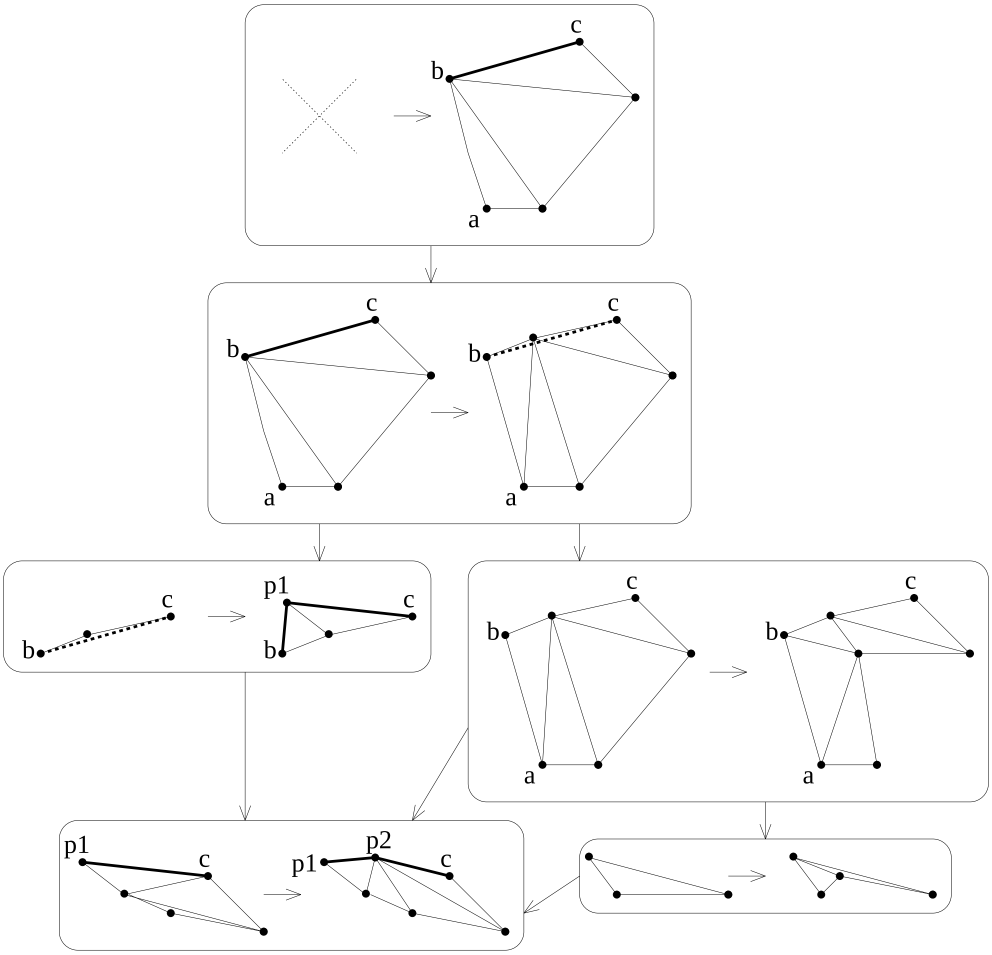


Figure 8.20: The MC for patch ϕ_2 of Figure 8.16 (a).

we refine the current domain boundary by performing one of the modification patterns contained in a binary tree. The approach is similar to that described in Section 8.2.1.2.

At a generic stage, all the points lying inside the carrier of the current triangulation are potential candidate to be inserted. Among them, the preferred one is the point p such that the distance of $\phi(p)$ from the triangle $\phi(p_1)\phi(p_2)\phi(p_3)$ is maximum, where triangle $p_1p_2p_3$ is the one containing p. The potential candidates on the trimming curves are the point insertions (already computed and stored as nodes in the binary trees) which have not yet been applied, and are children of modifications already applied. Among them, the preferred one is the one having the maximum stored error. Thus, we have a candidate interior point, and a candidate boundary modification: among the two candidates, the one corresponding to the largest error is chosen to refine the current mesh.

We generate a history that will be used for building the local MC for the current patch, where

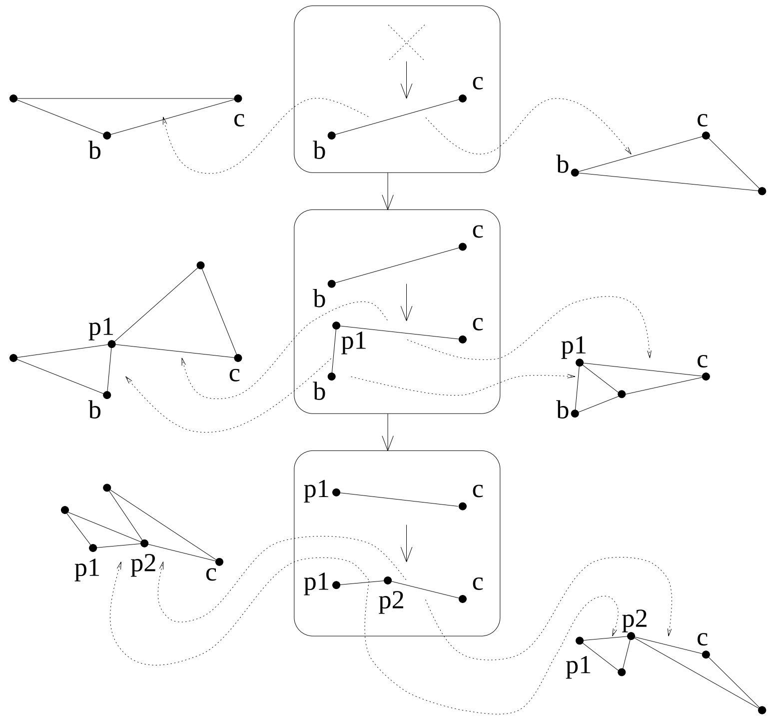


Figure 8.21: The one-dimensional MC of the common trimming curve of patches ϕ_1 and ϕ_2 enriched with information about incident triangles of the two individual MCs.

each insertion of a new point either inside or on the boundary of the patch domain yields a modification pattern.

In addition, we generate information that will be used to glue the MCs of adjacent patches in step 3. Let us consider a one-dimensional modification pattern C in the binary tree for a trimming curve L, which represents the refinement of a segment p_1p_2 with two segments p_1p and p_2p. In the binary tree of L, we associate with each of p_1p and p_2p a pointer to the triangle created in the boundary triangulation that is incident in p_1p and p_2p, respectively (see Figure 8.21). Note that, if L is shared by two neighboring patches ϕ_1 and ϕ_2, then a node of the binary tree receives a pointer to a triangle of the local MC of ϕ_1 and a pointer to a triangle of the local MC of ϕ_2 (see Figure 8.21).

Step 3 glues adjacent nodes of the local MCs, built at Step 2, in order to obtain the final MC of the whole surface. MC-nodes to be glued are detected by using the binary trees of the trimming curves which, in Step 2, have been enriched by references to triangles of the local MCs. In any tree node we find references to two adjacent triangles contained in $\Pi_2(C_1)$ and in $\Pi_2(C_2)$, respectively, for two nodes C_1 and C_2, possibly belonging to local MCs of different patches; we glue C_1 and C_2 into one node of the final MC. The MC-node resulting from glueing inherits incoming and outgoing arcs from both C_1 and C_2. Duplicated arcs with the same source and

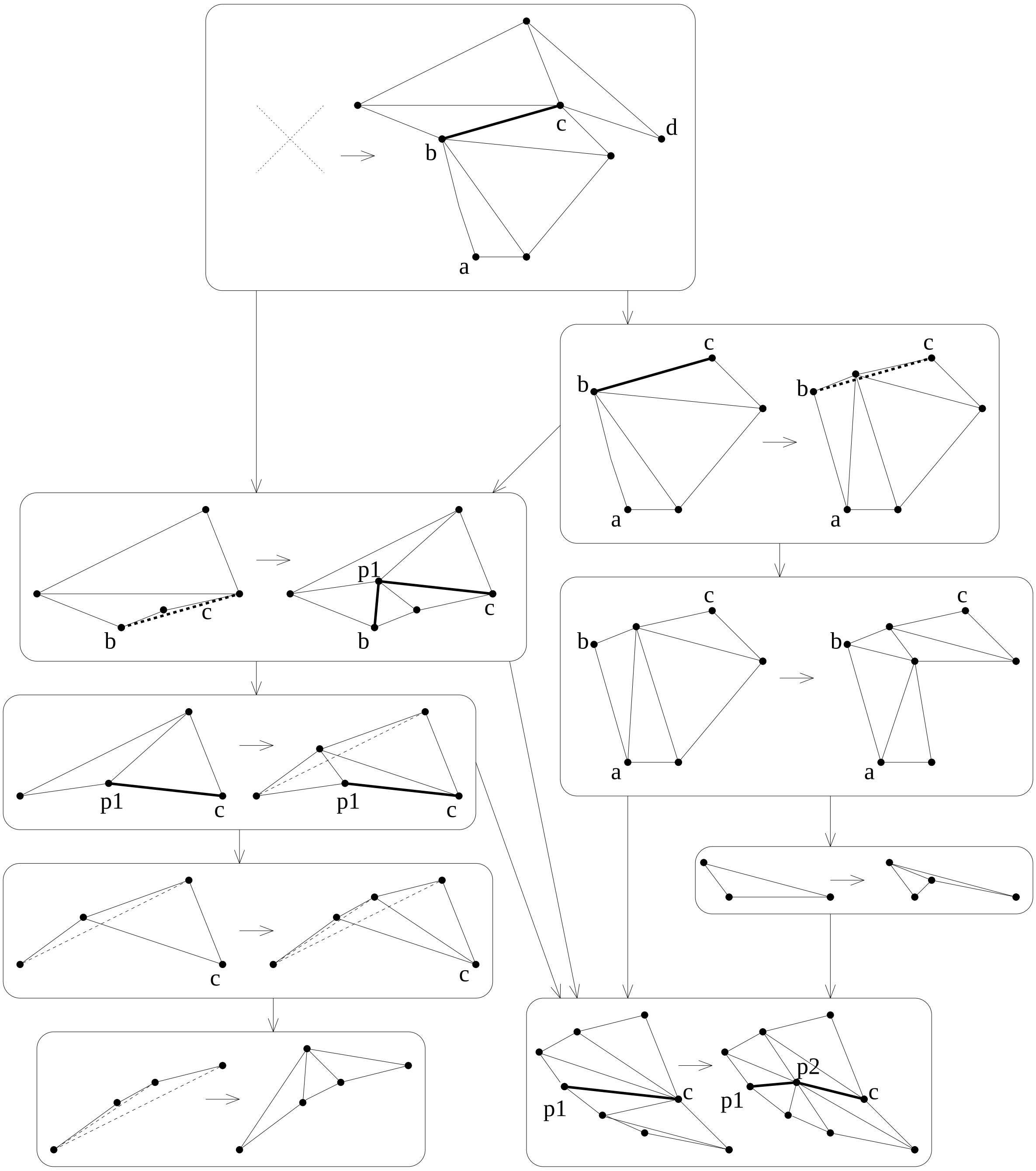


Figure 8.22: The final MC for the two neighboring parametric patches ϕ_1 and ϕ_2 of Figure 8.16 (a), obtained by glueing the two individual MCs of Figures 8.19 and 8.20.

destination, which may be created in such process, are collapsed into a single arc. Figure 8.22 shows the final MC for two adjacent patches.

The algorithm described in this subsection has not been implemented yet. We have developed just the preprocessing algorithm needed to ensure matching between the trimming curves of neighboring patches (see [Del97]). This task turned out to be more difficult than expected, due to inconsistencies and errors present in typical CAD files.

8.4 Reconstruction of Solid Objects

In this section, we use the Multi-Complex to represent the volume of a solid object $\mathcal{O}$ in $\mathbb{E}^3$, reconstructed from a set S of scattered points on its surface. A preliminary version of this work appeared in [DFMP98d].

Our aim is modeling the *space occupied* by $\mathcal{O}$ at multiple resolutions. The coarsest representation of $\mathcal{O}$ is the convex hull of the set of data points S, and each finer approximation is contained into coarser ones. Less detailed representations hide fine concave details of the object shape and provide sketches of its volume which can be useful in rendering (e.g., for testing whether the object intersects the view frustum), shape recognition and classification (a lower resolution permits to focus on the basic features of the shape), and collision detection (e.g., in virtual reality and planning of manipulation tasks).

Our MC is built from the history generated by a *sculpturing* process that starts from a tetrahedralization filling the convex hull of data set S and iteratively removes tetrahedra until all points of S appear on the surface of the resulting shape. This type of MCs, that we call *Multi-Shape*, provides a multiresolution representation of the interior volume of the object (through a tetrahedral mesh) as well as a multiresolution representation of its surface (through the boundary triangle mesh of the tetrahedralization). We have designed a Multi-Shape having in mind its application in the field of virtual reality.

8.4.1 Multi-Shapes

Multi-Shapes are MCs which encode three-dimensional modification patterns for a tetrahedral mesh and, at the same time, encode the corresponding two-dimensional modification patterns for its boundary triangulation. There is a one-to-one correspondence between three- and two-dimensional modification patterns.

To avoid confusion, in the following we use the symbols Γ and γ for tetrahedral meshes and tetrahedra, and Σ and σ for triangle meshes and triangles, respectively.

Three-dimensional modification patterns describe the removal of a subset Γ_i of tetrahedra from a current mesh Γ. Two-dimensional modification patterns describe the replacement of a subset

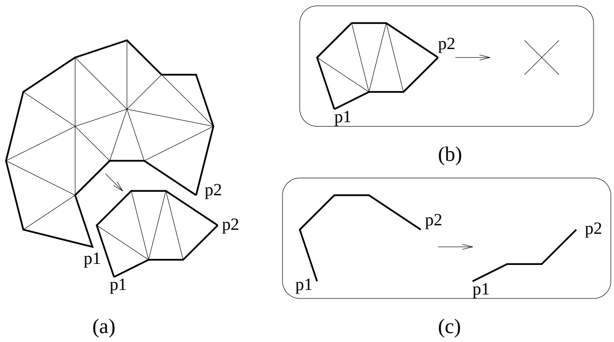


Figure 8.23: (a) Removing a subset of tetrahedra from a mesh; (b) the corresponding three-dimensional modification pattern; (c) the corresponding two-dimensional modification pattern.

of boundary triangles (namely, the triangular faces of tetrahedra in Γ_i which belong to $\partial(\Gamma)$) with another set of triangles (namely, the triangular faces of tetrahedra in Γ_i such that the other incident tetrahedron is in $\Gamma \setminus \Gamma_i$, see Figure 8.23).

Three-dimensional modification patterns have the form $(\Gamma_i, \emptyset)$, where each Γ_i is contained into the initial tetrahedral mesh Γ_0. Thus, all three-dimensional modification patterns in a Multi-Shape depend on the one in the root (corresponding to the creation of Γ_0). The relation of direct dependency between three-dimensional modification patterns does not correspond to the intuition that a set of tetrahedra can be removed only if has some triangular face on the boundary of the shape. Therefore, the partial order encoded in the Multi-Shape is the one determined by two-dimensional modification patterns, which corresponds to the intuition.

The nodes of a Multi-Shape contain two- and three-dimensional information, as follows:

- the root C_0 represents a three-dimensional modification pattern of the form $(\emptyset, \Gamma_0)$, where Γ_0 is a non-empty tetrahedralization; C_0 also represents a two-dimensional modification pattern $(\emptyset, \Sigma_0)$, where Σ_0 is the boundary triangulation of Γ_0;
- every node C_i, different from the root, represents a three-dimensional modification pattern of the form $(\Gamma_i, \emptyset)$, where $\Gamma_i \subset \Gamma_0$; C also represents a two-dimensional modification pattern (Σ_i, Σ_i'), where $\Sigma_i \cup \Sigma_i'$ is the same as boundary triangulation of Γ_i.

The arcs of a Multi-Shape encode links of direct dependency between the two-dimensional modification patterns associated with its nodes: there is an arc (C_i, C_j) if and only if $\Sigma_i' \cap \Sigma_j$ contains at least one triangle.

A Multi-Shape can be encoded with a data structure for two-dimensional simplicial MC in $\mathbb{E}^3$ enriched with volumetric information as follows: the root C_0 stores the tetrahedra of Γ_0 and every other node C_i stores references to the tetrahedra of Γ_i.

Any sub-MC $\mathcal{M}'$ of $\mathcal{M}$ provides a tetrahedral mesh approximating the volume of the object $\mathcal{O}$

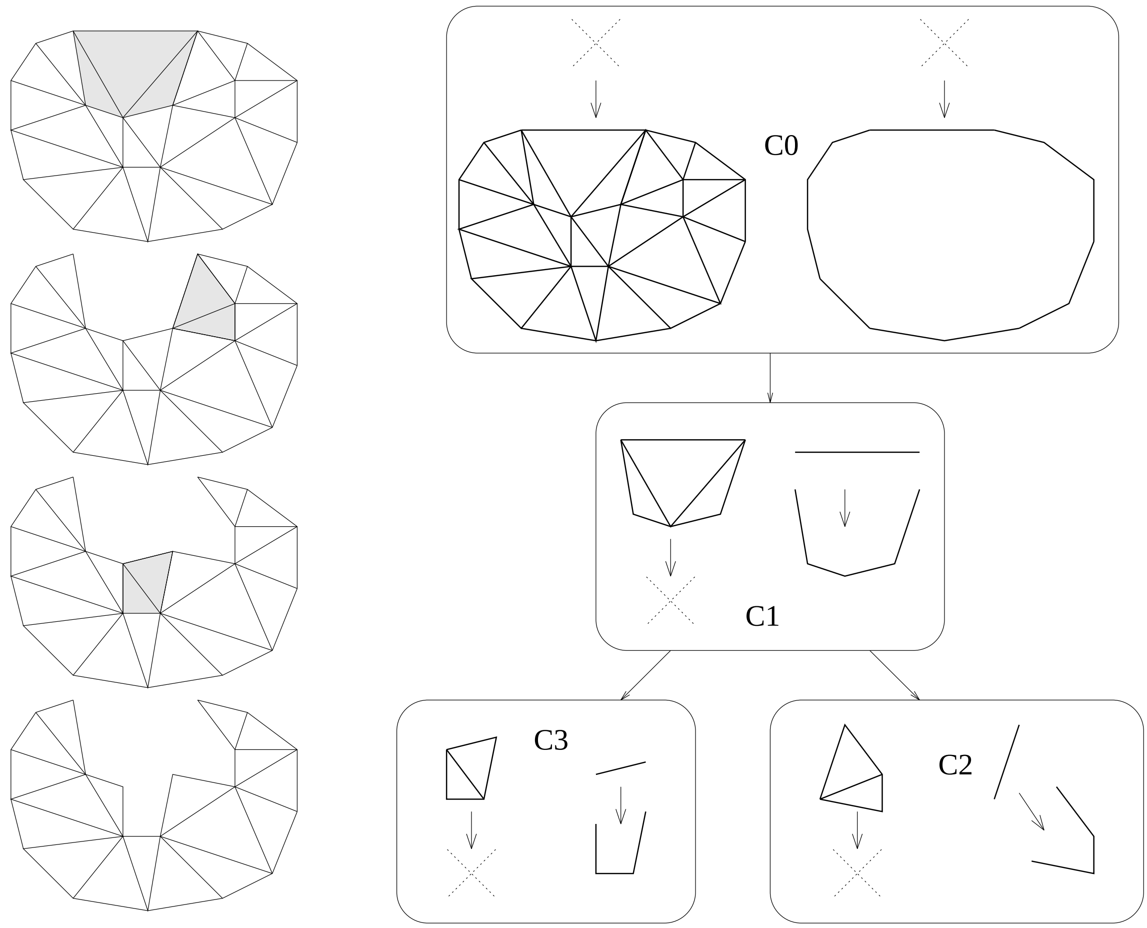


Figure 8.24: (a) The phases of a process of incremental sculpturing, for simplicity seen in two dimensions; the triangles removed at each step are shown shaded. (b) The corresponding MC, showing both the two-dimensional and the one-dimensional modification patterns associated with each node.

(obtained by applying all the three-dimensional modification patterns of $\mathcal{M}'$ to the initial tetrahedralization Γ_0), and also a triangulation approximating the bounding surface of $\mathcal{O}$ (obtained by applying all the two-dimensional modification patterns of $\mathcal{M}'$ to the initial triangulation Σ_0).

8.4.2 Construction of a Multi-Shape

We assume that the object $\mathcal{O}$ to be modeled is known only through a set S of unorganized points, known to be part of its boundary (e.g., points sampled with a sensing device).

In order to build a Multi-Shape for $\mathcal{O}$, we start from a tetrahedralization of the convex hull of S

and iteratively remove subsets of tetrahedra. Each step of such process modifies both the volume and the surface of the current shape. During the process, we generate a two-dimensional history with volumetric information added: at each step, we record the two-dimensional modification pattern $(\Sigma_{old}, \Sigma_{new})$ corresponding to the change in the surface triangulation as well as the set Γ of removed tetrahedra. It is easy to see that such type of history is always conflict-free.

In the literature, we do not find algorithms that are good candidates to be used as history generators. The algorithm by Boissonnat [Boi84] (see Section 3.2.4) is iterative but, because it removes just one tetrahedron at a time, it can only reconstruct solids of genus zero. The method proposed by Bajaj et al. [BBX95, BBCS96, BBCS97] (see Section 3.2.4) can deal with solids of arbitrary genus, but it is not iterative.

Here, we propose a new sculpturing method that is iterative (thus, it can be used to generate a history), and can reconstruct objects of arbitrary genus [DFMP98d]. Beside its use to build a Multi-Shape, the sculpturing algorithm proposed here improves over previous techniques because it has no limitation on the genus of the reconstructed object and can adapt the resolution of sculpturing operations depending on the density of data points in the different parts of the object.

Our starting point is a Delaunay tetrahedralization of the convex hull of S. If not all the points of S are on the convex hull, then we start a sculpturing process that iteratively removes a *connected set of tetrahedra* in order to obtain a new tetrahedral mesh with more points of S on its external surface.

A tetrahedral mesh Γ provides a *valid* shape representation if it satisfies the following three conditions [BBCS96]:

1. Γ is connected;
2. Γ contains all data points inside or on its boundary;
3. the domain of the boundary triangulation of Γ is a two-manifold.

Assuming that the current tetrahedral mesh Γ is a valid shape representation, each step of the sculpturing process removes a connected set Γ' of tetrahedra from Γ; set Γ' must have at least one triangle on the external boundary of Γ, and $\Gamma \setminus \Gamma'$ must be connected. Removing Γ' may effect the current shape in three ways (see Figure 8.25):

- *genus-preserving modifications* make a cavity,
- *genus-increasing modifications* create through-holes or handles, and
- *genus-decreasing modifications* break handles.

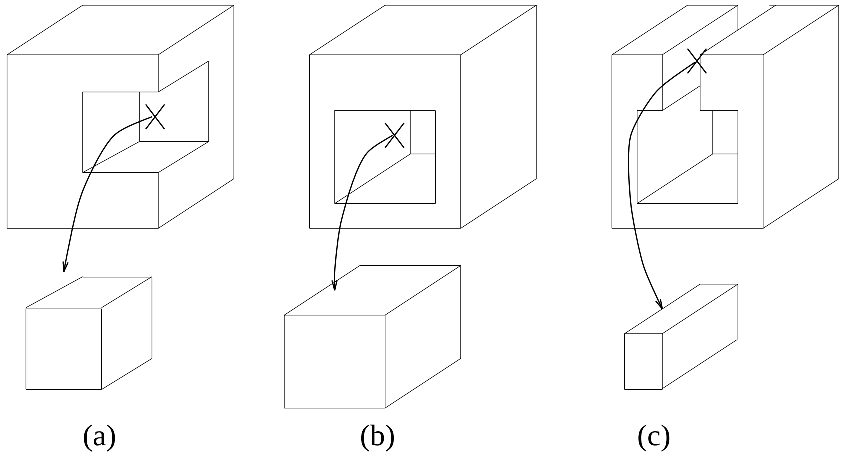


Figure 8.25: Types of mesh modifications: (a) genus-preserving, (b) genus-increasing, (c) genus-decreasing.

Our algorithm selects tetrahedra to be eliminated in such a way to generate a nested sequence of α-solids. The α-solid [BBX95, BBCS96, BBCS97] has been informally introduced in Section 3.2.4; here, we give here a more precise definition:

Definition 8.1 *Given $\alpha \in \mathbb{R}^+$, and a set S of points in $\mathbb{E}^3$, the* α-solid *is the tetrahedral mesh obtained from the Delaunay tetrahedralization of S by removing every tetrahedron γ for which there exists a path $\gamma_0, \ldots \gamma_h$ of adjacent tetrahedra such that:*

- $\gamma_0 \equiv \gamma$,
- *the circle circumscribed to the triangular face shared by any pair of tetrahedra γ_i, γ_{i+1} has a radius $\geq \alpha$,*
- *γ_h has a triangle on the boundary of the Delaunay tetrahedralization, and the circle circumscribed to such triangle has a radius $\geq \alpha$.*

An important property of α-solids is that they are nested: a solid for a smaller α is contained into a solid for a bigger α, and more data points lie on its boundary. Changing the value of α may change the genus of the corresponding α-solid.

The algorithm starts with the Delaunay tetrahedralization of S and performs a sculpturing process based on a decreasing sequence of values of α, equal to the sequence $l_1 \ldots l_k$ of radius lengths of the circles circumscribed to all triangles of the Delaunay tetrahedralization of S, sorted in decreasing order. It iterates a loop with a value of α decreasing from l_1 to l_k. At the i-th step, all tetrahedra belonging to the α-solid for $\alpha = l_{i-1}$ and not for $\alpha = l_i$ are removed.

Tetrahedra having at least one triangle on the boundary of the mesh are maintained in a priority queue according to their corresponding α-values. At the i-th step, the algorithm extracts

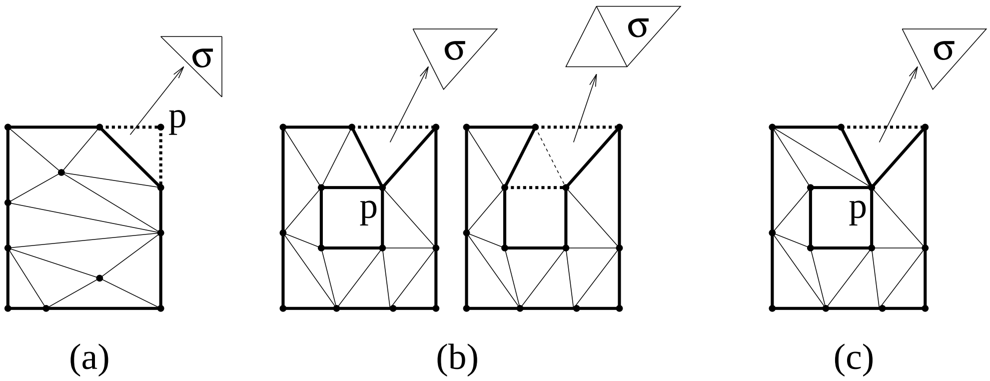


Figure 8.26: Cases of violation of validity in two-dimensions: (a) unrecoverable, removing σ leaves p isolated; (b) recoverable, removing σ with one of its adjacent triangles restores the manifold condition; (c) unrecoverable, removing any set of adjacent triangles to σ either isolates a point or does not resolve the non-manifold condition.

tetrahedra from the queue and removes them from the mesh as long as the first tetrahedron in the queue has an α-value $\geq l_i$.

For a certain value of α, the α-solid is not necessarily valid. Thus, before removing a tetrahedron, we carefully check whether validity is preserved. Given a tetrahedron σ with an α-value larger or equal to l_i, the following situations may occur:

- removing σ leaves the mesh valid: remove σ, and add the corresponding modification pattern to the history;
- removing σ either disconnects the mesh or leaves some data point outside the mesh [1]: this is an unrecoverable violation of mesh validity; do not remove σ, and delete it from the priority queue (see Figure 8.26 (a));
- removing σ only introduces a non-manifold condition: in this case, the violation of mesh validity may be recoverable; try performing further sculpuring around the non-manifold edge or vertex, in the attempt to recover validity. One of three situations may occur:
  - success: perform a valid genus-varying update, which removes σ *together with* some other tetrahedra (see Figure 8.26 (b)); add the corresponding modification pattern to the history;
  - failure due to an unrecoverable violation of validity: do not remove σ, and eliminate it from the queue (see Figure 8.26 (c));
  - failure because the current value of α is too big to proceed further: do not remove σ now, leave it in the queue and change its α-value to the larger value among those that blocked the procedure (in this way, σ may still be deleted in a later step).

[1] Note that the removal of a single tetrahedron cannot disconnect the mesh.

In order to find the set of tetrahedra that must be removed to recover validity (if possible), we perform a depth-first search of the adjacency graph of the current tetrahedralization. We start from a tetrahedron incident at the non-manifold entity, and traverse the common triangle of two tetrahedra only if the radius of its circum-circle is larger than α. We limit the depth of the search to ten tetrahedra, since we have found experimentally that in practice either validity can be recovered by removing just a few tetrahedra, or it cannot be recovered at all. During the search, other elements of the mesh may become temporarily non-manifold: we tolerate at most ten non-manifold elements simultaneously, otherwise we report failure on that branch of the search.

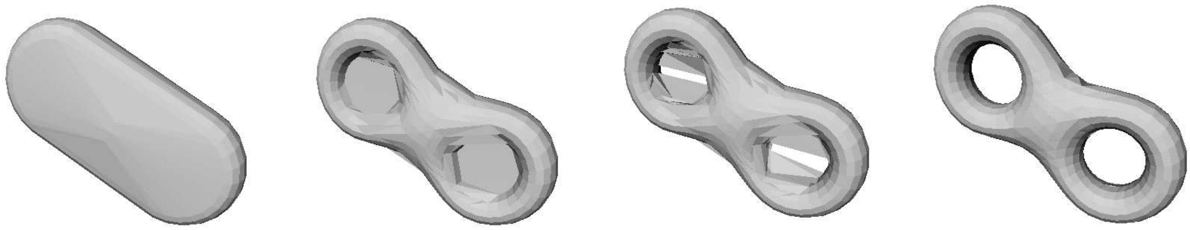
Note that we always try removing one tetrahedron at a time, unless validity issues require us to remove more tetrahedra together. This ensures the minimality of the modification patterns produced.

We have implemented the sculpturing algorithm and we are currently working at the implementation of the Multi-Shape. Figure 8.27 shows some stages of the algorithm.

Now, we evaluate the time complexity of the sculpturing algorithm. Let us consider a generic iteration, and let γ be the tetrahedron candidate for removal. Following [Boi84], the effect of removing γ can be determined in constant time: the mesh remains valid if and only if either γ has one triangle on the current boundary, and the opposite vertex inside, or γ has two triangles on the boundary, and the opposite edge inside; we have an unrecoverable violation of validity if and only if γ has three triangles on the current boundary (i.e., removing γ leaves an isolated point); in all other cases, we have a recoverable violation. It is easy to see that, in case validity is preserved, or violated in an unrecoverable way, processing γ takes $O(1)$ time. In this case, t appears in the queue only once. The worst case occurs if γ causes a recoverable violation of validity. In this case, the elements (vertices or edges) of γ that become non-manifold are determined in $O(1)$ time. The procedure in charge of trying to restore the manifold condition explores at most a bounded number of tetrahedra (equivalent to traversing a ternary tree of depth ten). At each visited tetrahedron γ', it is necessary to test whether its elimination disconnects the mesh, and to determine which elements (vertices or edges) of γ' change their state from non-manifold to manifold, or conversely. The most expensive test is the connectivity test because, in the worst case, we may have to traverse the whole mesh through adjacencies[2]. Thus, the total cost of processing γ, in case of a recoverable violation, is $O(t)$ where t is the number of tetrahedra in the mesh.

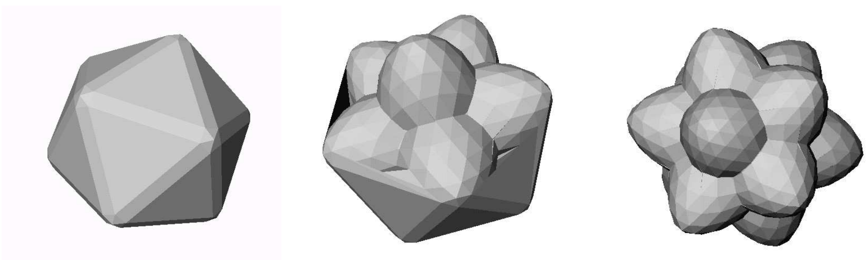
Note that a tetrahedron γ that causes a recoverable violation of validity is inserted into (and, thus, extracted from) the priority queue more than once if the procedure for recovering validity gets blocked due to a too large value of α. In the worst case, γ could appear in the queue $O(t)$ times, but this is really unlikely. We denote with K the number of recoverable violations of validity occurred during the algorithm.

[2] We *can* disconnect the mesh here because we are removing more than one tetrahedron.



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

Figure 8.27: Four stages of sculpturing an object: (a) the convex hull of the data set; (b,c) intermediate stage of the process; (d) final result.



(a) (b) (c)

Figure 8.28: (a) The convex hull of the data set; (b) the result of sculpturing just half of the object; (c) the result of sculpturing the whole object.

The total time complexity of the algorithm is $O((K+t)\log t + Kt + t$, where the first term is due to the management of the priority queue, the second term is due to the management of recoverable violations, and the final term is due to the processing of tetrahedra which do not cause recoverable violations; $K = O(n^2)$ in the worst case.

It is straight-forward to modify the algorithm in order to sculpture an object with different values of α in different parts, thus adapting to the resolution of shape features, or to the density of sampling. In practice, the algorithm can stop sculpturing some part of the object for a certain value of α, while continuing on other parts with a smaller α. For instance, in Figure 8.28, we have sculptured just half on an object, leaving $\alpha = +\infty$ on the other half.

8.4.3 Operations on Multi-Shapes

In this section, we consider resolution filters suitable to extract shape representations from a Multi-Shape. Since Multi-Shapes are both two-and three-dimensional models, they can be queried with resolution filters that refer either to triangles or to tetrahedra, and the intersection with a focus set can be tested either on triangles, or on tetrahedra.

Resolution filters acting on triangles are not very interesting for the applications. We simply note that a resolution filter depending on the circum-radius of triangles (as the one mentioned in Section 8.3) gives the same result as sculpturing up to a certain value of α.

In the case of Multi-Shapes, filters can be defined which take into account the importance of shape details that remain hidden (i.e., not sculptured) if a certain modification pattern is not applied.

The basic idea is that each node of a Multi-Shape discovers a (part of a) concave feature (a cavity or a through-hole) of the object. Each node can be assigned an importance based on the relevance of the corresponding cavity / hole for the object shape. For instance, the following parameters may be taken into account for defining the importance of a node C_i:

- the type of modification pattern (genus-preserving, -increasing, -decreasing);
- the volume subtracted in the modification (i.e., the volume of the set of removed tetrahedra);
- the increment/decrement in the area of the object surface caused by the modification (i.e., the difference in area between triangle meshes Σ_i and Σ'_i, where (Σ_i, Σ'_i) is the two-dimensional modification pattern associated with C_i);

Moreover, importance values can be computed incrementally starting from the leaves of the Multi-Shape and setting the "accumulated" importance of a node C_i as the sum of the importance of C_i and of the importances of its descendants. In this way, a "small" modification pattern may become important since it blocks "bigger" modifications which come after it in the partial order.

We can extract from a Multi-Shape representations corresponding to having performed (i.e., having included in the sub-MC which gives the solution) all modification patterns above a certain importance chosen by the user, which may be different in different parts of the object. Here, an importance threshold function is used to induce resolution filters; query algorithms do not evaluate the resolution filter on the cells of the current (triangle or tetrahedral) mesh, but on the nodes which are the destinations of the arcs of the current MC-front.

We have developed an operation on Multi-Shapes that, given a viewpoint v, extracts an object description where only the concave features (over a user-defined importance) which are visible from v have been sculptured.

The basic idea is that a concave feature, represented in a node C_i, can be visible only if the triangles of the "entrance" of the cavity or hole (i.e., the triangles of Σ_i, where (Σ_i, Σ'_i) is the two-dimensional modification pattern in C_i) face towards v [3]. A similar test replaces the test for intersection with the focus set in a local extraction algorithm. A working example of the

[3] Triangles in a Multi-Shape must be oriented in such a way that their normals point outside the object.

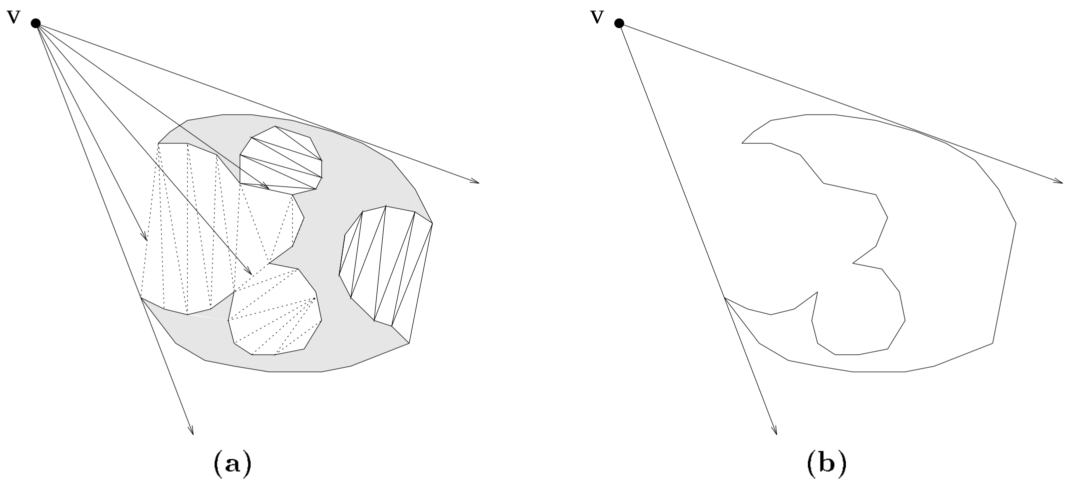


Figure 8.29: (a) An object represented through Multi-Shape and a viewpoint v; tetrahedra filling the cavities are shown; the "true" object shape is colored in grey. Only the cavities whose entrance faces toward the viewpoint are sculptured, thus removing just dashed tetrahedra. (b) The resulting object representation, which cannot be distinguished from the true shape by an observer located at v.

method in two-dimensions is shown in Figure 8.29. This approach can be coupled with an importance threshold function that depends on the distance of a node from the focal axis, thus sculpturing with a higher accuracy near the center of the image.

An important problem in virtual reality is that of testing whether two objects $\mathcal{O}_1$ and $\mathcal{O}_2$ collide, i.e., intersect. We think that representing objects through Multi-Shape can help collision detection. The idea is to visit the Multi-Shapes of $\mathcal{O}_1$ and $\mathcal{O}_2$ simultaneously, starting from the root. Initially, the convex hulls of the two objects are considered; next each object is iteratively sculptured by visiting nodes that refine the representation where an intersection is found. This process permits to check whether the intersection really exists, or it is a false intersection because it occurs with the interior of a cavity or of a hole. An algorithm using such an idea is currently under investigation.

8.5 Volumetric Data Visualization

Volumetric data are three-dimensional scalar fields. By extending the approach used in Section 8.2 for two-dimensional scalar fields (terrains), we represent a three-dimensional scalar field $\phi : \Omega \subseteq \mathbb{E}^3 \longrightarrow \mathbb{E}$ through a three-dimensional simplicial Multi-Complex $\mathcal{M}$ in $\mathbb{E}^3$, where each vertex p has associated the corresponding field value $\phi(p)$. As in the case of terrains, the field values define a transformation of $\mathcal{M}$ into a three-dimensional MC in $\mathbb{E}^4$, that we denote with $\hat{\mathcal{M}}$. Similarly, a tetrahedron $\sigma \in \mathcal{M}$ becomes a tetrahedron $\hat{\sigma} \in \hat{\mathcal{M}}$. Approximation errors are defined based on the field metrics (see Section 2.4.3).

History generators for volumetric MCs are provided by iterative algorithms that progressively refine or coarsen a tetrahedralization (see Section 3.2.2). In collaboration with the University of Pisa, we have developed a history generator for Delaunay tetrahedralizations which uses a refinement strategy based on vertex insertion.

The approach used in two dimensions to insert point in a Delaunay triangulation extends to Delaunay tetrahedralizations. A new point p is inserted by locating the tetrahedron σ containing p, splitting σ at p, and then optimizing the mesh through flips. In three dimensions, a flip involves a set of five points, and switches between a configuration with two tetrahedra and one with three tetrahedra. The cost of inserting a point is $O(t)$, in the worst case, where t is the size (i.e., the number of tetrahedra) of the current tetrahedralization, but it is constant on the average [Dwy89]; we recall that, denoted with n the number of data points, $t = \Theta(n^2)$ in the worst case [For92].

The refinement history generator inserts one point at a time in a Delaunay tetrahedralization, chosen as the one which causes the maximum error in the current mesh; such method is the three-dimensional extension of Variant TD2 presented in Section 7.5 for two-dimensional fields.

For encoding an MC representing volumetric data, we adopt a three-dimensional version of the explicit data structure. Since tetrahedra in a three-dimensional MC may be a very large number (much more than triangles in a two-dimensional MC with the same number of vertices), compressed data structures are of great interest. We are planning to implement a three-dimensional version of the compressed Delaunay-based data structure described in Section 6.4.2.1.

We have considered operations on three-dimensional MCs for answering basic queries and supporting scientific visualization. Basic queries can be defined as local queries on a three-dimensional MC in the same way as done two-dimensional MCs (see Section 8.3):

- *point location* queries allow getting the field value at a given point in space; the focus set is a point in $\mathbb{E}^3$;
- *windowing* queries allow focusing the attention inside a box; the focus set is an axis-parallel cuboid in $\mathbb{E}^3$.

The above focus sets are defined in the space of the field domain; thus, assuming that the MC is conservative (i.e., the domain of the field is preserved across different resolutions), the corresponding queries can be answered referring to active tetrahedra.

Beside basic queries, we consider operations for supporting visulaization of three-dimensional data with the most common volume rendering techniques:

- *Depth sorting* of a tetrahedral mesh with respect to a viewpoint is an operation common to different visualization techniques. In order to support efficient depth sorting of a mesh returned from querying an MC, we generate adjacency information and flag each adjacency link (from a tetrahedron σ_1 to a tetrahedron σ_2) with a bit telling which of the two adjacent tetrahedron is nearer to the viewpoint (i.e., whether σ_1 is before σ_2 in the depth order, or conversely). Actual depth sorting is performed as a post-processing on the tetrahedral mesh returned from querying an MC (e.g., through a topological sorting of the adjacency graph in which each adjacency link is directed based on the value of the flag).

- *Visualization through isosurfaces* involves the same problems mentioned in Section 8.2.2.5 when dealing with contour lines of two-dimensional fields. We briefly recall the two possible approaches:

  1. given on-line a field value h, perform a local query using a focus set in $\mathbb{E}^4$ consisting of the three-dimensional hyperplane, locus of the points with their fourth coordinate equal to h;
  2. for a fixed set of field values, build off-line a two-dimensional MC representing the corresponding isosurfaces at multiple resolutions.

- *Visualization through cross sections* reduces to a local query in $\mathbb{E}^3$ with an arbitrary plane π as focus set (the queried MC is conservative). For each reported tetrahedron σ, we compute the intersection of σ with π. By interpolating the field values on the resulting polygons, we obtain a two-dimensional field which can be visualized through color coding.

- *Visualization through ray-tracing* needs to determine all the tetrahedra (at a user-defined resolution or accuracy), which intersect a given visual ray, also known in the literature as a *ray shooting* query. In this case, the focus set is a line in $\mathbb{E}^3$, which represents a visual ray emanating from the viewpoint (the queried MC is conservative). It is useful to generate adjacency relations, which directly provide a depth order of the tetrahedra.

The queries described here are currently under implementation.

8.6 Progressive Compression of Triangle Meshes

In this section, we apply the Multi-Complex for compressing triangle meshes. We propose a progressive method for compressing the connectivity of a triangle mesh which is a bitstream variant of the compressed structure for two-dimensional simplicial MCs described in Section 6.4.2.2. The method has been described in [DFMP98a] in the context of TIN compression, and in [DFMP98b]. Moreover, we outline the architecture of a client / server system based on the MC for sending meshes at variable resolution.

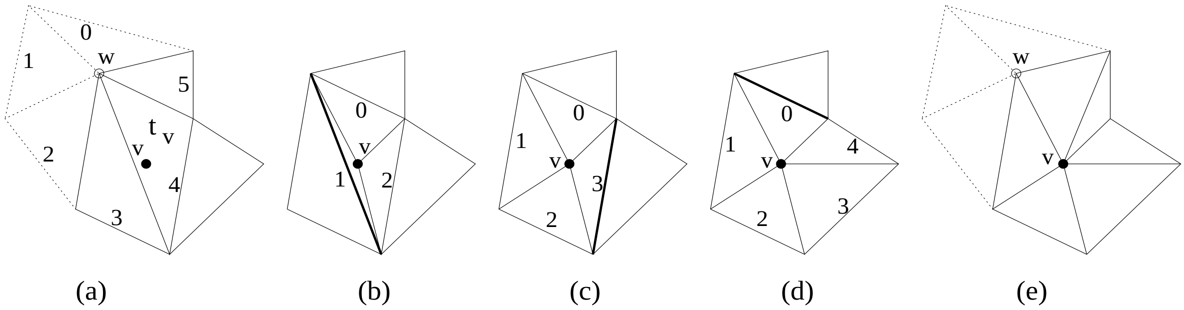


Figure 8.30: The triangulation (a) after and (e) before removing a vertex v, and (b,c,d) the related sequence of edge flips, to be read right-to-left, that restore situation (e) from situation (a). In (a), the edges around w, and, in (b,c,d), the edges around v are numbered. Triangle σ_v is encoded as the pair $(w, 4)$ and the sequence of edge flips is 1,3,0.

Progressive compression methods (see Section 3.4.2) encode a triangle mesh by iteratively coarsening it through the application of an invertible *destructive operator* until the resulting mesh has a sufficiently small size. Then, they transmit the final coarse mesh along with the encoded sequence of modifications performed, sorted in reversed order. Decoding recovers the initial mesh from the coarse one by sequentially applying the *constructive operators* corresponding to the inverse of operators used in coding.

The destructive operator used in our method removes a vertex v of degree bounded by a constant b from the current mesh and retriangulates the corresponding polygonal hole π. The inverse constructive operator inserts vertex v and recovers the previous triangulation of π.

According to the idea used in Section 6.4.2.2, in decoding we recover the triangulation of π containing v from the one without v by first splitting the triangle σ_v containing v into three triangles, and then applying a sequence of *edge flips*. Our bitstream encodes, for each removed vertex v, the starting triangle σ_v and the sequence of edge flips to be performed.

The information given for each vertex is very similar to that stored at each MC-node in the compressed data structure described in Section 6.4.2.2. Encoding the sequence of edge flips requires $\sum_{i=3}^{b-1}(\log i) = \log((b-1)!) - 1$ bits, as explained in Section 6.4.2.2. The starting triangle σ_v is encoded by exploiting the fact that, when inserting v, *all* vertices preceding v in the sequence have been inserted. Then, the current triangulation is exactly the same triangulation as the one from which v has been removed during coding. We encode σ_v by providing a reference to a vertex w of σ_v and the index of σ_v in the star of triangles incident at w. The reference to w requires $\log n$ bits (where n is the total number of vertices in the given triangulation); the index of σ_v requires $\log b$ bits.

Our bitstream contains each vertex once and, for each vertex, a vertex reference, a triangle index, and a packed sequence of edge flips for each vertex, as described above. Figure 8.30 illustrates the information sent with a vertex v and their use to re-insert v into the current triangulation. Connectivity information is represented in $(\log n + \log(b!) - 1)$ bits per vertex.

8.6.1 Coding and Decoding Algorithms

Here, we give a coding algorithm assuming that the input is a triangulation; in Section 8.6.2, we will explain how to generate the bitstream starting from a Multi-Complex.

The *coding algorithm* works as follows:

1. Initialize the current triangulation as the input one.
2. While the current triangulation is not "small enough", select a vertex v of degree at most b:
   - remove v and retriangulate the resulting polygonal hole π;
   - locate the triangle σ_v of the new triangulation of π, which contains v;
   - choose a vertex w of σ_v, having degree at most b, and compute the index of σ_v around w;
   - simulate the reinsertion of v and the reconstruction of the previous triangulation of π; record the sequence of edge flips.
3. Send the current triangulation, in a direct compressed format, followed by the sequence of removed vertices in reverse order; for each vertex v, output also the index of the associated vertex w, the index of σ_v in the set of triangles incident in w, and the packed sequence of edge flips.

The coding algorithm has two key points: the choice of the vertex v to be removed at each iteration of loop 2, and the criterion for retriangulating the corresponding hole π. *Any criterion* can be used for both tasks without affecting the format of the bitstream or the decoding algorithm. Since hole π has a bounded number of edges, retriangulating it always requires a constant number of operations.

The vertex selection criterion affects the complexity of the coding algorithm as well as the quality of the intermediate triangulations obtained from an interrupted bitstream (i.e., the ratio between accuracy and number of vertices used), and thus the efficiency of the sequence for *progressive transmission*. A number of techniques for vertex selection have been discussed and experimented in Section 7.5, in the context of Delaunay triangulations. In particular, Variants BU1-2-3 can be applied; Variant BU4 is not feasible because the degree of removed vertices is not bounded. Note that Variants BU1 and BU2, at each iteration, do not select one vertex, but a set of vertices; the vertices of such set can be removed (and, thus, encoded) one at a time in any order, since they are independent.

The *decoding algorithm* performs the following steps:

1. Initialize the current triangulation with the coarse triangulation read from the bitstream.
2. Read the next vertex v, with associated information (index of a vertex w already loaded, index of a triangle σ_v in the star of w, and sequence of edge flips). Insert v into the current triangulation by splitting triangle σ_v at v and performing the given sequence of edge flips.
3. The process ends when the bitstream is finished (in this case the original triangulation is completely reconstructed and we have a lossless decompression), or when the current triangulation has a certain number vertices, or when a certain time has past (in these cases, a triangulation approximating the original one is built and we have a lossy decompression).

The decoding algorithm works in linear time, since n vertices are read, and the insertion of any vertex v into the current triangulation involves a search for σ_v in the star of a vertex w of degree bounded by b, and a number of flips bounded by $b-1$, with b a predefined constant. Note that no numerical calculations are needed in reconstructing the triangulation.

8.6.2 A Client-Server Architecture with the MC

In the client/server architecture outlined in Section 2.6, a Multi-Complex can be used to implement the server in charge of triangle mesh management. When a client asks for a mesh satisfying certain requirements, the server extracts a mesh from the MC and sends it to the client in encoded format.

For such purpose, we use a Multi-Complex $\mathcal{M}$ with bounded width (bounded by the constant b), where nodes represent vertex insertions. For instance, $\mathcal{M}$ could have been produced through one of algorithms BU1-2-3 presented in Section 7.5. $\mathcal{M}$ may be stored with the compressed data structure based on edge flips (described in Section 6.4.2.2), but this is not mandatory since flips can be computed on-line.

The encoding algorithm does not have to coarsen a mesh and then to reverse the sequence of modifications. Instead, we traverse the MC top-down by applying either the basic or the local query algorithm (depending on the request of the user) with the given input parameters. During DAG traversal, we assign a progressive number to each vertex. Initially, we send the triangulation at the root of the MC, and assigns a progressive number to all its vertices. For each MC-node C included in the current (pseudo-)sub-MC, we assign the next progressive number to the new vertex v of $\Pi_2(C)$ and compute the coding parameters for v in the following way:

- if the MC is stored with a general-purpose data structure: we search in $\Pi_1(C)$ the triangle σ_v containing v; we choose a vertex w of σ_v, of degree $\leq b$, and compute the index of σ_v in the star of w in the current mesh; we compute the sequence of edge flips as in the ordinary coding algorithm;

- if the MC is stored with the compressed data structure based on flips: w is the vertex contained in the first parent C' of C; the index of σ_v in the star of w must be recomputed on the current mesh since the one stored in C refers to $\Pi_1(C')$; the flip code is already stored in C.

We send vertex v, the number previously assigned to vertex w, the index of σ_v and the flip sequence, packed as described in Section 8.6. The encoding algorithm working on the MC has a linear time complexity because the input MC has linear growth (for Theorem 4.6). The decoding algorithm is unchanged.

In the case of global queries only, it does not seem difficult to implement a client / server system based on the dynamic algorithm in which both the server and the client maintain a current triangle mesh and, on each new query, the server sends to the client a sequence of records encoding the modifications necessary to update the current mesh. Each record corresponds to the insertion of a new vertex (encoded as explained before) or to the removal of an existing vertex. In this latter case, the record contains the index of a vertex v of the current mesh and a flip code. In order to decode it, the client starts from the star of v and performs the reverse sequence of edge flips, ending in a situation where v has just three incident triangles; then, vertex v is simply removed by merging such three triangles into one.

8.6.3 Comparison with Other Progressive Methods

Other progressive compression methods have been proposed by Hoppe [Hop96] and by Snoeyink and van Kreveld [SvK97] (see Section 3.4).

Our method is less compact than [Hop96] and [SvK97]. Disregarding the cost of the initial triangulation, which is always very small, our method produces $n(\log n + \log(b!) - 1)$ bits of connectivity information [4]. Hoppe's method requires $n(\log n + \log(b(b-1)))$ bits (see Section 3.4.2). Table 8.1 compares the sizes of the our bitstream and Hoppe's bitstream for some values of b and n. Our bitstream is up to twice as long on average. Note that our algorithm is more sensitive to the value of the maximum degree b.

The bitstream of [SvK97] describes connectivity in just $O(\log^2 n)$ bits; no experimental results are known. However, their decoding algorithm involves heavy numerical calculations, which are time-consuming and subject to floating point inaccuracies; our method does not perform any numerical computation in decoding.

The main advantage of our method is in being more general and flexible since it does not impose any specific criterion for removing vertices and for retriangulating the surface after vertex removal. It can incorporate different criteria for choosing the vertices to remove, trading off between compression time and quality of the intermediate meshes. The only constraint is on the

[4] Indeed, we could send variable-length sequences of edge flips, preceded by a few bits indicating the length of the sequence; this might reduce the overall length of the file in practical cases.

n	b	Hoppe's	this
65536(64k)	4	20n	20n
	6	21n	25n
	8	22n	31n
	11	23n	41n
	16	24n	50n
131072(128k)	4	21n	21n
	6	22n	26n
	8	23n	32n
	11	24n	42n
	16	25n	51n
262144(256k)	4	22n	22n
	6	23n	27n
	8	24n	33n
	11	25n	43n
	16	26n	52n

Table 8.1: Connectivity bits in our compression method, and in Hoppe's method. We show values of b corresponding to $2^2, 2^3, 2^4$, 6 (the average vertex degree in a Delaunay triangulation) and 11 (the degree bound used for the vertex removal algorithms of Section 7.5).

maximum degree of removed vertices (similar to Hoppe's). Hoppe does not take into account the approximation error when selecting the next edge to collapse. Also, Snoeyink and Van Kreveld do not select vertices with an error-based criterion. Moreover, deleting independent sets of vertices at once puts a strong constraint, hence reducing adaptivity: the number of vertices necessary to achieve a given approximation error may be much higher, as shown by our experiments of Section 7.5.

In our approach, the specific criterion used for retriangulation is encoded in the bitstream itself rather than predefined. On the contrary, [SvK97] is restricted to Delaunay triangulations and [Hop96] to triangulations obtainable through edge flips (for instance, it cannot guarantee a Delaunay triangulation at each step).

8.7 A Software Library Based on the Multi-Complex

We have developed a prototype system for the representation and manipulation of geometric models at multiple resolutions based on the Multi-Complex. The main idea is to provide an *open system* which can be integrated with modules performing analysis and processing of geometric models with the additional characteristic of using resolution as a further parameter in the computation.

The system architecture consists of *library modules* and *user modules*. Library modules are the "standard" part of the system; they provide operations that allow a user to build and query an MC. User modules are the extensible part of the system; they include programs which generate histories for building MCs and programs that query an MC to perform specific operations at variable resolution.

The library modules are:

- The *MC-history-tracer*: this module provides basic primitives that can be called within a user program for tracing the history of an iterative process incrementally refining or coarsening a cell complex.
- The *MC-manager*: this is the only module having direct access to the MC data structure; it provides a standard interface between the MC and the applications, containing:
  - *query operations* described in Chapter 5 (basic and dynamic algorithms for the global query, algorithms for the local query);
  - *construction operations* described in Chapter 7 (used to build an MC from a given history);
  - *I/O operations* used to read/write an MC from/to disk.
- The *MC-builder*: this module builds an MC starting from a history read from a file (for doing that, it uses the construction operations and the I/O operations provided by the MC-manager).
- A library of predefined *resolution filters and focus sets*, to be used as input parameters for querying an MC through the operations of the MC-manager; focus sets can be used in local queries or to restrict resolution filters in a global query.

The user modules can be subdivided into the following types:

- *MC-history-generators*, which implement methods for the iterative refinement or coarsening of a cell complex and use the primitives the MC-history-tracer to write the history of the process [5].
- *MC-clients*, which use the query operation of the MC-manager to perform operations at variable resolution on geometric objects encoded through an MC (some examples of application-dependent operations implemented on top of the query primitives of the MC have been illustrated in this chapter).
- Additional *resolution filters and focus sets* implemented by the user to reflect specific application needs.

The system architecture is depicted in Figure 8.31 in terms of the relations among the different modules. Referring to Figure 8.31, we illustrate two typical scenarios: the construction of an MC and the extraction of a cell complex from an MC.

[5] Note that we have used the sequential approach to MC construction introduced in Section 7.3.2.

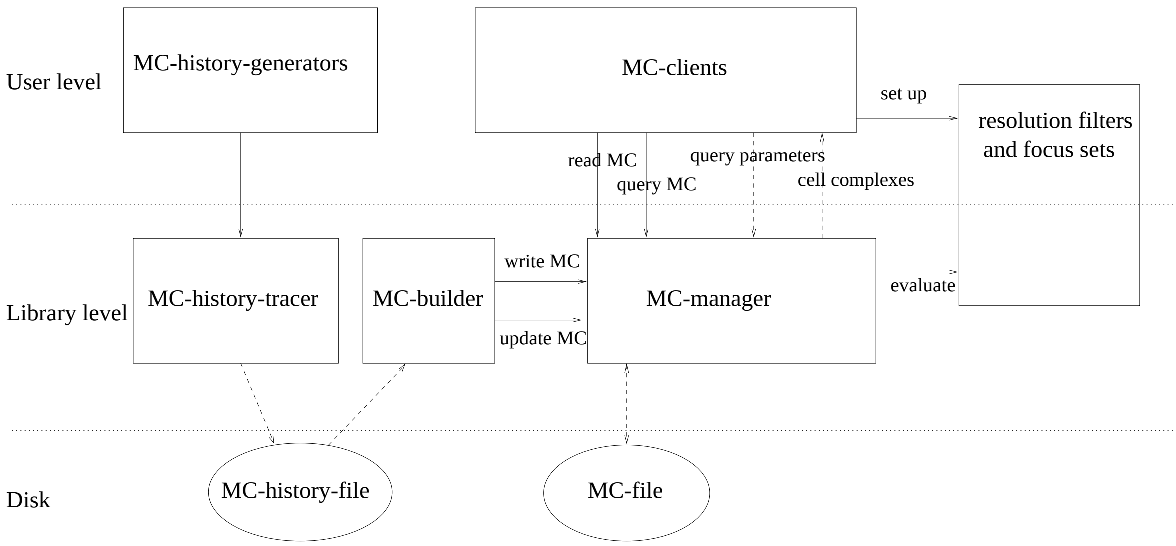


Figure 8.31: Components of the MC system and their relations. Dashed arrows represent data flow; solid arrows represent control flow (arrows directed from the module invoking an operation to the module providing such operation).

In order to construct an MC, first we run an MC-history-generator. Such a program performs a sequence of modifications either refining or coarsening a cell complex, and uses the primitives of the MC-history-tracer to write the corresponding history to disk into an *MC-history-file*. Then, the MC-builder reads the MC-history-file, builds an MC from the history, and writes it on disk into an *MC-file*. The MC-history-generators and the MC-builder run off-line.

An existing MC can be read from disk and used to perform operations on a cell complex representing the target object at an user-defined, possibly variable, resolution. An MC-user first asks the MC-manager to read an MC from an MC-file, then it sets up the desired query parameters and invokes a query. As a result, the MC-manager returns to the user a cell complex. The MC-library provides a number of predefined resolution filters and focus sets, but the system allows the programmer to implement its own ones and add them to the system.

At this point, we have developed a prototype library for two-dimensional simplicial MCs and the kernel of a library for three-dimensional simplicial MCs. Both libraries have been implemented in C++ under the Unix operating system.

The two-dimensional library implements the explicit data structure with zero-adjacency information (but it allows reconstruction the connecting structure of an extracted complex through the method described in Section 6.2.3) and the basic, dynamic, and local algorithms in their min-size version. The compressed data structure for Delaunay triangulations has also been implemented and we are integrating it into the system. We are currently implementing the compressed structure based on edge flips.

We have implemented MC-history-generators for two-dimensional fields based on a Delaunay triangulation of the domain (described in Section 7.5) and a history generator for free-form surfaces based on iterative vertex removal (see Section 8.3.1).

A number of MC-clients have been defined for querying and visualizing triangle meshes extracted from an MC. The experimental results shown in this thesis have been obtained from such implementations. Our MC-clients interact with the user through a multi-purpose graphic interface [Smo97]. Such interface is a separate program which runs in parallel with an MC-client and manages interaction with the user:

- it collects query parameters from the user and passes them to the MC-client;
- it receives from the MC-client the cell complex resulting from a query and displays it;
- it manages all tasks related to the interactive display of such a model (changes in the view parameters, lights, colors, etc.);
- it informs the MC-client when actions of the user cause a change in the current parameters (e.g., a change in the viewpoint position when using a resolution filter depending on distance from the observer).

Such multi-purpose interface is easily configurable according to the needs of each MC-client and has revealed to be an effective, easy-to-use and simple tool for obtaining satisfactory graphic interfaces with little effort. It has been developed under the Unix system and the X-Window environment based on the OpenInventor toolkit, and currently runs on Silicon Graphics workstations.

The three-dimensional library implements the explicit data structure with zero-adjacency information (and possibility of reconstructing the connecting structure of an extracted complex) and the basic, dynamic, and local algorithms in the version that minimizes the output size for a given resolution filter. We are integrating into the library the MC-history-generator for scalar fields mentioned in Section 8.5. MC-clients for volume visualization are under design.

8.8 Summary

In this chapter, we have shown applications of the Multi-Complex to represent and process geometric entities at variable resolution in different contexts.

We have provided methods to build an MC for different kinds of objects and from several data sources. The two-step approach to MC construction (i.e., history generation separated from assembly of the DAG) reveals to be very effective for isolating application-dependent aspects. Moreover, it is very simple to transform a generic program for the progressive refinement or coarsening of a cell complex into a history generator.

We have demonstrated through examples how application-dependent operations can be defined on top of basic queries on a Multi-Complex, thus proving the generality of MC operations defined in Chapter 5. The implementations presented here are not optimized, but are sufficient to show the applicability of the approach.

Nevertheless, there is much work still going on, and related problems which need to be investigated. For two-dimensional fields (terrains), we are implementing the construction technique that preserves lineal features described in Section 8.2.1.1, and the system of modular MCs described in Sections 6.5 and 8.2.1.2. The algorithm of Section 8.2.1.1 will be used to produce an MC that incorporates representations of contour lines at different resolutions.

For free-form surfaces in $\mathbb{E}^3$, there is the problem that existing algorithms do not provide conflict-free histories. This problem does not seem to have an easy solution. However, experiments show that it is not too relevant in practice. The use of mesh simplification algorithms as history generators is easy, and produces satisfactory results (see Section 8.3.1). Multiresolution discretization of parametric surfaces must face the problem of inconsistency in CAD files, which was the first and hard problem encountered in our (currently ongoing) implementation work (see Section 8.3.2).

We will investigate collision detection operations based on the Multi-Shape introduced in Section 8.4 and we will provide more operations for three-dimensional MCs representing scalar fields.

An open problem for MCs representing three-dimensional fields, due to their large size, is the design of efficient compressed structures to store them in main memory and of schemes for encoding and retrieving them on secondary storage. The extension to three dimensions of compressed two-dimensional data structures proposed in Section 6.4 is an open problem. We are currently studying the design of disk-based structures for two- and three-dimensional MCs.

Chapter 9

Concluding Remarks

In this thesis, we have proposed the Multi-Complex (MC), a general framework for multiresolution modeling of geometric objects based on cell decomposition. We have defined general application-independent queries on an MC as well as general operations to build an MC from application-dependent histories. We have proposed algorithms and data structures for an efficient implementation of queries on an MC. We have shown applications of the MC in various contexts by implementing application-dependent operations on top of the general operations of the MC. All proposed algorithms have been, are being, or will be integrated into a prototype software library for multiresolution geometric modeling based on the MC. Experimental results from implementations developed within such library have been shown.

In the following, we outline the main contributions, discuss future developments, and list some open problems.

9.1 Main Contributions

The contributions of this thesis are in four main directions:

1. theory of multiresolution geometric modeling;
2. design of data structures and algorithms for implementing the MC;
3. examples of the MC in case-study applications;
4. development of a software library for multiresolution geometric modeling.

9.1.1 Theoretical Contributions

The Multi-Complex data type provides the first unified approach to multiresolution modeling of (manifold) geometric objects through cell complexes. In particular:

- The MC has been defined in arbitrary dimensions and for complexes made up of cells with arbitrary shapes, and its properties have been developed in this general context.
- The query operations defined on the MC formalize the two basic problems mentioned by many authors, but never defined in a general setting: obtaining the smallest cell complex for some given accuracy requirements, and obtaining the cell complex that best approximates some accuracy requirements for a given size.
- The operations for building an MC are general. An MC can be built from any iterative process that is capable of producing either a refinement or a coarsening history.
- Existing multiresolution models can be seen as specific instances of an MC which are obtained from specific construction algorithms and are represented through specific data structures, usually compressed structures that exploit the peculiar characteristics of the construction method.

These features, for the first time, have made it possible to:

- Classify existing multiresolution models on the basis of the "dependency" relations that they maintain between their constituent elements, and comparing them with the well-defined notion of dependency used in the MC (see Section 4.7).
- Use the operations defined on the MC and quality measures defined for an MC as a standard platform for evaluating and comparing different multiresolution models.

  A specific multiresolution model is an instance of the MC, using specific data structures and algorithms which optimize storage space or query time (this applies to existing multiresolution models as well).

  We have demonstrated the use of the MC as a term of comparison to evaluate the loss in optimality that a specific model has to pay for improving its storage space and/or query time:

  - in Section 6.4.4, we have evaluated a compressed data structure which reduces storage space by encoding triangles in an indirect way, but pays for this in terms of a larger size in the result of a query, and a larger query time;
  - in Section 5.7, we have evaluated a dynamic extraction algorithm which exhibits a reduced query time at the expense of a sub-optimal solution.

- Compare different construction strategies for multiresolution models.

  In Section 7.5, we have evaluated different methods based on vertex insertion and deletion. Such comparative study can be extended to other refinement and coarsening algorithms with very little effort. This is an important issue since it allows performing preliminary investigations before starting the design and implementation of multiresolution models based on a certain construction technique.

9.1.2 Algorithms and Data Structures

We have proposed data structures and algorithms implementing the MC and its operations. More precisely, we have:

- Designed, classified, analyzed and compared three basic families of data structures for the description of a Multi-Complex. Such three families correspond to encoding the DAG explicitly, encoding a totally ordered sequence of its nodes, or encoding the DAG indirectly by means of adjacency information. Sample data structures have been proposed for two-dimensional simplicial MCs, which extend to higher dimensions in a straightforward way.

- Designed, classified, and analyzed schemes of compressed data structures for two-dimensional simplicial MCs built based on vertex insertion. Among them, one (the compression scheme for Delaunay complexes) extends directly to higher dimensions.

- Designed, analyzed and developed a dynamic algorithm for global queries on an MC and an algorithm for local queries. The basic algorithm for the global query described in this thesis is an interruptible extension of the algorithm proposed in [Pup96, Pup98].

In most cases, we have provided both a theoretical and an experimental evaluation of the proposed data structures and algorithms. In some others it has not been possible because an implementation was not available yet.

9.1.3 Instantiation of the MC in Applications

In Chapter 8, we have shown applications of the MC in a variety of contexts, including geographic information systems, CAD and virtual reality, solid modeling, volume visualization, and geometric compression. We remark that the applications considered do not cover all the possibilities of the model. In particular, we have:

- designed, implemented and compared history generation algorithms for two-dimensional scalar fields (terrains) (see Section 7.5);

- designed and developed algorithms for the construction of an MC, which contains a multiresolution description of lineal features of a terrain (see Section 8.2.1.1);
- designed and developed algorithms for the construction of a modular collection of MCs for terrains, which is the basis for organizing an MC on secondary storage (see Section 8.2.1.2);
- designed and developed an algorithm for the construction of an MC from parametric surfaces (see Section 8.3.2);
- designed and developed an algorithm for the construction of a Multi-Shape (see Section 8.4);
- designed and developed a compression method for triangle meshes based on the MC (see Section 8.6).

9.1.4 A Software Library

An important contribution of the thesis is the design and the development of a prototype library based on the MC which allows experimenting with the proposed data structures and algorithms.

In its current state, the library includes a two-dimensional version at a good stage of development. It implements both general-purpose and compressed data structures, algorithms for global and local queries, and a number of application-dependent operations developed on top of them (mainly in the field of geographic information systems), and history generators both for two-dimensional scalar fields and for free-form surfaces. A basic kernel of a three-dimensional library has also been developed which includes just an implementation of a general-purpose data structure and its query algorithms, and a construction algorithm based on Delaunay tetrahedralizations.

9.2 Current and Future Developments

The work presented in this thesis will be developed in the following directions:

1. on a theoretical direction, with the extension of the Multi-Complex and its operations;
2. in the field of applications, with the development of more history generators and application-dependent queries;
3. on the side of implementations, with the integration of the designed data structure and algorithms within the prototype libraries (including the ones described in this thesis which have not yet been implemented, and the ones that will be developed).

9.2.1 Theoretical Work

The set of operations of the MC data type will be extended to include *editing* of a cell complex at multiple resolutions. In multiresolution editing, the user modifies the geometry of a complex extracted from an MC at some intermediate resolution, and the changes are propagated automatically to both finer and coarser resolutions.

The basic idea for supporting editing operations in an MC is that we must not encode the geometry of each cell in an absolute way, but we have to encode it in terms of difference from cells at coarser and lower resolutions in the DAG.

For simplicity, let us consider simplicial MCs: here, geometric information is given by vertex coordinates. For instance, in a simplicial MC where nodes represent vertex insertions, we can encode the coordinates of the new vertex v_C of $\Pi_2(C)$, for each modification pattern C, as a function of the central vertices of the parents of C (e.g., a linear combination, or an offset from their mean point). Thus, changes in vertex positions at a lower resolution propagate automatically to higher resolutions. In order to propagate modifications from higher to lower resolutions, it is necessary to express the position of v_C as a function of the positions of the central vertices of the children of C.

There are non-trivial problems involved in this process, since we have to prevent or solve geometric inconsistencies that may arise in changing vertex locations.

9.2.2 Work on Applications of the MC

We will develop analysis operations on three-dimensional scalar fields. This involves defining the relevant operations of the application in terms of queries on an MC and providing algorithms to perform such operations.

We plan to develop, as a case of study, some more complex operations of terrain analysis, which use the query operations of the MC as its basic primitives for accessing the surface model. An example is visibility analysis (e.g., computation of a network of mutually visible facilities on a terrain) which will use the point visibility test described in Section 8.2.2.3.

9.2.3 Implementation Work

A number of algorithms and data structures described in this thesis are currently under implementation:

MC Data structures

We are implementing the versions of the explicit structure which store half- and full-adjacency information and related algorithms to generate adjacency information in an extracted complex (see Section 6.3.1).

The compressed data structure based on edge flips for two-dimensional MCs, and the mesh encoding and decoding algorithms based on it, (see Sections 6.4.2.2 and 8.6) are at the moment in an advanced implementation stage.

We are going implement the Multi-Shape, while the history generator described in Section 8.4 already exists.

History generators

We are implementing the construction and management of a modular set of MCs for terrains (described in Sections 6.5 and 8.2.1.2), and a coarsening history generator that preserves lineal features of a surface (described in Section 8.2.1.1). The approach on which this latter algorithm extends to triangulated free-form surfaces.

More history generators will be implemented for triangle meshes, including the one for parametric surfaces described in Section 8.3.2, and other variants of triangle mesh simplification. A history generator based on edge collapse, that can be applied to triangle meshes in free space or to two-dimensional scalar fields, is under implementation.

The history generator developed for three-dimensional fields (see Section 8.5) will be tested and improved.

9.3 Open Problems

Open problems related to the Multi-Complex are of three types:

1. overcoming theoretical limitations of the model;
2. inadeguacy of existing history generators;
3. problems connected with the design of compressed data structures for MCs;
4. problems connected with application domains of the MC.

9.3.1 Theoretical Limitations of the MC

The following aspects have not been taken into account while developing the Multi-Complex:

Representation of non-Manifold Objects

The Multi-Complex has been defined to represent *manifold* objects only.

For extending it to model non-manifold objects, it is necessary to define a broader notion of a modification pattern, which must be allowed to generate or solve non-manifold situations in a cell complex, and a new notion of dependency. The first non-trivial problem here seems to be that of capturing the change of dimensionality of some elements of a complex when passing from a manifold to a non-manifold situation.

Multiresolution of Functions

A Multi-Complex applies to the representation of cell complexes. If *functions* are defined on the cells of an MC, then these functions are at a single resolution, while multiresolution only applies to the underlying complex. This problem has an impact, for instance, in the approximation of scalar and vector fields. When approximating scalar fields, we have always assumed a fixed space of functions (namely, linear interpolants defines on simplicial cells).

This limitation is at a conceptual level: we have thought a cell complex Γ as *the model* of a geometric object; however, in some applications, a cell complex is just *a basis* for defining a model of an object, while the true model is obtained by defining a space of functions over the cells of Γ. In this case, the accuracy of a representation is determined not only by the resolution of the complex, but also by the functions used (e.g., on their degree).

Multiresolution of Attributes

The MC addresses just the *geometric aspects* of objects, disregarding their possible *attribute* values.

In practical applications, the various parts of a geometric objects may have information attached, called attributes (e.g., for rendering of surfaces, typical attributes are colors, normals, texture coordinates, etc.). The MC does not address the issue of consistency for attribute information across different resolutions. We have faced this problem with the normals used in surface visualization. For this case, we have adopted an ad-hoc heuristic solution that stores one normal for each vertex, computed at the maximum resolution, although the vertex may be visualized within a mesh at a coarser resolution. Of course, this approach may not be satisfactory for other kinds of attributes and / or in other contexts. A different problem is extending an MC to include a multiresolution description of the attributes of the represented geometric object, thus guaranteeing the consistency of attributes for any extracted cell complex.

9.3.2 Compressed MC Data Structures

The main open problem in this context is the definition of *compressed data structures for MCs of dimension greater than two.*

The need for compact encodings is even stronger in higher dimension since, for a fixed number of vertices, the number of cells in a k-dimensional MC increases with k, but, unfortunately, it also becomes harder to find working and efficient compression schemes.

The only compressed structure that extends to higher dimensions, among the ones described in Chapter 6.4, is the one based on vertex insertion in Delaunay simplicial complexes. However, such structure is limited to a specific case of MCs (for instance, such MCs cannot cover non-convex domains).

A more promising direction seems to be in the design of compressed structures for MCs built from coarsening histories based on edge collapses (that has been proposed for simplifying higher dimensional meshes as well [GS98, THJW98]). A vertex split (the refinement pattern corresponding to the inverse of an edge collapse) can be described by providing the split vertex v, the two new vertices that are the endpoints of the edge created by splitting v, and the other endpoints of the two edges, incident at v, which are expanded into triangles (see also [Hop96, Hop97, Hop98b, XV96, XESV97, DWS+97, MMS97, GTLH98, Paj98]). However, this approach imposes limitations on the way in which the model can be built. For instance, it cannot represent MCs based on Delaunay tetrahedralizations.

9.3.3 Applications of the MC

Below is a partial list of open research issues related to the application of the Multi-Complex:

Conflicting Histories

For some application domains (e.g., representation of free-form surfaces given as dense triangle meshes) it is apparently impossible or extremely difficult (computationally) to define algorithms that generate conflict-free histories (see Section 7.2).

This problem is common to all multiresolution models developed for such applications, and does not seem to have an easy solution. The only cases of algorithms capable of generating conflict-free histories are those for d-dimensional domains in $\mathbb{E}^d$, building conservative MCs, and the sculpturing approach for three-dimensional solid objects proposed in Section 8.4.

Vertex removal algorithms in higher dimensions

Defining coarsening algorithms for Delaunay tetrahedralizations, which extend the ones presented in Section 7.5 for Delaunay triangulations, is an open problem. In the case of a convex domain, an approach seems to be possible which removes a vertex p by inverting the sequence of flips performed by Joe's algorithm [Joe91] to insert p. However, this cannot work for non-convex domains.

Consistent Representation of Isolines and Isosurfaces

In Section 8.2.2.5, we have noticed that extracting an isoline at a low resolution from an MC does not produce the same result as computing a coarser representation of the isoline extracted at full resolution for the same elevation.

In order to overcome such problem, we need to build an MC for the field through a method capable of controlling the changes in the topology of the field while increasing or decreasing resolution, and not just of controlling the approximation error of individual cells.

Collision Detection with the Multi-Shape

Another problem that needs to be investigated is the design of a method for detecting collisions between two solid objects both encoded through a Multi-Shape (see Section 8.5).

Bibliography

[AAB+96] C. Andujar, D. Ayala, P. Brunet, R. Joan-Arinyo, and J. Sole'. Automatic generation of multiresolution boundary representations. *Computer Graphics Forum (Eurographics'96 Proceedings)*, 15(3):87–96, 1996.

[AD97] P.K. Agarwal and P.K. Desikan. An efficient algorithm for terrain simplification. In *Proceedings 8th ACM-SIAM Symposium on Discrete Algorithms*, 1997.

[Ago76] M.K. Agoston. Algabraic topology: a first course. In M. Dekker, editor, *Pure and Applied Mathematics*. New York, 1976.

[AS94] P.K. Agarwal and S. Suri. Surface approximation and geometric partitions. In *Proceedings 5th ACM-SIAM Symposium On Discrete Algorithms*, pages 24–33, 1994.

[AS96] M.E. Algorri and F. Schmitt. Surface reconstruction from unstructured 3D data. *Computer Graphics Forum*, 15(1):47–60, 1996.

[BBCS96] C.L. Bajaj, F. Bernardini, J. Chen, and D.R. Schikore. Automatic reconstruction of 3D CAD models. In W. Straßer, R. Klein, and R. Rau, editors, *Theory and Practice of Geometric Modeling*. Springer-Verlag, 1996.

[BBCS97] C.L. Bajaj, F. Bernardini, J. Chen, and D.R. Schikore. A triangulation-based object reconstruction method. In *6th Annual Video Review of Computational Geometry, 13th ACM Symposium on Computational Geometry*, 1997.

[BBDFP94] M. Bertolotto, E. Bruzzone, L. De Floriani, and E. Puppo. Multiresolution Representation of Volume Data through Hierarchical Simplicial Complexes. In *Aspects of Visual Form Processing*, pages 73–82. World Scientific, Singapore, 1994.

[BBX95] C. L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH '95), ACM Press*, pages 109–118, Aug. 6-12 1995.

[BCL96] C. Bajaj, E. Coyle, and K. Lin. Arbitrary topology shape reconstruction from planar cross sections. *Graphical Models and Image Processing*, 58(6):524–543, 1996.

[BDFM95a] M. Bertolotto, L. De Floriani, and P. Magillo. Overlapping hierarchical maps. In *Proceedings ACM International Workshop on Advances in Geographic Information Systems*, pages 85–92, Baltimore, Maryland, 1995.

[BDFM95b] M. Bertolotto, L. De Floriani, and P. Marzano. Pyramidal simplicial complexes. In *Proceedings 4th International Symposium on Solid Modeling*, pages 153–162, Salt Lake City, Utah, U.S.A., May 17-19 1995. ACM Press.

[BE92] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*, pages 23–90. World Scientific, 1992.

[Ber97] M. Bertolotto. *Geometric Modeling of Spatial Entities at Multiple Levels of Resolution*. PhD thesis, Department of Computer and Information Science, University of Genova, Italy, December 1997.

[BFLBM90] J. D. Boissonnat, O.D. Faugeras, and E. Le Bras-Mehlman. Representing stereo data with the Delaunay triangulation. *Artificial Intelligence*, 44:41–89, 1990.

[BG95] H. Brönnimann and M.T. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete and Computational Geometry*, 15:463–479, 1995.

[BK92] J.D. Boissonnat and K.Dobrindt. On-line construction of the upper envelope of triangles in E^3. In *Proceedings 4th Canadian Conference on Computational Geometry*, pages 311–315, August 1992.

[BM71] H. Bruggeser and P. Mani. Shellable decompositions of cells and spheres. *Math. Scand.*, 29:197–205, 1971.

[BM98] V. Bertocci and P. Magillo. A modular approach to the construction of a multiresolution surface. Technical Report DISI-TR-98-19, Department of Computer and Information Science, University of Genova (Italy), January 1998.

[Boi84] J.D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, 1984.

[Boi88] J. D. Boissonnat. Shape reconstruction from planar cross-sections. *Computer Vision, Graphics and Image Processing*, 44:1–29, 1988.

[Bow81] A. Bowyer. Computing Dirichlet tesselations. *Computer Journal*, 24:162–166, 1981.

[BR94] C. L. Bajaj and A. Royappa. Triangulation and display of rational parametric surfaces. In A. Kaufman R. Bergeron, editor, *IEEE Visualization '94 Conference*, pages 69–76, Washington, DC, 1994. IEEE Computer Society Press.

[Bri87] W.L. Briggs. *A Multigrid Tutorial.* SIAM, 1987.

[BS94] G. Barequet and M. Sharir. Piecewise linear interpolation between polygonal slices. In *Proceedings 10th ACM Symposium on Computational Geometry*, pages 93–102, 1994.

[BS96] C. L. Bajaj and D.R. Schikore. Error bounded reduction of triangle meshes with multivariate data. *SPIE*, 2656:34–45, 1996.

[BT93] J.D. Boissonnat and M. Teillaud. On the randomized construction of the Delaunay tree. *Theoretical Computer Science*, 112:339–354, 1993.

[BYG96] R. Bar-Yehuda and C. Gotsman. Time/space tradeoffs for polygon mesh rendering. *ACM Transactions on Graphics*, 15(2):141–152, 1996.

[CCMS97] A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. *The Visual Computer*, 13(5):228–246, 1997.

[CDFM+94] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, and R. Scopigno. Multiresolution modeling and rendering of volume data based on simplicial complexes. In *Proceedings 1994 Symposium on Volume Visualization*, pages 19–26. ACM Press, October 17-18 1994.

[Che89] L.P. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4:97–108, 1989.

[Che93] L.P. Chew. Guaranteed-quality mesh generation for curved surfaces. In *Proceedings 9th ACM Symposium on Computational Geometry*, pages 274–280. ACM Press, 1993.

[Cho97] M.M. Chow. Optimized geometry compression for real-time rendering. In R. Yagel and H. Hagen, editors, *IEEE Visualization '97 Proceedings*, pages 347–354. IEEE Press, 1997.

[Cig97] P. Cignoni. *Scientific Visualization Based on Simplicial Complexes.* PhD thesis, Computer Science Department, University of Pisa, Italy, December 1997.

[CMPS97] P. Cignoni, C. Montani, E. Puppo, and R. Scopigno. Multiresolution Representation and Visualization of Volume Data. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):352–369, 1997.

[CMRS97] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. Resolution modeling. Technical Report C97-02, CNUCE – C.N.R., Pisa, Italy, January 1997.

[CPS97] P. Cignoni, E. Puppo, and R. Scopigno. Representation and visualization of terrain surfaces at variable resolution. *The Visual Computer*, 13:199–217, 1997. (A preliminary version appeared on "Scientific Visualization '95", Proc. Inter. Symposium, World Scientific, pp.50-68).

[CS78] H. Christiansen and T. Sederberg. Conversion of complex contour line definition into polygonal element mosaics. *ACM Computer Graphics*, 12(3):187–192, July 1978.

[CS89] R. Cole and M. Sharir. Visibility problems for polyhedral terrains. *Journal of Symbolic Computation*, 17:11–30, 1989.

[CVM+96] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. In *Computer Graphics Proc., Annual Conf. Series (SIGGRAPH '96), ACM Press*, pages 119–128, Aug. 6-8 1996.

[Dau92] I. Daubechies. *Ten lectures on wavelets*. SIAM, Philadelphia, 1992.

[DB93] M. De Berg. Ray shooting, depth orders and hidden surface removal. In *Lecture Notes in Computer Science*, volume 703. Springer Verlag, New York, 1993.

[dBBD+96] M. de Berg, P. Bose, K. Dobrindt, M. van Kreveld, M. Overmars, M. de Groot, T. Roos, J. Snoeyink, and S. Yu. The complexity of rivers in triangulated terrains. In *Proceedings 8th Canadian Conference on Computational Geometry*, pages 325–330, Ottawa, 1996.

[dBD95] M. de Berg and K. Dobrindt. On levels of detail in terrains. In *Proceedings 11th ACM Symposium on Computational Geometry*, pages C26–C27, Vancouver (Canada), 1995. ACM Press.

[dBvK97] M. de Berg and M. van Kreveld. Trekking in the alps without freezing or getting tired. *Algorithmica*, 18:306–323, 1997.

[dBvKvOO97] M. de Berg, R. van Kreveld, R. van Oostrum, and M. Overmars. Simple traversal of a subdivision without extra storage. *International Journal of Geographic Information Science*, 11, 1997.

[Dee95] M. Deering. Geometry compression. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH '95), ACM Press*, pages 13–20, 1995.

[Del97] A. Delfino. Ricostruzione delle relazioni topologiche da dati CAD. Master's thesis, Dept. of Computer and Information Sciences, University of Genova, Genova (Italy), 1997.

[DF89] L. De Floriani. A pyramidal data structure for triangle-based surface description. *IEEE Computer Graphics and Applications*, 8(2):67–78, 1989.

[DFFJ+89] L. De Floriani, B. Falcidieno, D.M. Jung, G. Nagy, and C. Pienovi. Polyhedral terrain description using visibility criteria. Technical Report 17, Istituto per la Matematica Applicata del C.N.R., Genova, Italy, October 1989.

[DFFP85] L. De Floriani, B. Falcidieno, and C. Pienovi. Delaunay-based representation of surfaces defined over arbitrarily shaped domains. *Computer Vision, Graphics, and Image Processing*, 32:127–140, 1985.

[DFFPN84] L. De Floriani, B. Falcidieno, C. Pienovi, and G. Nagy. A hierarchical data structure for surface approximation. *Computers and Graphics*, 8(2):475–484, 1984.

[DFGMP94] L. De Floriani, G. Gattorna, P. Marzano, and E. Puppo. Spatial queries on a hierarchical terrain model. In *Proceedings 6th International Symposium on Spatial Data Handling*, pages 819–834, Edimburgh, UK, September 5-9 1994.

[DFM94a] L. De Floriani and P. Magillo. Computing visibility maps on hierarchical terrain models. In *Proceedings 2nd ACM Workshop on Advances in Geographic Information Systems*, pages 8–15, Gaithersburg, Maryland, 1994.

[DFM94b] L. De Floriani and P. Magillo. Visibility algorithms on triangulated terrain models. *International Journal of Geographic Information Systems*, 8(1):13–42, 1994.

[DFM95a] L. De Floriani and P. Magillo. Horizon computation on a hierarchical triangulated terrain model. *The Visual Computer*, 11:134–149, 1995.

[DFM95b] L. De Floriani and P. Magillo. Updating visibility information on multiresolution terrain models. In W. Kuhn A.U. Frank, editor, *Spatial Information Theory - A Theoretical Basis for GIS, Lecture Notes in Computer Science*, volume 988, pages 279–296. Springer-Verlag, 1995.

[DFM97] L. De Floriani and P. Magillo. Visibility computations on hierarchical triangulated terrain models. *Geoinformatica*, 1(3):219–250, 1997.

[DFM99] L. De Floriani and P. Magillo. Intervisibility on terrains. In D. Maguire, M.F. Goodchild, P. Longley, and D. Rhind, editors, *Geographic Information Systems: Principles, Techniques, Managament and Applications*, pages 543–556 (Chapter 38). Wiley, 1999.

[DFMP93] L. De Floriani, D. Mirra, and E. Puppo. Extracting contour lines from a hierarchical surface model. *Computer Graphics Forum (Conference Issue)*, 12(3):249–260, September 1993.

[DFMP94] L. De Floriani, P. Marzano, and E. Puppo. Multiresolution models for topographic surface description. Technical Report 1-94, Department of Information and Computer Sciences, University of Genova, ITALY, 1994.

[DFMP96a] L. De Floriani, P. Magillo, and E. Puppo. Multiresolution representation and reconstruction of triangulated surfaces. Technical Report DISI-TR-96-26, Department of Computer and Information Sciences, University of Genova, Genova, Italy, 1996.

[DFMP96b] L. De Floriani, P. Marzano, and E. Puppo. Multiresolution models for topographic surface description. *The Visual Computer*, 12(7):317–345, 1996.

[DFMP97a] L. De Floriani, P. Magillo, and E. Puppo. Building and traversing a surface at variable resolution. In *Proceedings IEEE Visualization 97*, pages 103–110, Phoenix, AZ (USA), October 1997.

[DFMP97b] L. De Floriani, P. Magillo, and E. Puppo. Multiresolution representation and reconstruction of triangulated surfaces. In *Third International Workshop on Visual Form*, Capri (Italy), May 1997.

[DFMP97c] L. De Floriani, P. Magillo, and E. Puppo. Visualizing parametric surfaces at variable resolution. In *Proceedings 9th International Conference on Image Analysis and Processing, Lecture Notes In Computer Science*, number 1311, pages 308–315, Firenze (Italy), Sept. 1997.

[DFMP98a] L. De Floriani, P. Magillo, and E. Puppo. Compressing TINs. In *Proceedings 6th ACM Workshop on Advances in Geographic Information Systems*, pages 145–150, 1998. (to appear on Geoinformatica).

[DFMP98b] L. De Floriani, P. Magillo, and E. Puppo. Compressing triangle meshes. In *Proceedings IEEE Visualization Workshop on Multiresolution Representation of 3D Geometry for Progressive Transmission*, Triangle park, North Carolina, 1998.

[DFMP98c] L. De Floriani, P. Magillo, and E. Puppo. Efficient implementation of multi-triangulations. In *Proceedings IEEE Visualization 98*, pages 43–50, Research Triangle Park, NC (USA), October 1998.

[DFMP98d] L. De Floriani, P. Magillo, and E. Puppo. Managing the level of detail in 3D shape reconstruction and representation. In *Proceedings 14th International Conference on Pattern Recognition*, pages 389–391, Brisbane, Queensland, Australia, 1998.

[DFMPB96] L. De Floriani, P. Magillo, E. Puppo, and M. Bertolotto. Variable resolution operators on a multiresolution terrain model. In *Proceedings Fourth ACM International Workshop on Advances in Geographic Information Systems*, pages 123–130, Rockville, Maryland, November, 15-16 1996.

[DFP92a] L. De Floriani and E. Puppo. An on-line algorithm for constrained Delaunay triangulation. *CVGIP, Graphical Models and Image Processing*, 54(3):290–300, 1992.

[DFP92b] L. De Floriani and E. Puppo. A hierarchical triangle–based model for terrain description. In *Theories and Methods of Spatio–Temporal Reasoning in Geographical Space, LNCS n. 639*, pages 236–251. Springer-Verlag, Pisa, (Sept. 1992).

[DFP95a] L. De Floriani and E. Puppo. Hierarchical triangulation for multiresolution surface description. *ACM Transactions on Graphics*, 14(4):363–411, October 1995.

[DFP95b] L. De Floriani and E. Puppo. Hierarchical triangulation for multiresolution surface description. *ACM Transactions on Computers*, 14(4):363–411, 1995.

[DFPM97] L. De Floriani, E. Puppo, and P. Magillo. A formal approach to multiresolution modeling. In R. Klein, W. Straßer, and R. Rau, editors, *Geometric Modeling: Theory and Practice*. Springer-Verlag, 1997.

[DFPM99] L. De Floriani, E. Puppo, and P. Magillo. Geometric structures and algorithms for geographical information systems. In J.R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*. Elsevier Science, 1999. (to appear).

[DL89] D.P. Dobkin and M.J. Laszlo. Primitives for the manipulation of three–dimensional subdivisions. *Algorithmica*, 4:3–32, 1989.

[DLS90] N. Dyn, D. Levin, and Rippa S. Data dependent triangulations for piecewise linear interpolation. *IMA Journal of Numerical Analysis*, 10:137–154, 1990.

[DWS+97] M. Duchaineau, M. Wolinsky, D.E. Sigeti, M.C. Miller, C. Aldrich, and M.B. Mineed-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. In *Proceedings IEEE Visualization'97*, pages 81–88, 1997.

[Dwy89] R.A. Dwyer. Higher dimensional Voronoi diagrams in linear expected time. In *Proceedings 5th ACM Symposium on Computational Geometry*, pages 326–333, 1989.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, 1987.

[Ede93] H. Edelsbrunner. An upper bound for conforming Delaunay triangulations. *Discrete and Computational Geometry*, 10:197–213, 1993.

[EDRD+95] M. Eck, T. De Rose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Computer Graphics Proc.,* *Annual Conf. Series (SIGGRAPH '95), ACM Press*, pages 173–181, Aug. 6-12 1995.

[EKT97] W. Evans, D. Kirkpatrick, and G. Townsend. Right triangular irregular networks. Technical Report 97-09, University of Arizona, May 1997.

[EM94] H. Edelsbrunner and E.P. Mücke. Three-Dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, January 1994. ISSN 0730-0301.

[EPO91] A.B. Ekoule, F.C. Peyrin, and C.L. Odet. A triangulation algorithm for arbitrarily shaped multiple planar contours. *ACM Transactions on Graphics*, 10(2):182–199, 1991.

[ES96] H. Edelsbrunner and N. R. Shah. Incremental topological flipping works for regular triangulations. *Algorithmica*, 15:223–241, 1996.

[ESV96] F. Evans, S. Skiena, and A Varshney. Optimizing triangle strips for fast rendering. In *Proceedings IEEE Visualization '96*, pages 319–326, 1996.

[ESV98] J. El-Sana and A. Varshney. Topology simplification for polygonal virtual environments. *IEEE Transactions on Visualization and Computer Graphics*, 2(4):133–143, 1998.

[FG90] J.P. Felleman and C. Griffin. The role of error in GIS-based viewshed determination - a problem analysis. Technical Report EIPP-90-2, Institute for Environmental Policy and Planning, State University of New York, 1990.

[Fie95] D.A. Field. The legacy of automatic mesh generation from solid modeling. *Computer Aided Geometric Design*, 12:651–673, 1995.

[FKU77] H. Fuchs, Z. Kedem, and S. Uselton. Optimal surface reconstruction from planar contours. *Communication of the ACM*, 20(10):693–702, 1977.

[FL79] R.J. Fowler and J.J. Little. Automatic extraction of irregular network digital terrain models. *ACM Computer Graphics (SIGGRAPH '79 Proceedings)*, 13(3):199–207, Aug. 1979.

[FN94] Thomas A. Foley and Gergory M. Nielsen. Modeling of scattered multivariate data. In C. Giertsen and P. A. Fevang, editors, *State of the Art Reports of Eurographics '94*, pages 38–59, 1994.

[For87] S. Fortune. A sweep-line algorithm for Voronoi diagrams. *Algorithmica*, 2(2):153–174, 1987.

[For92] S. Fortune. Voronoi diagrams and Delaunay triangulations. In D.-Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry – Lecture Notes Series on Computing*, volume 1, pages 193–234. World Scientific, 1992.

[FP91] V. Ferrucci and A. Paoluzzi. Extrusion and boundary evaluation for multi-dimensional polyhedra. *Computer Aided Design*, 23(1):40–50, February 1991. Special issue entitled *Beyond Solid Modeling*.

[GD82] S. Ganapathy and T.G. Dennehy. A new general triangulation method for planar contours. *Computer Graphics*, 16(3):69–75, 1982.

[Gei93] B. Geiger. *Three dimensional modeling of humans organs and its application to diagnosis and surgical planning*. PhD thesis, INRIA, 06902 Sophia Antipolis, France, 1993.

[GG79] D. Gomez and A. Guzman. Digital model for three-dimensional surface representation. *Geo-Processing*, 1:53–70, 1979.

[GH95] M. Garland and P.S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, School of Computer Sciences, Carnegie Mellon University, Pittsburgh, PA, USA, 1995.

[GH97] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH '97), ACM Press*, pages 209–216, 1997.

[GH98] M. Garland and P.S. Heckbert. Simplifying surfaces with color and normals with quadric error metrics. In *Proceedings IEEE Visualization'98*, pages 263–269, Research Triangle Park, NC, 1998.

[GHJ+97] T.S. Gieng, B. Hamann, K.I. Joy, G.L. Schussman, and I.J. Trotts. Constructing hierarchies of triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):145–160, 1997.

[GKS90] L.J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoy diagrams. In *Automata, Languages and Programming, LNCS N.443*, pages 414–431. Springer-Verlag, 1990.

[GKS92] L.J. Guibas, D.E. Knuth, and M. Sharir. Randomized incremental construction of the Delaunay and Voronoi diagrams. *Algorithmica*, 7:381–413, 1992.

[GLE97] R. Grosso, C. Luerig, and T. Ertl. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In *IEEE Visualization '97*, pages 387–394, Phoenix, AZ, October 19-24 1997.

[Gro94] OpenInventor Architecture Group. *Inventor Mentor: OpenInventor Reference Manual*. Addison Wesley, 1994.

[GS85] L. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Transaction on Graphics*, 4(2):74–123, April 1985.

[GS98] M.H. Gross and O.G. Staadt. Progressive tetrahedralizations. In *Proceedings IEEE Visualization'98*, pages 397–402, Research Triangle Park, NC, 1998. IEEE Comp. Soc. Press.

[GSG96] M.H. Gross, O.G. Staadt, and R. Gatti. Efficient triangular surface approximations using wavelets and quadtree data structures. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):130–144, June 1996.

[GTLH98] A. Guéziec, G. Taubin, F. Lazarus, and W. Horn. Simplicial maps for progressive transmission of polygonal surfaces. In *Proceeding ACM VRML98*, pages 25–31, 1998.

[Gué96] A. Guéziec. Surface simplification inside a tolerance volume. Technical Report RC 20440, I.B.M. T.J. Watson Research Center, 1996.

[Gut84] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings ACM SIGMOD Conference*, pages 47–57, 1984.

[Ham94] B. Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11(2):197–214, 1994.

[HC94] B. Hamann and J.L. Chen. Data point selection for piecewise trilinear approximation. *Computer Aided Geometric Design*, 11:477–489, 1994.

[HDD+92] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuelzle. Surface reconstruction from unorganized points. In *ACM Computer Graphics Proceedings, Annual Conference Series, (SIGGRAPH '92)*, pages 71–78, 1992.

[HDD+93] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *ACM Computer Graphics Proceedings, Annual Conference Series, (SIGGRAPH '93)*, pages 19–26, 1993.

[HDD+94] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *ACM Computer Graphics Proceedings, Annual Conference Series, (SIGGRAPH '94)*, pages 295–302, 1994.

[Heb94] D.J. Hebert. Symbolic local refinement of tetrahedral grids. *Journal of Symbolic Computation*, 17:457–472, 1994.

[HG94] P. Heckbert and M. Garland. Multiresolution Modeling for Fast Rendering. In *Graphics Interface '94 Proceedings*, pages 43–50, 1994.

[HH93] P. Hinker and C. Hansen. Geometric optimization. In *IEEE Visualization '93 Proceedings*, pages 189–195, October 1993.

[HHK+95] T. He, L. Hong, A. Kaufman, A. Varshney, and S. Wang. Voxel-based object simplification. In *IEEE Visualization '95 Proceedings*, pages 296–303. IEEE Comp. Soc. Press, 1995.

[HHVW96] T. He, L. Hong, A. Varshney, and S. Wang. Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–183, 1996.

[HK95] D.J. Hebert and H.-J. Kim. Image encoding with triangular wavelets. In *Proceedings SPIE*, volume 2569(1), pages 381–392, 1995.

[Hof89] C.M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann, San Mateo, California, 1989. The Morgan Kaufmann series in computer graphics and geometric modeling.

[Hop96] H. Hoppe. Progressive meshes. In *ACM Computer Graphics Proceedings, Annual Conference Series, (SIGGRAPH '96)*, pages 99–108, 1996.

[Hop97] H. Hoppe. View-dependent refinement of progressive meshes. In *ACM Computer Graphics Proceedings, Annual Conference Series, (SIGGRAPH '97)*, pages 189–198, 1997.

[Hop98a] H. Hoppe. Efficient implementation of progressive meshes. *Hoppe. Computers & Graphics*, 22:27–36, 1998.

[Hop98b] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings IEEE Visualization'98*, pages 35–42, Research Triangle Park, NC, 1998. IEEE Comp. Soc. Press.

[HS95] B. Hamann and R.F. Sarraga, editors. *Special issue on Grid Generation, Finite Elements, and Geometric Design*, volume 12(7). Elsevier, 1995.

[HT95] M.J. Herbert and J.G. Tough. Surface reconstruction from complex earth science data. In *Proceedings 13th Eurographics*, pages 84–106, Loughborough, UK, March 1995.

[HZ91] M.J. De Haemer and M.J. Zyda. Simplification of objects rendered by polygonal approximations. *Computers & Graphics*, 15(2):175–184, 1991.

[Joe91] B. Joe. Construction of three-dimensional Delaunay triangulations using local transformations. *Computer Aided Geometric Design*, 8:123–142, 1991.

[Kau90] A. Kaufman. *Volume Visualization*. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[Kau93] A. Kaufman, editor. *Volume Visualization (ACM SIGGRAPH Course Notes No.41)*. ACM Press, 1993.

[KCVS98] L. Kobbelt, S. Campagna, J. Vorsatz, and H.P. Seidel. Interactive multiresolution modeling of arbitrary meshes. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH '98), ACM Press*, 1998.

[Kep75] E. Keppel. Approximating complex surfaces by triangulation of contour lines. *IBM Journal of Research and Development*, 19(1):2–11, 1975.

[Kir83] D.G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal of Computing*, 12:28–35, 1983.

[KLY92] A. Kaufman, W. Lorensen, and R. Yagel, editors. *Introduction to Volume Visualization (Visualization '92 Tutorial No.1)*. IEEE Com. Soc. Press, 1992.

[KMS91] T. Kao, D. Mount, and A. Saalfeld. Dynamic maintenance of Delaunay triangulations. In *Proceedings Auto-Carto 10*, pages 219–233, 1991.

[KOS91] M.J. Katz, M.H. Ovemmars, and M. Sharir. Efficient hidden surface removal for objects with small union size. In *Proceedings 7th ACM Symposium on Computational Geometry*, pages 31–40, 1991.

[KS95] R. Klein and W. Straßer. Large mesh generation from boundary models with parametric face representation. In C. Hoffmann and J. Rossignac, editors, *Proceedings 3rd ACM Symposium on Solid Modeling and Applications*, pages 431–440, 1995.

[KS97] R. Klein and W. Straßer. Generation of multiresolution models from CAD data for real time rendering. In R. Klein, W. Straßer, and R. Rau, editors, *Theory and Practice of Geometric Modeling (Blaubeuren II)*. Spinger-Verlag, 1997.

[KT96] A. D. Kalvin and R.H. Taylor. Superfaces: Poligonal mesh simplification with bounded error. *IEEE C.G.&A.*, 16(3):64–77, 1996.

[Law77] C.L. Lawson. Software for C1 Surface Interpolation. In J.R. Rice, editor, *Mathematical Software III*, pages 161–164. Academic Press, 1977.

[LC87] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–170, July 1987.

[LDW97] M. Lounsbery, T.D. DeRose, and J. Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics*, 16(1):34–73, 1997.

[LE97] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *ACM Computer Graphics Proceedings, Annual Conference Series, (SIGGRAPH '97)*, pages 199–207, 1997.

[Lee89] J. Lee. A drop heuristic conversion method for extracting irregular networks from digital elevation models. In *Proceedings GIS/LIS'89*, pages 30–39, Orlando, FL, USA, 1989.

[LKR+96] P. Lindstrom, D. Koller, W. Ribarsky, L.F. Hodges, N. Faust, and G.A. Turner. Real-time, continuous level of detail rendering of height fields. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH '96), ACM Press*, pages 109–118, New Orleans, LA, USA, Aug. 6-8 1996.

[LL86] D.T. Lee and A.K. Lin. Generalized Delaunay triangulation for planar graphs. *Discrete and Computational Geometry*, 1:201–217, 1986.

[LMS97] M. Lanthier, A. Maheshwari, and J.R. Sack. Approximating weighted shortest paths on polyhedral surfaces. In *Proceedings ACM Symposium on Computational Geometry*, pages 274–283, Nice, France, 1997.

[Lou94] M. Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, Dept. of Computer Science and Engineering, U. of Washington, 1994.

[LS80] D.T Lee and B.J. Schachter. Two algorithms for constructing a Delaunay triangulation. *Int. J. of Computer and Information Science*, 9(3):219–242, 1980.

[LS98] M. Lee and H. Samet. Navigating through triangle meshes implemented as linear quadreees. Technical Report CS-TR-3900, University of Maryland, College Park, MD, April 1998.

[LT97] K.L. Low and T.S Tan. Model simplification using vertex clustering. In *1997 ACM Symposium on Interactive 3D Graphics*, 1997.

[LT98] P. Lindstrom and G. Turk. A fast and memory efficient polygonal simplification. In *Proceedings IEEE Visualization'98*, pages 287–295, Research Triangle Park, NC, 1998.

[Mal89] S. G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. on Patt. Anal. and Mach. Intel.*, 11(7):674–693, 1989.

[MMS97] A. Maheshwari, P. Morin, and J.-R. Sack. Progressive TINs: Algorithms and applications. In *Proceedings 5th ACM Workshop on Advances in Geographic Information Systems, Las Vegas*, 1997.

[Mun84] J.R. Munkres. *Elements of Algebraic Topology*. Addison Wesley, 1984.

[Nag94] G. Nagy. Terrain visibility. *Computers and Graphics*, 18(6):763–773, 1994.

[NHM97] G.M. Nielson, H.Hagen, and H. Müller. *Scientific Visualization: Overviews, Metodologies, Techniques.* IEEE Computer Society, 1997.

[Nie93] G.M. Nielson. Scattered data modeling. *IEEE Computer Graphics & Applications*, 13(1):60–70, 1993.

[Nie97] G.M. Nielson. Tools for triangulations and tetrahedralizations and constructing functions defined over them. In G.M. Nielson, H.Hagen, and H. Müller, editors, *Scientific Visualization: Overviews, Metodologies, Techniques*, pages 429–525. IEEE Computer Society, 1997.

[NORS97] R. Neubauer, M. Ohlberger, M. Rumpf, and R. Schwirer. Efficient visualization of large–scale data on hierarchical meshes. In W. Lefer and M. Grave, editors, *Visualization in Scientific Computing '97.* Springer Wien, 1997.

[Ore91] J. Orenstein. An algorithm for computing the overlay of k-dimensional spaces. In O. Günter and H.J. Schek, editors, *Lecture Notes in Computer Science*, volume 525, pages 381–400. 1991.

[OS92] M.H. Overmars and M. Sharir. A simple output-sensitive algorithm for hidden surface removal. *ACM Transactions on Graphics*, 11:1–11, 1992.

[Paj98] R. Pajarola. Large scale terrain visualization using the restricted quadtree triangulation. In *Proceedings IEEE Visualization'98*, pages 19–24, Research Triangle Park, NC, 1998. IEEE Comp. Soc. Press.

[PBCF93] A. Paoluzzi, F. Bernardini, C. Cattani, and V. Ferrucci. Dimension-independent modeling with simplicial complexes. *ACM Transactions on Graphics*, 12(1):56–102, January 1993.

[PH97] J. Popovic and H. Hoppe. Progressive simplicial complexes. In *ACM Computer Graphics Proceedings, Annual Conference Series, (SIGGRAPH '97)*, pages 217–224, 1997.

[PJ92] F.P. Preparata and J.S.Vitter. A simplified technique for hidden line elimination in terrains. In *Proceedings STACS'92*, pages 193–200, Paris, February 1992.

[PK95] H. Park and K. Kim. An adaptive method for smooth surface approximation to scattered 3D points. *Computer-Aided Design*, 27(12):929–939, December 1995.

[PT96] L. Piegl and W. Tiller. *The NURBS Book.* Springer, 1996.

[Pup95] E. Puppo. Segmentation/reconstruction of range images based on piecewise-linear approximation. In C. Braccini, L. De Floriani, and G. Vernazza, editors, *Image Analysis and Processing*, number 974 in Lecture Notes in Computer Science, pages 367–372. Springer-Verlag, 1995.

[Pup96] E. Puppo. Variable resolution terrain surfaces. In *Proceedings Eight Canadian Conference on Computational Geometry*, pages 202–210, Ottawa, Canada, August 12-15 1996.

[Pup98] E. Puppo. Variable resolution triangulations. *Computational Geometry Theory and Applications*, 11(3-4):219–238, December 1998.

[Raj91] V.T. Rajan. Optimality of the Delaunay triangulation in R^d. In *Proceedings 7th ACM Symposium on Computational Geometry*, pages 357–363, 1991.

[RB93] J. Rossignac and P. Borrel. Multi-resolution 3D approximation for rendering complex scenes. In B. Falcidieno and T.L. Kunii, editors, *Geometric Modeling in Computer Graphics*, pages 455–465. Springer Verlag, 1993.

[Red96] M. Reddy. Scrooge: Perceptually-driven polygon reduction. *Computer Graphics Forum*, 15(4):191–203, 1996.

[Rip88] S. Rippa. Adaptive approximations by piecewise linear polynomials on triangulations of subsets of scattered data. *SIAM Journal on Scientific and Statistic Computing*, 13(1):1123–1141, 1988.

[Rip90] S. Rippa. Minimal roughness property of the Delaunay triangulation. *Computer Aided Geometric Design*, 7(7):489–497, 1990.

[RO96] K.J. Renze and J.H. Oliver. Generalized unstructured decimation. *IEEE C.G.&A.*, 16(6):24–32, 1996.

[Ros97] J. Rossignac. *Simplification an compression of 3D scenes (Eurographics '97 Tutorial Notes)*. Eurographics, 1997.

[RR96] R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum (Eurographics'96 Proceedings)*, 15(3):67–76, 1996.

[RS88] J.H. Reif and S. Sen. An efficient output–sensitive HSR algorithm and its parallelization. In *4th ACM Symposium on Computational Geometry*, pages 193–200. A.C.M. Press, 1988.

[RS89] J. Ruppert and R. Seidel. On the difficulty of tetrahedralizing 3-dimensional non-convex polyhedra. In *Proceedings 5nd ACM Symposium on Computational Geometry*, pages 380–392, 1989.

[Saa91] A. Saalfeld. Delaunay edge refinement. In *Proceedings 3rd Canadian Conference on Computational Geometry*, pages 33–36, 1991.

[Sam90a] H. Samet. *Applications of Spatial Data Structures*. Addison Wesley, Reading, MA, 1990.

[Sam90b] H. Samet. *The design and Analysis of Spatial Data Structures*. Addison Wesley, Reading, MA, 1990.

[Sch97] W.J. Schroeder. A topology modifyinfg progressive decimation algorithm. In R. Yagel and H. Hagen, editors, *Proceedings IEEE Visualization'97*, pages 205–212, 1997.

[SDS96] E.J. Stollnitz, T.D. DeRose, and D.H. Salesin. *Wavelets for Computer Graphics: Theory and Applications*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann, San Francisco, Ca, 1996.

[SH81] K.R. Sloan and L.D. Hrechanyk. Surface reconstruction from sparse data. In *Proceedings Conference on Pattern Recognition and Image Processing*, pages 45–48, 1981.

[SH92] X. Sheng and B.E. Hirsch. Triangulation of trimmed surfaces in parametric space. *Computer Aided Design*, 24(8):437–444, 1992.

[Sha81] M. Shantz. Surface definition for branching contour-defined objects. *Computer Graphics*, 15(2):242–270, 1981.

[SK91] Y. Shinagawa and T.L. Kunii. The homotopy model: a generalized model for smooth surface generation from cross sectional data. *The Visual Computer*, 7:72–86, 1991.

[SL96] M. Soucy and D. Laurendeau. Multiresolution surface modeling based on hierarchical triangulation. *Computer Vision and Image Understanding*, 63(1):1–14, 1996.

[Smo97] D. Smoquina. Una interfaccia polivalente orientata agli oggetti per applicazioni geometriche. Master's thesis, Dept. of Computer and Information Sciences, University of Genova, Genova (Italy), 1997.

[Sou91] D.A. Southard. Piecewise linear surface models from sampled data. In *Proceedings Computer Graphics International '91*, pages 667–680, June 22-28 1991.

[SP92] L. Scarlatos and T. Pavlidis. Hierarchical triangulation using cartographics coherence. *CVGIP: Graphical Models and Image Processing*, 54(2):147–161, March 1992.

[SS92] R. Sivan and H. Samet. Algorithms for constructing quadtree surface maps. In *Proceedings 5th International Symposium on Spatial Data Handling*, pages 361–370, Charleston, 1992.

[SvK97] J. Snoeyink and M. van Kreveld. Linear-time reconstruction of Delaunay triangulations with applications. In *Proceedings 5th European Symposium on Algorithms*, 1997.

[SW87] L. Schubert and C.A. Wang. An optimal algorithm for constructing the Delaunay triangulation of a set of line segments. In *Proceedings 3rd ACM Symposium on Computational Geometry*, pages 223–232, Waterloo, 1987.

[SZL92] W.J. Schroeder, J.A. Zarge, and W.E. Lorensen. Decimation of triangle meshes. In Edwin E. Catmull, editor, *ACM Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 65–70, July 1992.

[TG98] C. Touma and C. Gotsman. Triangle mesh compression. In *Proceedings Graphics Interface'98*, pages 26–34, 1998.

[THJW98] I.J. Trotts, B. Hamann, K.I. Joy, and D.F. Wiley. Simplification of tetrahedral meshes. In *Proceedings IEEE Visualization'98*, pages 287–295, Research Triangle Park, NC, 1998.

[TR96] G. Taubin and J. Rossignac. Geometric compression through topological surgery. Technical Report TR RC-20340(n.89924) 01/16/96, IBM Research Report, Yorktown (NY), January 1996.

[Tur92] G. Turk. Re-tiling polygonal surfaces. In Edwin E. Catmull, editor, *ACM Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 55–64, July 1992.

[VB93] M. Vigo and P. Brunet. Piecewise linear approximation of trimmed surfaces. In G. Farin, H. Hagen, and H. Noltemeier, editors, *Geometric Modelling*. 1993.

[Vel93] R.C. Veltkamp. 3D Computational Morphology. *Computer Graphics Forum*, 12(3):115–127, Sept. 1993.

[VHB87] B. Von Herzen and A.H. Barr. Accurate triangulations of deformed, intersecting surfaces. *Computer Graphics (SIGGRAPH 87 Proceedings)*, 21(4):103–110, 1987.

[vK94] M. van Kreveld. On quality paths on polyhedral terrains. In J. Nievergelt, H.J. Roos, T. amd Schack, and P. Widmayer, editors, *Lecture Notes in Computer Science*, volume 884, pages 113–122. Springer-Verlag, 1994. Proceedings IGIS'94: Geographic Information Systems.

[VRM96] *The Virtual Reality Modeling Language Specification - Version 2.0*, August 1996.

[WA86] Y.F. Wang and J.K. Aggarwal. Surface reconstruction and representation of 3D scenes. *Pattern Recognition*, 19(3):197–207, 1986.

[Wat81] D.F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal*, 24:728–746, 1981.

[Wer94] J. Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor*. Addison Wesley, 1994.

[Wil93] P.L. Williams. *Interactive Direct Volume Rendering of Curvilinear and Unstructured Data*. PhD thesis, University of Illinois at Urbana–Champaign, 1993.

[WLH85] L.T. Watson, T.J. Laffey, and R.M. Haralick. Topographic classification of digital image intensity surfaces using generalized splines and the discrete cosine transformation. *Computer Vision, Graphics and Image Processing*, 29:143–167, April 1985.

[WMW86] G. Wyvill, C. McPheeters, and B. Wyvill. Data structures for soft objects. *The Visual Computer*, 2(4):227–234, August 1986.

[WvG94] J. Wilhelms and A. van Gelder. Multi-dimensional trees for controlled volume rendering and compression. In *Proceedings 1994 Symposium on Volume Visualization*, pages 27–34. ACM Press, October 17-18 1994.

[XESV97] J.C. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):171–183, 1997.

[XV96] J.C. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In R. Yagel and G. Nielson, editors, *IEEE Visualization '96 Proceedings*, pages 327–334, 1996.

[Yag93] R. Yagel. Volume viewing: state of the art survey. In *ACM SIGGRAPH '93 Course Notes - Volume Visualization*, volume 41, pages 109–129, 1993.

[YVS96] S. Yu, M. Van, and J. Snoeyink. Drainage queries in TINs: from local to global and back again. In *Proceedings Symposium on Spatial Data Handling (SDH'96)*, pages 13A.1–13A.14, 1996.

[ZSS97] D. Zorin, P. Schröder, and W. Sweldens. Interactive multiresolution mesh editing. In *Comp. Graph. Proc., Annual Conf. Series (SIGGRAPH '97), ACM Press*, pages 259–268, 1997.

Appendix A

Index of Terms and Symbols

This appendix provides a quick reference to the places in the thesis where certain concepts, terms, symbols and notations are defined.

A.1 Terms

Accuracy: – of a cell complex: see Section 2.4.3.3.

Active: – cell with respect to a focus set, see Definition 5.2; strongly –, loosely –, see Definition 5.7.

Adjacency relation: – in a cell complex, see Section 2.3.1; – in a Multi-Complex, see Definitions 4.23 and 4.24.

Approximation error: see Section 2.4.3.3; – based on Hausdorff distance, see Definitions 2.31 and 2.32; – for fields, see Definitions 2.33 and 2.34.

Boundary: – of a uniformly k-dimensional set, see Section 2.2.1; – complex of a cell complex, see Section 2.3.1; combinatorial – of a cell, see Section 2.3.1.

Boundary representation, or BRep: see Section 2.5.2.2.

Carrier: – of a set of cells, or of a complex, see Section 2.3.1.

CDT: Constrained Delaunay Triangulation, see Definition 2.23.

Cell: see Definition 2.14; top- –, see Section 2.3.1.

Cell Complex: see Definition 2.15.

Coarsening pattern: see Section 4.2.2.

Co-boundary: combinatorial –, see Section 2.3.1.

Combined cell complex: – of an MC, see Definition 4.19; global –, see Definition 4.20; partial –, not a global one, see Section 4.4.

Complex: see cell complex.

Compression power: see Definition 7.4.

Conflict: – between modification patterns, see Definition 4.10.

Conflict-free: – set or sequence of modification patterns, see Definition 4.11; – history, see Definition 7.3.

Conforming: – triangulation, see Definition 2.21. – Delaunay triangulation: see Definition 2.24.

Conservative: – Multi-Complex, see Definition 4.21.

Consistent permutation: see Definition 4.13.

Constrained: – triangulation, see Definition 2.20. – Delaunay triangulation (CDT): see Definition 2.23.

k-dimensional: at least –, see Definition 2.1. uniformly –, see Definition 2.2. – cell, – cell complex, etc., see the corresponding term.

Cuboid: see Definition 2.25.

Conflict-free: – set of modification patterns, see Definition 4.11; – history, see Definition 7.3.

Conflicting: see conflict-free.

Delaunay: – simplicial complex: see Definition 2.22.

Dependency: – between modification patterns, see Definition 4.8.

Destination: – of a cell in an (enriched) MC, see Definition 4.17.

Direct dependency: – between modification patterns, see Definition 4.7.

Drain: – of an enriched MC, see Section 4.3.2.

Enriched Multi-Complex: or Enriched MC, see Definition 4.16.

Explicit surface: see Definition 2.9.

Error: see approximation error.

Error threshold function: see Definition 2.35.

Face: – of a cell, see Definition 2.16.

Feasible: – cell for a resolution filter, see Definition 5.1.

Field: scalar –, see Definition 2.7; vector –, see Definition 2.8; approximation error evaluation for –, see approximation error.

Fixed-pattern: – cell complex, see Definition 2.26.

Focus set: see Section 5.2.

Front: – cell complex of an MC, see Definition 4.14; – arcs of an MC, see Section 4.3.2.

Global combined cell complex: see Definition 4.20.

Grid: see Section 2.3.3.

Growth: linear –, see Definition 4.32.

Hausdorff distance: see Definition 2.30; approximation error evaluation based on –, see approximation error.

Height: – of an MC, see Definition 4.26; logarithmic –, see Definition 4.31.

History: see Definition 7.1; coarsening –, refinement –, see Section 7.2.

Incidence relations: see Section 2.3.1.

Indexed data structure: see Section 2.3.6.

Interference: see Section 5.2.2.

Interior: – of a uniformly k-dimensional set, see Section 2.2.1.

Loose: – interference, – -ly active cell, – local query, – restriction of a resolution filter, see Section 5.3.3.

Manifold: – without boundary: see Definition 2.3; – with boundary: see Definition 2.4; – cell complex: see Section 2.3.1.

MC: see Multi-Complex.

MC-node: node of an MC representing a modification pattern, see Section 4.3.

MC-front: see Definition 4.18; pseudo- –, see Section 5.5.2.

Mesh: triangle –, tetrahedral – (in the context sometimes simply –), see Section 2.3.2.

Minimal: – modification pattern, see Definition 4.4.

Modification pattern: see Definition 4.1; result of applying a –, see Definition 4.2.

Multi-Complex, or MC: see Definition 4.12; enriched –, see Definition 4.16.

Non-redundant: – modification pattern with respect to a set of modification patterns, see Definition 4.9; – sequence of modification patterns, see Subsection 4.2.3.1.

Parametric patch: see Definition 2.10.

Parametric surface: see Definition 2.11.

Partial combined cell complex: see Section 4.4.

Raw data structure: see Section 2.3.6.

Refinement pattern: see Section 4.2.2.

Regular: – cell complex, see Definition 2.17.

Regular Square Grid, or RSG: see Section 2.5.1.

Resolution: – of a cell complex, see Definition 2.28.

Resolution filter: see Section 5.2.1; – induced by an error threshold function, see Section 5.2.1.

Restriction: – of a resolution filter to an area of intersest (or to a focus set), see Section 5.2.1; loose or strict –, see 5.3.3.

Reverse: – of a modification pattern, see Definition 4.5.

Root: – of an MC, see Section 4.3.

Scalar field: see field.

Self-similar: – complex, see Section 2.3.5.

Self-similarity: – pattern, see Section 2.3.5.

Simplex: see Definition 2.19.

Simplicial: – complex, see Definition 2.19; – MC, see Section 4.3.

Size: – of a cell, see Section 2.3.7; – of a regular cell complex, see Definition 2.18; – of an MC, see Definition 4.28.

Solid: see Definition 2.6.

Source: – of a cell in an (enriched) MC, see Definition 4.17.

Star: see co-boundary.

Strict: – interference, – -ly active cell, – local query, – restriction of a resolution filter, see Section 5.3.3.

Sub-MC: see Definition 4.15; pseudo- –, see Section 5.5.2.

Surface: see Definition 2.5.

Surface-on-surface: see Definitions 2.12 and 2.13.

Tetrahedralization: see Section 2.3.2.

TIN: see Triangulated Irregular Network.

Triangulated Irregular Network, or TIN: see Section 2.5.1.

Triangulation: see Section 2.3.2.

Target object: the object represented by a cell complex, see Section 2.4.1.5.

Triangle mesh, or triangulation: see Section 2.3.2.

Tetrahedral mesh, or tetrahedralization: see Section 2.3.2.

Terrain: see Section 2.5.1.

Top cell: see Section 2.3.1.

Valid: – modification pattern for a complex, see Definition 4.3; – sequence of modification pattern, see Definition 4.6; – history, see Definition 7.2.

Vector field: see field.

Vertex: – of a cell or of a cell complex: see Section 2.3.1.

Warping: see Section 2.4.3.2.

Width: – of an MC, see Definition 4.27; bounded –, see Definition 4.30.

A.2 Symbols

Throughout the thesis, these symbols may appear with subscripts or superscripts. The arguments of operators are indicated with $-$.

- $-^1[-^2]$: result of applying modification pattern $-^1$ to cell complex $-^2$.
- A: arc in the (enriched) MC.
- $\mathcal{A}$: subset of arcs of an (enriched) MC, sometimes the arcs of an MC-front or a pseudo-MC-front.
- $\mathbb{B}$: the set of the Boolean numbers.
- C: modification pattern, or MC-node.
- $\mathcal{C}$: set of modification patterns or of MC-nodes.
- γ: cell.
- Γ: set of cells, cell complex.

- $\Gamma^{(s)}$: set of cells of dimension k from complex Γ.
- $\mathbb{\Gamma}_d$: set of all the cells of $\mathbb{E}^d$.
- $\mathbb{\Gamma}_d^{(k)}$: set of all the k-cells of $\mathbb{E}^d$.
- $d(-^1, -^2)$: Eucliden distance between two points $-^1$ and $-^2$.
- $\Delta(-)$: carrier of a cell complex $-$.
- $\partial(-)$: boundary complex of cell complex $-$.
- $\mathbb{E}^d$: the d-dimensional Euclidean space.
- ε: approximation error (function defined piecewise on the cells of a complex).
- e: edge (in a triangulation).
- f: function defined piecewise on the cells of a complex typically to approximate a field.
- front$(-)$: front cell complex of MC $-$.
- F: focus set.
- ϕ: function, scalar or vector field.
- $\mathcal{M}$: Multi-Complex (MC), sub-MC.
- $\mathbb{N}$: set of the integer numbers.
- $\mathcal{N}$: subset of nodes of an (enriched) MC, sometimes the nodes of an MC-front or a pseudo-MC-front;
- $\mathcal{O}$: manifold object (surface or solid), usually the object approximated through a cell complex.
- o: subset of a manifold object, generic entity in space.
- p, q: points, sometimes vertices of a complex.
- $\Pi_1(-)$,$\Pi_2(-)$: first and second complex in the pair forming a modification pattern $-$.
- S: used in general for a set of points.
- σ: simplex.
- Σ: set of simplices, simplicial complex, triangulation, tetrahedralization.
- size$(-)$: size of regular cell complex $-$.
- sub$(-^1, -^2)$: sub-complex induced by subset $-^1$ of cells of complex $-^2$.

- $\mathbb{R}$: the set of real numbers.
- R: resolution filter.
- τ: error threshold function.
- v, w: points, sometimes vertices of a complex.
- Ω: domain of a (scalar or vector) field.