Technical Report

Pattern Recognition and Image Processing Group
Institute of Computer Aided Automation
Vienna University of Technology
Favoritenstr. 9/183-2
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801-18351
Fax: +43 (1) 58801-18392
E-mail: illetsch@prip.tuwien.ac.at

URL: http://www.prip.tuwien.ac.at/

PRIP-TR-110 November 3, 2006

Minimal Combinatorial Maps for analyzing 3D Data [1] [2]
Diploma Thesis

T. Illetschko

Abstract

Combinatorial maps and irregular pyramids based on combinatorial maps for 2D data have been studied in great detail. It has been shown that this concept is advantageous for many applications in the field of image processing and pattern recognition by providing means to store information of the topological relation of the represented data. While the concept of combinatorial maps has been defined for any dimension, most of the studies concentrated on the representation of two dimensional data and only few results exist regarding higher dimensions. This report studies the properties of combinatorial maps for 3D data. Especially collapsing an initial map of the volumetric data by applying contraction and removal operations to produce a minimal representation while preserving the topological relations is presented in this report. Formal conditions for applying these operations as well as the minimal configurations of the topological relations found in volumetric data are presented in this report and means for discriminating and identifying these minimal configurations using pseudo elements are introduced.

[1]The support of the K plus Competence Center ADVANCED COMPUTER VISION is greatfully acknowledged.

[2]This work was partially supported by the Austrian Science Fund under grants S9103-N04 and P18716-N13.

Contents

1 Introduction **8**

2 Recall of Combinatorial Maps **11**
2.1 2D Combinatorial Maps 12
2.2 Dual Combinatorial Maps 14
2.3 i-cells, Degree, Dual-degree 14
2.4 Bounding Relationship Diagram 18
2.5 Initial Map 19
2.6 Contraction and Removal Operations 21
2.7 2D Combinatorial Pyramids 22

3 3D Combinatorial Maps **24**
3.1 i-cells 25
3.2 Degree and Dual-Degree 26
3.2.1 0-cells (Vertices) 27
3.2.2 1-cells (Edges) 27
3.2.3 2-cells (Faces) 28
3.2.4 3-cells (Volumes) 29

4 Contraction and Removal in 3D **31**
4.1 Contraction Operation 32
4.1.1 Condition 1: Dual Degree 34
4.1.2 Condition 2: Bounding and Incidence Relations 35
4.1.3 Condition 3: Self-Loops 36
4.1.4 Condition 4: Common $(i-2)$-cells 37
4.1.5 Condition 5: Background vertex 40
4.1.6 Edge Contraction 40
4.1.7 Face Contraction 42
4.1.8 Volume Contraction 45
4.1.9 Simplification of a Cube 47
4.2 Removal Operation 50
4.2.1 Condition 6: Degree 53

4.2.2 Condition 7: Bounding and Incidence Relations 53
4.2.3 Condition 8: Bridges 53
4.2.4 Condition 9: Common $(i+2)$-cells 54
4.2.5 Condition 10: Connected Vertices 55
4.2.6 Vertex Removal 56
4.2.7 Edge Removal 58
4.2.8 Face Removal 60

5 3D Combinatorial Pyramids 63
5.1 Describing 3D Data 63
5.1.1 Size of the Initial Combinatorial Map 65
5.2 Building the Pyramid 69
5.2.1 Algorithm Step 69
5.2.2 Simplification Step 72
5.2.3 Simplification with Contraction Operations 74
5.2.4 Simplification with Removal Operations 75
5.2.5 Building the next level 79
5.3 Receptive Fields 81

6 Configurations and Topological Relations 86
6.1 Minimal 3D Combinatorial Maps 86
6.2 Minimal Configurations 89
6.2.1 Edges 90
6.2.2 Faces 94
6.2.3 Volumes 95
6.3 Topological Relations 98
6.3.1 Adjacency 99
6.3.2 Surrounds 100
6.3.3 Contains 104
6.4 Invalid Configurations 106

7 Experimental Results 109
7.1 Construction of 3D Combinatorial Pyramids 109
7.1.1 Experimental Setup 109
7.1.2 Simplex 112
7.1.3 Tunnel 113
7.1.4 Hole 116
7.1.5 Two Rings 119
7.2 Minimal 3D Combinatorial Maps 121
7.2.1 Simplex 122
7.2.2 Tunnel 122

7.2.3 Hole . . . 123
7.2.4 2 Rings . . . 124
7.3 Performance . . . 125
7.3.1 Size of Combinatorial Pyramids . . . 125
7.3.2 Bottom-up Construction Times . . . 128

8 Conclusion and Outlook **132**
8.1 Minimal 3D Combinatorial Maps . . . 132
8.2 Minimal Configurations and Pseudo Elements . . . 133
8.3 Minimal Combinatorial Maps for analyzing 3D Data . . . 134
8.4 Outlook . . . 134

9 Acknowledgments **136**

Listings

3.1 Calculation of the degree of a vertex . . . 28
3.2 Calculation of the degree of an edge . . . 29
3.3 Calculation of the dual-degree of a volume . . . 30

5.1 Algorithm step . . . 70
5.2 Simplification step using contraction operations . . . 76
5.3 Simplification step using removal operations . . . 78
5.4 Building the next level of a pyramid . . . 80
5.5 Retrieving the surviving dart using connecting walks . . . 82
5.6 Recursively calculating the successor for a removal operation . 83

6.1 Identifying the adjacency relations of an object . . . 101
6.2 Algorithm for detecting "surrounds" relations in a combinatorial map . . . 103

List of Figures

2.1 2D combinatorial maps using different notations. 12
2.2 Degree of an edge. 17
2.3 Examples of bounding relationship diagrams. 19
2.4 Representation of a 3x3 image using a combinatorial map. . . 20
2.5 Example of a graph pyramid built from a 3x3 image. 23

3.1 3D combinatorial map permutations 25

4.1 Contraction of an i-cell. 34
4.2 Contraction operation leading to invalid bounding and incidence relations. 36
4.3 Contraction violating the "common $(i-2)$-cells" condition . . 38
4.4 Uncontractible volume 39
4.5 Edge Contraction 41
4.6 Face Contraction 43
4.7 Volume Contraction 45
4.8 Simplification of a cube 48
4.9 Alternative simplification of a cube 50
4.10 Removal of an i-cell. 52
4.11 Unremovable Vertex 55
4.12 Vertex Removal 57
4.13 Edge Removal 58
4.14 Face Removal 61

5.1 Describing 3D data 64
5.2 A vertex within the initial 3D combinatorial map. 65
5.3 Examples of redundant i-cells 73
5.4 Examples of non-empty parallel i-cells 74
5.5 Connecting walk 84

6.1 The 3 minimal configurations of an edge. 93
6.2 The minimal representation of a face. 94
6.3 The minimal representation of a volume. 95

6.4 The minimal representation of a volume using 5 i-cells. 97
6.5 Examples of pseudo elements 99
6.6 The minimal representation of the "surrounds" relation. 102
6.7 First minimal representation of the "contains" relation. 104
6.8 Second minimal representation of the "contains" relation. 106
6.9 Invalid combinatorial map created by the removal of faces. . . 108

7.1 Reduction of a simplex configuration 112
7.2 Reduction of a tunnel configuration 114
7.3 Visualization of the tunnel configuration 115
7.4 Reduction of a hole configuration 117
7.5 Visualization of the hole configuration 118
7.6 Bounding relationship diagram for the final map of two rings . 119
7.7 Visualization of the final map of two rings 120
7.8 Size of combinatorial pyramids 126
7.9 Bottom-up construction times of combinatorial pyramids . . . 129

List of Tables

3.1 Permutations for the i-cells of a 3D combinatorial map. 26
3.2 Degree and dual-degree of i-cells in a 3D combinatorial map. 26

4.1 Number of i-cells after each simplification step. 49
4.2 Relation of the contraction and removal operations in the primal and dual graph. 50

5.1 Number of darts, vertices, edges, faces and volumes in the initial grid-like maps of different sizes. 68
5.2 Approximated number of darts, vertices, edges, faces and volumes in the initial grid-like maps of different sizes. 68
5.3 Operations used to eliminate different kinds of redundant i-cells in a 3D combinatorial map. 74
5.4 Permutation sequences for the construction of a connecting walk. 81

6.1 Number of cells and darts for the three minimal representations of an edge. . . . 91
6.2 Number of cells and darts for the minimal representation of a face. . . . 94
6.3 Number of cells and darts for the two minimal representations of a volume. . . . 96
6.4 Topological relations in a 3D combinatorial map. . . . 100

7.1 Reduction of a simplex configuration. . . . 112
7.2 Reduction of a tunnel configuration. . . . 113
7.3 Reduction of a hole configuration. . . . 116
7.4 Reduction of two rings. . . . 119
7.5 Minimal combinatorial map for the simplex configuration . . . 122
7.6 Minimal combinatorial map for the tunnel configuration . . . 123
7.7 Minimal combinatorial map for the hole configuration . . . 123
7.8 Minimal combinatorial map for the 2 rings configuration . . . 124
7.9 Memory allocated by pyramids of different sizes. . . . 125
7.10 Number of darts and cells for a $40 \times 40 \times 40$ combinatorial pyramid. . . . 128
7.11 Bottom-up times of combinatorial pyramids. . . . 130

List of Conditions

Condition 1 - Dual Degree . . . 35
Condition 2 - Bounding and Incidence Relations . . . 35
Condition 3 - Self-Loops . . . 37
Condition 4 - Common $(i-2)$-cells . . . 37
Condition 5 - Background vertex . . . 40
Condition 6 - Degree . . . 53
Condition 7 - Bounding and Incidence Relations . . . 53
Condition 8 - Bridges . . . 54
Condition 9 - Common $(i+2)$-cells . . . 54
Condition 10 - Connected Vertices . . . 56

List of Definitions

Definition 1 - Vertex Model of a 2D Combinatorial Map 13
Definition 2 - Face Model of a 2D Combinatorial Map 13
Definition 3 - Orbit 14
Definition 4 - i-Cell 15
Definition 5 - Incidence 16
Definition 6 - Degree 16
Definition 7 - Dual-Degree 17
Definition 8 - Removal Operation in a vertex model 2D combinatorial map 21
Definition 9 - Contraction Operation in a vertex model 2D combinatorial map 22
Definition 10 - 3-map 24
Definition 11 - Cycle of a Permutation 26
Definition 12 - equivalent combinatorial map 31
Definition 13 - Edge Contraction 32
Definition 14 - Face Contraction 33
Definition 15 - Volume Contraction 33
Definition 16 - Vertex Removal 51
Definition 17 - Edge Removal 51
Definition 18 - Face Removal 51
Definition 19 - Parallel i-Cell 72
Definition 20 - Empty i-Cell Self-Loop 72
Definition 21 - Redundant i-Cell 72
Definition 22 - Minimal 3D Combinatorial Map 86
Definition 23 - Minimal 3D Configuration 90
Definition 24 - Pseudo Element 98

Chapter 1

Introduction

This technical report presents the diploma thesis "Minimal Combinatorial Maps for analyzing 3D Data" [Illetschko, 2006].

Combinatorial pyramids are irregular pyramids where each level is represented by a combinatorial map, encoding a topological representation of the processed data. Within the pyramid each level represents a reduction of the level below, up to the top level which is the minimal topological equivalent representation of the initial data. As such, the combinatorial pyramid is a hierarchical stack of topological encodings that can be used to analyze an image and sequences of images. In the past, it has been shown that this concept can be advantageous in various fields of image processing e.g. the segmentation of images [Grasset-Simon et al., 2005, Brun and Kropatsch, 1999b].

Instead of analyzing images solely based on the values represented by the pixels of an image (like colors, contrast, shapes, etc.), it is often important to know about the structure within the data. For example, an application might need to distinguish between different parts of an object. These parts of an object resemble the structure of the object. Such an application can be helped by representing the relations of the regions within an image.

Another example would be an application that needs to distinguish between solid objects and objects that enclose other objects. In order to do this, topological relations like adjacency, "contains" and "surrounds" must be representable as well as the inside and outside of regions must be known [Brun and Kropatsch, 2005].

The structure or topology of an object can be understood as the decomposition of the object into its sub-objects and the description of their relations

to each other. E.g. one way of representing the topology of a person would be to describe a person as a set of six sub-objects and their relations - a head, a torso, two arms and two legs. The person is further defined by a strict relation between these sub-objects: the torso must be adjacent to the other sub-objects, the head, the two arms and the two legs. In the field of image analysis such sub-objects are equivalent with regions within the images.

A region can be defined as a (structured) set of pixels (2D) or voxels (3D). A more general definition would be that of an abstract cellular complex consisting of dimensions 0, 1, 2, 3 and a bounding relation. The topological relations can then be described by the adjacency of these cells and their bounding relations [Kovalevsky, 1989].

In the past, different models have been introduced to represent topological relations. These models have different properties and few are capable of handling data of any dimension. A comparison can be found in [Lienhardt, 1991].

Combinatorial maps encode any subdivision of nD topological spaces orientable or non-orientable with or without boundaries. As such they have been shown to be a suitable topological representation model for the field of 2D image analysis [Brun and Kropatsch, 2000, Haxhimusa et al., 2005a]. 2D segmentation algorithms [Felzenszwalb and Huttenlocher, 1998, Borůvka, 1926] have been successfully implemented and studied using combinatorial maps [Haxhimusa et al., 2005b].

These examples have shown that combinatorial maps are a useful framework for analyzing images. But even though combinatorial maps are defined for any dimension [Lienhardt, 1989] and [Damiand, 2001, Braquelaire et al., 2003] have presented suitable extensions to 3D, most studies so far have concentrated on 2D. One of the reasons for this is the fact that combinatorial maps are computational expensive and resource intensive compared to other methods.

For 2D combinatorial maps efficient operations have been defined to reduce the complexity of the initial maps [Kropatsch, 1995]. Using these operations irregular pyramids can be built, where each level is a reduced version of the level below [Brun and Kropatsch, 1999b]. The top level of such a combinatorial map will represent the minimal topological equivalent representation of the 2D data. Working with these minimal maps allows efficient processing of the data while more details can be retrieved from the levels below when needed [Brun and Kropatsch, 2003].

[Damiand, 2001] has presented a definition for 3D combinatorial maps as well as methods of creating minimal maps using removal operations.

The goal of this report is to study the properties of 3D combinatorial maps with a strong focus on minimal maps. For this purpose a combinatorial map will be considered minimal if each connected component is represented by a single vertex. Suitable contraction and removal operations are introduced to reduce initial combinatorial maps to their minimal form. It will be shown, that a minimal 3D combinatorial map can be obtained for any initial 3D combinatorial map that does not contain a face self-loop.

Further minimal configurations with respect to the number of cells and the operations introduced in this report are examined and the topological relations presented therein are analyzed. Finally, methods for identifying the topological relations and for distinguishing between them are presented.

An introduction to 2D combinatorial maps and combinatorial pyramids together with the basic notions needed in the context is presented in chapter 2. Chapter 3 recalls the extension of combinatorial maps to 3D defined by [Damiand, 2001, Braquelaire et al., 2003].

The contraction and removal operations needed to reduce a 3D combinatorial map to its minimal form are described in detail in chapter 4. Definitions of these operations are presented as well as 10 conditions when these operations may be applied. The 10 conditions ensure that the result of these operations is a valid combinatorial map and that the operations used to simplify the maps will preserve adjacency and inclusion information as well as the genus of the bounding surfaces.

Chapter 5 describes the process of building a 3D combinatorial pyramid using the contraction and removal operations, starting from an initial map representing the original data. Minimal maps, where each connected component is represented by a single vertex, are discussed in chapter 6 and it is shown that such minimal combinatorial maps can be obtained for any 3D combinatorial map that does not contain a face self-loop. Additionally minimal configurations representing topological relations are examined and methods identifying and discriminating between each of these topological relations are presented.

Chapter 7 describes experimental results. A C++ program for 3D combinatorial maps has been created for this report implementing the operations defined here. Representative configurations reduced using this C++ framework are shown as well as some observations regarding the performance and resource costs of 3D combinatorial pyramids built with this implementation.

Finally chapter 8 will present a discussion of the results and the conclusions of this report together with an outlook.

Chapter 2

Recall of Combinatorial Maps

Within this chapter the definition of a nD combinatorial map and definitions for 2D combinatorial maps are recalled. In addition, basic operations needed for building a combinatorial pyramid are defined as well as the concept of such pyramids in the example of 2D combinatorial maps. A complete and detailed definition of these concepts can by found in [Brun and Kropatsch, 1999b].

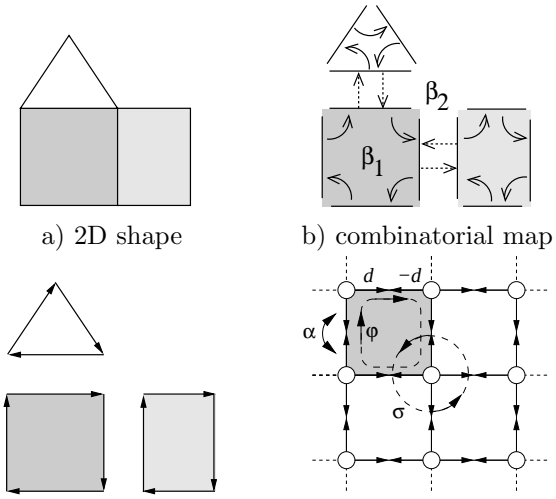
Combinatorial maps are a mathematical model of representation of space subdivision in any dimension. They were introduced by [Edmonds, 1960], at first as a planar graph representation model, and extended by [Lienhardt, 1989] in dimension n to represent orientable or not-orientable quasi-manifolds. Combinatorial maps encode space subdivisions and all incidence relations [Damiand and Resch, 2002, Bertrand et al., 2000].

In dimension n a combinatorial map is a $(n + 1)$-tuple $M = (D, \beta_1, \beta_2, \ldots, \beta_n)$ such that D is the set of abstract elements called darts, β_1 is a permutation on D and the other β_i are involutions on D. An involution is a permutation whose cycle has the length of two or less.

For combinatorial maps of dimension n there is more than one way of attributing the permutations β_i. When encoding the same configuration using two different ways of attributing the permutations, the differences between these two encodings are limited to the number of darts, the number of permutations, and their meaning.

2.1 2D Combinatorial Maps

Using the above definition, a 2D combinatorial map is defined by the triplet $M = (D, \beta_1, \beta_2)$. Two different sets of permutations are often found in literature. Fig. 2.1 shows the encoding of simple 2D objects using these different sets.



a) 2D shape b) combinatorial map

c) simplified representation d) combinatorial map

Figure 2.1: 2D combinatorial maps using different notations.

[Brun and Kropatsch, 1999a] define the combinatorial map by the triplet $G = (D, \sigma, \alpha)$ where D represents the set of darts, σ is a permutation in D connecting all darts encountered when turning clockwise around a vertex and α is an involution connecting 2 darts belonging to the same edge (see Fig. 2.1d).

Notation 2.1 *Let D be a set of darts, and let β_i be a permutation on D. Applying β_i n times on a dart $d \in D$ will be denoted $\beta_i^n(d)$.*

$$\beta_i^n(d) = \underbrace{\beta_i(\beta_i(\ldots\beta_i(}_{n-times} d)\ldots))$$

Definition 1 (Vertex Model of a 2D Combinatorial Map) *The vertex model is a combinatorial map G given by the triplet $G = (D, \sigma, \alpha)$, where D is a set called the set of darts and σ, α are two permutations defined on D such that α is an involution:*

$$\forall d \in D \quad \alpha^2(d) = d$$

As such the vertex model of a 2D combinatorial map can be understood as a particular encoding of a planar graph, where each edge is split into 2 half-edges represented by the darts and connected by the α involution. Additionally the orientation of edges around a vertex is encoded by the permutation σ.

A similar definition is used by [Damiand, 2001] that will be called the "face model" in this report.

Definition 2 (Face Model of a 2D Combinatorial Map) *The face model is a combinatorial map G given by the triplet $G = (D, \beta_1, \beta_2))$ where D represents the set of darts, β_1 is a permutation of the darts belonging to the same face and β_2 is an involution connecting two darts belonging to the same edge:*

$$\forall d \in D \quad \beta_2^2(d) = d$$

Fig. 2.1 shows examples of combinatorial maps using the "vertex model" (Fig. 2.1d) and the "face model" (Fig. 2.1b).

Proposition 2.1 *Definition 1 (vertex model) and definition 2 (face model) are equivalent definitions of a 2D combinatorial map. Both definitions can be derived from each other by using $\beta_1 = \sigma \circ \alpha$ and $\beta_2 = \alpha$. Using these two equations, it follows that $\sigma = \beta_1 \circ \beta_2$.*

Proof: Starting with $\beta_1 \circ \beta_2 = \beta_1 \circ \beta_2$ it can be shown that $\sigma = \beta_1 \circ \beta_2$ using the two substitutions from proposition 2.1:

$$\beta_1 \circ \beta_2 = \beta_1 \circ \alpha = \sigma \circ \alpha^2 = \sigma$$

□

Definition 2 (face model) will be used in this report when extending combinatorial maps to 3D.

2.2 Dual Combinatorial Maps

Each combinatorial map has a corresponding dual combinatorial map that can be obtained directly from the original map. E.g. given a combinatorial map $G = (D, \sigma, \alpha)$, the combinatorial map $\bar{G} = (D, \phi, \alpha)$ is called the dual of G. The permutation ϕ is defined by $\phi = \sigma \circ \alpha$ [Brun and Kropatsch, 1999a].

Intuitively a combinatorial map and its dual combinatorial map are symmetrical representations of the same configuration. Given a nD combinatorial map each i-cell corresponds to exactly one $(n-i)$-cell in the dual. (e.g. For 2D each vertex corresponds to a face in the dual).

In the same way incidence relations correspond to bounding relations in the dual. (e.g. the edges being incident to a vertex in the primal correspond to the edges bounding the face in the dual).

This relation also applies to the removal and contraction operations introduced later in this report. A contraction in the primal graph corresponds to a removal in the dual and vice versa.

2.3 i-cells, Degree, Dual-degree

A nD combinatorial map implicitly encodes the subdivision of a nD space into cells of dimension $0, \ldots, n$. In the case of 2D, combinatorial maps encode a subdivision of a 2D space into 0-cells (vertices), 1-cells (edges) and 2-cells (faces).

[Damiand, 2001] defines the i-cell for a nD combinatorial map using the notion of the orbit of a dart.

> **Definition 3 (Orbit)** *Let $\Phi = \{f_1, \ldots, f_k\}$ be some permutations on D. We note $< \Phi >$ the permutation group generated by Φ. This is the set of permutations obtained by any composition and inversion of permutations contained in Φ. The orbit of a dart d relatively to Φ is defined by $< \Phi > (d) = \{\phi(d) | \phi \in \Phi\}$.*

Definition 4 (i-Cell) *Let C be a nD combinatorial map $G = (D, \beta_1, \ldots, \beta_n)$, let d be a dart of D, and let $0 \leq i \leq n$. The i-cell incident to d is defined by*

- $< \beta_2 \circ \beta_1, \ldots, \beta_n \circ \beta_1 > (d)$ *if* $i = 0$
- $< \beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_n > (d)$ *if* $i > 0$

Notation 2.2 *Let G be nD combinatorial map and let c be an i-cell of G. $D(c)$ denotes the set of darts belonging to c.*

The darts belonging to a specific i-cell can be obtained by starting from any dart belonging to the i-cell using $(n-1)$ permutations β_i [Damiand and Lienhardt, 2003]. In case of 2D exactly one permutation is used: given a combinatorial map $G = (D, \sigma, \alpha)$ the darts belonging to a 0-cell (vertex) are obtained using σ, the darts belonging to a 1-cell (edge) are obtained using α and the darts belonging to a 2-cell (face) are obtained using ϕ.

Proposition 2.2 *Given a nD combinatorial map $G = (D, \beta_1, \ldots, \beta_n)$. Each dart $d \in D$ belongs to exactly one i-cell of each dimension ($0 \leq i \leq n$).*

Proof: Given a nD combinatorial map $G = (D, \beta_1, \ldots, \beta_n)$, a dart $d \in D$. The i-cells that d is incident to are obtained by using d as the starting dart. Therefore the set of darts obtained for the i-cells of dimension $0 \leq i \leq n$ must contain at least d itself. Thus d is incident to an i-cell for each dimension $0 \leq i \leq n$.

Given an i-cell $c_1 \in G$ that d is incident to. There cannot be another i-cell $c_2 \in G$, $c_1 \neq c_2$ that d is also incident to. An i-cell is defined as the set of darts obtained by the orbit of $n-1$ permutations. Due to the definition of an orbit, any dart incident to a given i-cell can be used as the starting dart to obtain the complete set of darts belonging to that i-cell. This means that if d is incident to c_1 and c_2, both cells can also be obtained by using d as the starting dart. Thus the same set of darts is obtained and $c_1 = c_2$.

Therefore d is incident to exactly one cell of each dimension $0 \leq i \leq n$. □

Definition 5 (Incidence) *Given a combinatorial map G, let c be an i-cell of G. An i-cell $c' \in G$ is called incident to c iff*

$$D(c) \cap D(c') \neq \emptyset.$$

Proposition 2.3 *Given an nD combinatorial map G.*

$\forall$ i-cell $c_i \in G, 0 \leq j \leq 1, j \neq i; \exists$ i-cell $c_j \in G$ such that c_j is incident to c_i

Proof: The set D_c of darts belonging to an i-cell c with dimension i is non-empty. Thus there is at least one dart $d \in D_c$. Since each dart belongs to exactly one cell of each dimension there also must be one cell c' of each dimension with $D_{c'}$ being the set of darts belonging to c', where $d \in D'_c \wedge d \in D_c$ which is the definition of incidence (definition 5). $\square$

The number of times that a given i-cell is incident to $(i+1)$-cells is called the degree of the i-cell. This corresponds to the definition of a local cell degree given by [Damiand, 2001]:

Definition 6 (Degree) *The degree of an i-cell c is the sum of the number of times that c' is incident to c for each $(i+1)$-cell c' incident to c.*

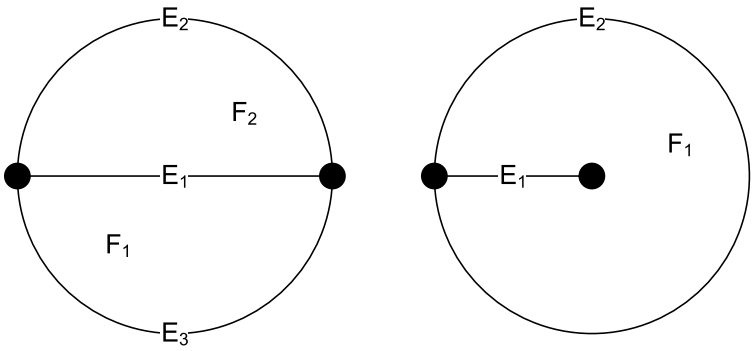
Notation 2.3 *Let G be a combinatorial map $G = (D, \beta_1, \beta_2, \ldots, \beta_n)$, let $d \in D$ be a dart of G and let c be an i-cell of G. $0 \leq i \leq n$.*

$\delta(d, i)$ will then denote the degree of the i-cell the dart d belongs to.

$\delta(c)$ will denote the degree of the cell c.

Note that definition 6 does not specify how to calculate $\delta(c_i)$ for a given nD combinatorial map $G = (D, \beta_1, \ldots, \beta_n)$ and i-cell c_i as the calculation of $\delta(c_i)$ depends on the attribution used for the permutations $\beta_1, \ldots, \beta_n$ as well as on the permutation sequences used for the c_i. Instead algorithms for the calculation of δ for i-cells of dimension $0, \ldots, 3$ will be provided in section 3.2 for the specific 3D combinatorial maps used in this report.

Fig. 2.2 gives an example of how the degree is determined for an edge. The degree for E_1 is two in both cases. In the example on the left side the degree is two because the edge is incident to two different faces (F_1 and F_2).



a) two faces separated by an edge b) one face with a pendant edge

Figure 2.2: Degree of an edge. In both cases the degree of E_1 is two. In the first example (a) the edge E_1 is incident to two different faces (F_1 and F_2). In the second example (b) E_1 is incident to only one face F_1, but the degree is still two, since it is incident to that face twice.

In the example on the right side the edge is incident to only one face (F_1) but the degree is still two since it is incident to this face twice (F_1 touches E_1 on both sides of the edge).

Using $G = (D, \sigma, \alpha)$ the degree of a 0-cell is equal to the length of the cycle of the σ permutation and the degree of a 1-cell is equal to the length of the cycle of the involution α.

The number of times that a given i-cell is bounded by $(i-1)$-cells is called the dual-degree of a cell. The dual-degree is defined as the degree of the dual i-cell:

Definition 7 (Dual-Degree) *The dual-degree of an i-cell c is the sum of the number of times that c' is incident to c for each $(i-1)$-cell c' bounding c.*

Notation 2.4 *Let G be a combinatorial map $G = (D, \beta_1, \beta_2, \ldots, \beta_n)$, let $d \in D$ be a dart of G and let c be an i-cell of G. $0 \leq i \leq n$.*

$\bar{\delta}(d, i)$ will then denote the dual-degree of the i-cell the dart d belongs to.

$\bar{\delta}(c)$ will denote the dual-degree of the cell c.

As for the definition of the degree (definition 6), definition 7 does not specify how to calculate $\bar{\delta}(c_i)$ for a given nD combinatorial map $G =$

$(D, \beta_1, \ldots, \beta_n)$ and i-cell c_i, but algorithms for the calculation of $\bar{\delta}$ for i-cells of dimension $0, \ldots, 3$ will be provided in section 3.2 for the specific 3D combinatorial maps used in this report.

Fig. 2.2 can be reused for an example of the dual-degree. On the left side the dual-degree of F_1 is two, as it is bounded by two edges (E_1 and E_3). On the right side the dual-degree of F_1 is three, as it is bounded by E_2 once and by E_1 twice.

Using $G = (D, \sigma, \alpha)$ the dual-degree of a 2-cell (the number of edges that bound a face) is equal to the length of the cycle of the ϕ permutation and the dual-degree of a 1-cell is equal to the length of the involution α (An edge is always bounded twice by vertices).

2.4 Bounding Relationship Diagram

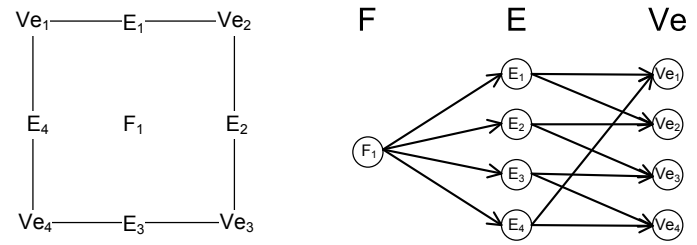
Following proposition 2.3, i-cells have strict bounding relations. Each i-cell with $i > 0$ is bounded by at least one $(i-1)$-cell. E.g. a face is bounded by edges which are themselves bounded by vertices.

These bounding relations can be visualized using a bounding relationship diagram. For a nD combinatorial map the diagram is divided into $n+1$ columns, where each column contains all i-cells for a given i $(0 \leq i \leq n)$ present in the map. The columns are ordered starting with the highest i on the left. E.g. a bounding relationship diagram for a 2D combinatorial map consists of three columns. The first one displays all faces (2-cells) of the map, the second one contains all edges (1-cells), and all vertices (0-cells) are presented in the rightmost column.

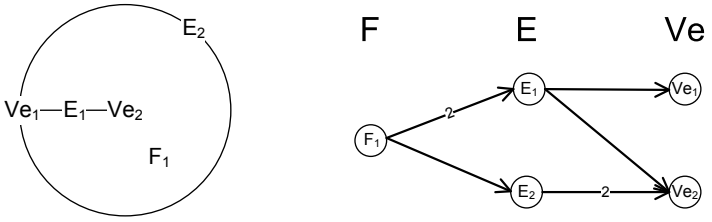
Each i-cell is connected with arrows to all $(i-1)$-cells bounding it, thus visualizing the bounding relation of this i-cell. A number w on the arrow indicates that the i-cell is bounded w times by this particular $(i-1)$-cell.

Fig. 2.3a shows the bounding relationship diagram for a square consisting of one face (F_1), 4 edges (E_1 to E_4) and 4 vertices (Ve_1 to Ve_4). The face is bounded by all four edges described by 4 arrows from F_1 to each 1-cell. The edges E_1 to E_4 are all bounded by two of the four vertices shown by two arrows from each edge to the bounding vertices in the rightmost column.

Fig. 2.3b shows another example for a bounding relationship diagram. A face (F_1) that is bounded by an edge self-loop (E_2) and a pendant edge (E_1). F_1 is therefore bounded by E_1 twice which is indicated by the 2 on the arrow



a) square



b) face bounded by an edge self-loop and a pendant edge

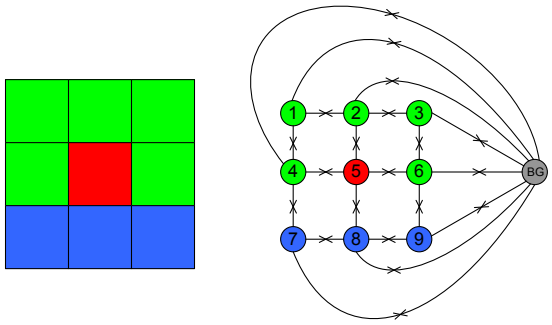
Figure 2.3: Examples of bounding relationship diagrams. The square (a) consists of 7 cells: one face (2-cell), four edges (1-cells) and four vertices (0-cells). Their bounding relations are presented in the bounding relationship diagram by arrows from the bounded cell to the bounding cells. The second example shows a face that is bounded by an edge self-loop and a pendant edge (b). The fact that the face is bounded twice by the pendant edge and that the edge self-loop is bounded twice by the same vertex is indicated by the 2 on the arrows.

from F_1 to E_1 in the bounding relationship diagram. In the same way the fact that the edge self-loop (E_2) is bounded by the same vertex (Ve_1) twice is indicated again by a 2 on the arrow from E_2 to Ve_1.

The bounding relationship diagram will be used throughout this report using the following abbreviations for the heading line: "Vo" - 3-cells (volumes), "F" - 2-cells (faces), "E" - 1-cells (edges) and "Ve" - 0-cells (vertices).

2.5 Initial Map

The first step when working with 2D combinatorial maps is building an initial map that represents the data. This is done by building a grid-like map, when



a) 3×3 image b) combinatorial map

Figure 2.4: Representation of a 3x3 image using a combinatorial map. Each pixel in the image (a) is represented by one vertex in the combinatorial map (b). The adjacency of pixels is represented by edges connecting these vertices. An additional background-vertex is added and connected to each pixel at the border of the image.

working with typical data sources like labeled images.

In that context, labeling (or filing or region detection) is one of the first tasks that arises frequently in picture analysis. It is the process of assigning the same labels to pixels that belong to the same component, such that two pixels that do not belong to the same component are assigned different labels (see [Klette and Rosenfeld, 2004] for a more complete and formal description of this process).

Within a grid-like map the pixels of the labeled image can either be represented by the 0-cells (vertices) of the map or by the 2-cells (faces) of the map. These cells are assigned the label of the pixel they represent. The adjacency relations of the 4 neighboring pixels are then either represented by edges connecting the vertices corresponding to these pixels or by edges separating the corresponding faces. (Fig. 2.4 shows such a representation for a 3x3 image using vertices to represent the pixels).

When using vertices to represent pixels, an additional vertex is added to the map. This vertex represents the background or outside of the image that each pixel on the border is adjacent to. Throughout this report the grid using vertices to represent pixels is used.

2.6 Contraction and Removal Operations

Two operations were defined by [Brun and Kropatsch, 1999a, Grasset-Simon et al., 2005] to simplify combinatorial maps: the contraction operation and the removal operation. In addition, conditions were presented that define when these operations may be applied without changing the topological relations encoded by the combinatorial map i.e. the original adjacency and inclusion relations are preserved.

The removal operation can be applied to an i-cell with $i < n$ (where n is the dimension of the combinatorial map). It is used to remove redundant i-cells like parallel edges or empty self-loops that contain no topological information from the map. [Brun and Kropatsch, 1999a] give the following definition of a removal operation for a 2D combinatorial map:

Lemma 2.1 ***The Restriction Operator***

Given a combinatorial map $G = (D, \sigma, \alpha)$ and $D' \subset D$ the application:

$$p_{D,D'} \left(\begin{array}{l} D' \rightarrow D \\ d \rightarrow \sigma^{n-1}(d) \quad with \quad n = \min\{p \in \mathbb{N}^\star | \sigma^p(d) \in D'\} \end{array} \right.$$

is an injective function.

Definition 8 (Removal Operation in a vertex model 2D combinatorial map)

Given a combinatorial map $G = (D, \sigma, \alpha)$ and $d \in D$. If $\alpha^\star(d)$ is not a bridge, the combinatorial map $G' = G \setminus \alpha^\star(d)$ is the submap defined by:

- *$D' = D - \alpha^\star(d)$ and*
- *$\sigma' = \sigma \circ p_{D,D'}$.*

This operation will be denoted R_d.

Notation 2.5 *Given a nD combinatorial map $G = (D, \beta_1, \ldots, \beta_n)$ and an i-cell $c \in G$ with D_c being the darts of c. The removal of c will be denoted $G \setminus D_c$.*

The contraction operation can be applied to an i-cell with $i > 0$ bounded by two $(i-1)$-cells to merge it to a single $(i-1)$-cell. E.g. a face bounded

by 2 edges can be contracted to an edge or an edge can be contracted to a vertex. The edge contraction is used to merge two vertices representing pixels that belong to the same component [Brun and Kropatsch, 1999a, Damiand and Lienhardt, 2003]. [Brun and Kropatsch, 1999a] give the following definition of a contraction operation for a 2D combinatorial map (in this definition $\overline{G}$ is used to denote the dual combinatorial map):

> **Definition 9 (Contraction Operation in a vertex model 2D combinatorial map)**
>
> *Given a combinatorial map $G = (D, \sigma, \alpha)$ and one dart $d \in D$ which is not a self loop. The contraction of dart d creates the graph:*
>
> $$G' = G/\alpha^\star(d) = \overline{\overline{G} \setminus \alpha^\star(d)}$$
>
> *This operation will be denoted C_d.*
>
> *Note that this operation is well defined since d is a self loop in G iff it is a bridge in $\overline{G}$.*

Notation 2.6 *Given a nD combinatorial map $G = (D, \beta_1, \ldots, \beta_n)$ and an i-cell $c \in G$ with D_c being the darts of c. The contraction of c will be denoted G/D_c.*

Definition 8 and definition 9 are specific definitions for the removal and contraction operation based on the the vertex model of a 2D combinatorial map (definition 1).

Chapter 4 will provide definitions for the six specific contraction and removal operations for 3D combinatorial maps used in this report.

2.7 2D Combinatorial Pyramids

A 2D Combinatorial pyramid is a hierarchical stack of 2D combinatorial maps. Such irregular pyramids can be obtained by removal and contraction operations. Each new level - starting from the grid-like map representing the original data - is built using edge contractions to merge two adjacent vertices belonging to the same component. Face contraction and edge removal are then used to simplify the map. This procedure is repeated until no more simplification is possible and the minimal map is reached. A detailed description of this process can be found in [Kropatsch, 1995, Brun and Kropatsch, 1999b].

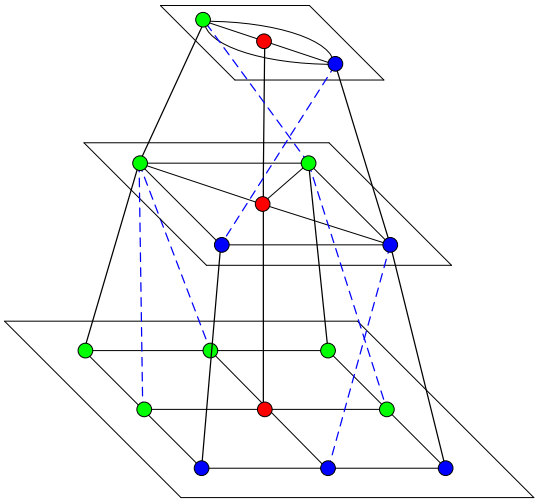


Figure 2.5: Example of a graph pyramid built from a 3x3 image. Vertices of the same color are merged at each level. Vertices that are eliminated by such an operation are connected to the corresponding surviving dart at the higher level. (This is indicated by the blue, dotted line).

Using the concept of surviving darts and connecting walks [Brun and Kropatsch, 1999b, Brun and Kropatsch, 2003] details can be retrieved from levels below within such a pyramid. Each non-surviving dart that is eliminated at a certain level by a contraction or a removal operation is assigned to exactly one surviving dart. This way each surviving dart is associated with a set of non-surviving darts at the level below. This information can then be used to obtain the receptive field for each dart of a higher level. Fig. 2.5 shows an example of a graph pyramid. Each non-surviving vertex is connected to one surviving vertex at the next level.

Combinatorial pyramids are a powerful tool when analyzing data. The top-level of the pyramid allows us to work with a minimal representation of the underlying data while maintaining topological relations within the data. Using connecting walks to obtain the receptive fields of any dart in the top level gives us the possibility to retrieve more details from a lower level when needed.

In the next chapters of this report the concepts of combinatorial maps, combinatorial pyramids, and contraction and removal operations are extended for representing and minimizing 3D data.

Chapter 3

3D Combinatorial Maps

This chapter describes the extension of combinatorial maps for representing volumetric data that is used within this report. The set of permutations, the definitions of i-cells as well as the degree and dual degree of such i-cells within these 3D combinatorial maps are explained.

This report uses the definition of 3D combinatorial maps given by [Braquelaire et al., 2003, Damiand, 2001]:

> **Definition 10 (3-map)** *A 3-map, is an 4-tuple* $M = (D, \delta_1, \delta_2, \delta_3)$ *where:*
>
> 1. *D is a finite set of darts;*
> 2. *δ_1 is a permutation on D;*
> 3. *δ_2, δ_3 and $\delta_1\delta_3$ are involutions on D.*

This definition is compliant to the general definition of a combinatorial map given in chapter 2. [Braquelaire et al., 2003] propose two equivalent models for the attribution of the permutation δ_1 and the involutions δ_2 and δ_3: the *topological map with global embedding (GE Model)* and the *topological map with hierarchical local embedding (HLE model)*.

This report uses the HLE model which is a 3-map defined by $M_{HLE} = (D, \beta_1, \beta_2, \beta_3)$. This model is an extension of definition 2 (Face Model of a 2D Combinatorial Map).

β_1 is a permutation that links darts that belong to consecutive edges of a face. Thus it encodes the orientation of these darts around the face. In 3D

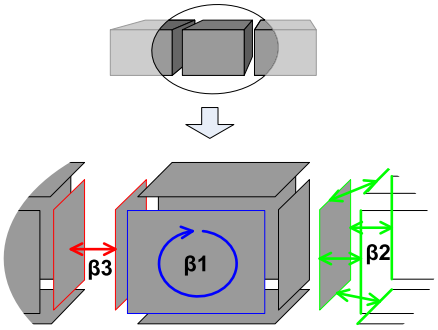


Figure 3.1: 3D combinatorial map permutations

each face has two sides. The set of darts that one side of a face consists of is disjoint from the set of darts that the other side of this face consists of. Both sides have the opposite orientation encoded by β_1.

β_2 is an involution that connects two darts belonging to the same edge and to the same volume.

β_3 is an involution that connects two darts that belong to the same edge and to the same face.

Intuitively β_1 is used to build faces, β_2 builds volumes by connecting these faces to create the border of the volume, and β_3 connects these volumes by linking two sides of a face together. Fig. 3.1 shows an example of how β_1, β_2 and β_3 are used to represent volumetric data.

3.1 i-cells

In volumetric data, four types of topological cells are present: 0-cells (vertices), 1-cells (edges), 2-cells (faces) and 3-cells (volumes). According to section 2.3, two permutations are needed to obtain the darts belonging to a specific i-cell in a 3D combinatorial map: given a 3D combinatorial map $M = (D, \beta_1, \beta_2, \beta_3)$ two permutations δ_1 and δ_2 are selected so that the 2D combinatorial maps $M' = (D', \delta_1, \delta_2)$ are a partition on D. $\delta_1, \delta_2 \in \{\beta_1, \beta_2, \beta_3\}, \quad \delta_1 \neq \delta_2$.

Table 3.1 lists the permutations for the i-cells of a 3D combinatorial map given by [Braquelaire et al., 2003].

Vertices	Edges	Faces	Volumes
$(D', \beta_2 \circ \beta_1, \beta_3 \circ \beta_1)$	(D', β_2, β_3)	(D', β_1, β_3)	(D', β_1, β_2)

Table 3.1: Permutations for the i-cells of a 3D combinatorial map.

3.2 Degree and Dual-Degree

As has been introduced in section 2.3, the degree $\delta()$ can be used to calculate the number of $(i+1)$-cells being incident to a given i-cell and the dual-degree $\bar{\delta}()$ can be used to calculate the number of $(i-1)$-cells the i-cell is being bounded by. In 2D both, the degree and the dual-degree, are determined by the length of the cycle of a certain permutation. The length of the cycle of a permutation is defined by the number of darts belonging to the cycle of the permutation.

Definition 11 (Cycle of a Permutation) *Let G be a combinatorial map $G = (D, \beta_1, \ldots, \beta_n)$. The cycle of β_i $(1 \leq i \leq n)$ for a given dart $d \in D$ is the maximum set of darts $\beta_i^\star(d)$ that can be obtained by starting with d and applying β_i any number of times.*

The number of darts in $\beta_i^\star(d)$ is called the length of the cycle of the permutation β_i for the dart d.

Note, that the definition of a cycle $\beta_i^\star(d)$ corresponds to the definition of an orbit $< \Phi > (d)$ if β_i is the only permutation in Φ.

The calculation in 3D is more complicated since the i-cells are defined as 2-maps using two permutations. $(i+1)$-cells being incident to a given i-cell can consist of more than one dart also belonging to this i-cell. In the same way $(i-1)$-cells bounding a given i-cell may be defined by more than one dart also being part of the i-cell. This section explains how to calculate the degree and the dual-degree for each kind of i-cell being present in a 3D combinatorial map.

Table 3.2 gives an overview of the bounding and incidence relations in a 3D combinatorial map. It lists the possible values for the degree δ and dual-degree $\bar{\delta}$ for i-cells of each dimension.

i-cell	Vertex	Edge	Face	Volume
degree	$1 \leq \delta$	$1 \leq \delta$	$\delta = 2$	not defined
dual-degree	not defined	$\bar{\delta} = 2$	$1 \leq \bar{\delta}$	$1 \leq \bar{\delta}$

Table 3.2: Degree and dual-degree of i-cells in a 3D combinatorial map.

3.2.1 0-cells (Vertices)

The degree of a 0-cell (vertex) is the number of times edges are incident to a given vertex. The vertex is defined by the permutations $\beta_2\beta_1$, $\beta_3\beta_1$ and edges are defined by the permutations β_2 and β_3.

Since both β_2 and β_3 are involutions, all darts belonging to one edge can be retrieved by consecutively applying both permutations in alternating order until the first dart is reached again.

The degree can therefore be calculated by examining all darts belonging to the vertex and using a flag to indicate which darts have been counted. If an unmarked dart is encountered the degree is increased by one and all darts belonging to the same edge are marked. If a marked dart is encountered the degree will not be increased. Listing 3.1 contains a C implementation of this algorithm.

By definition β_2 and β_3 will return a dart that belongs to the same edge but the opposite vertex. This can be used to refine the algorithm. When marking all darts that belong to the same edge, β_2 and β_3 are applied in an alternating way and only every second dart is marked. This ensures that only those darts are marked that also belong to the same vertex the degree is calculated for. This makes no difference in most cases since marking the darts of the opposite vertex has no impact on the algorithm, but in the case of self-loops both ends are bounded by the same vertex. In these cases the edge is two times incident to the vertex and must therefore be counted twice.

A vertex is not bounded by other i-cells. Therefore the dual-degree is not defined for vertices.

3.2.2 1-cells (Edges)

The degree of an edge is equivalent to the number of faces being incident to this edge.

The darts belonging to an edge are retrieved using the permutations β_2 and β_3. Both permutations are involutions so it is sufficient to apply these two permutations consecutively and in an alternating way on any dart of the edge to obtain the complete set of all darts of this edge.

β_3 will return a dart associated with the same face and edge. To calculate the degree of the edge it is therefore sufficient to iterate through all darts of

```
// set the degree to 0 and initialize the flag
// for all darts belonging to the vertex
degree = 0;
reset_marks_of_vertex(label_of_vertex);

// get one dart of the vertex
dart = get_dart_of_vertex(label_of_vertex);

// iterate through all darts belonging to the vertex
do {

  // if an unmarked dart is encountered
  if (dart_not_marked(dart)) {

    // increase the degree
    degree++;

    // and mark all darts belonging to the
    // same edge and vertex
    tmp_dart = dart;
    do {
      mark_dart(tmp_dart);
      tmp_dart = beta_2(tmp_dart);
      tmp_dart = beta_3(tmp_dart);
    } while (tmp_dart != dart);
  }

  // get the next dart of this vertex
  // NULL is returned if this was the last dart
  dart = get_next_dart();
} while (dart != NULL);
```

Listing 3.1: Calculation of the degree of a vertex

the edge counting the number of β_3-connected darts. Such an algorithm is shown in listing 3.2.

An edge is always bounded exactly two times by vertices. Therefore the dual-degree is always two for a 1-cell. In the case of an edge self-loop the edge is bounded twice by the same vertex, thus the dual-degree is also 2.

3.2.3 2-cells (Faces)

The degree of a face corresponds to the number of times this face is incident to volumes. In 3D a face is always incident to exactly two times to volumes.

```
degree = 0;

// get one dart of the edge
dart = get_dart_of_edge(label_of_edge);
tmp_dart = dart;

// iterate through all darts belonging to the edge
do {

  // increase the degree
  degree++;

  // apply beta_3 and beta_2.
  // the dart returned by beta_3 is skipped
  tmp_dart = beta_3(tmp_dart);
  tmp_dart = beta_2(tmp_dart);

// stop the iteration if the first dart is reached again
} while (tmp_dart != dart);
```

Listing 3.2: Calculation of the degree of an edge

The degree is therefore two for all faces present in a 3D combinatorial map.

The dual-degree of a face is the number of times a face is bounded by edges.

Since β_1 encodes the edges around the border of a face, the dual-degree is equivalent to the length of the cycle of the β_1 permutation.

The dual-degree of a face can be shown using the examples in Fig. 2.2. Face F_1 in Fig. 2.2a has a dual degree of two, as it is bounded by two edges (E_1 and E_3) once. In the second example (Fig. 2.2b) the dual-degree of F_1 is three. While F_1 is still only bounded by two different edges (E_1 and E_2), it is bounded by E_2 once and by E_1 twice. Therefore the dual-degree is three.

3.2.4 3-cells (Volumes)

Volumes are the highest dimensional i-cells present in a 3D combinatorial map. Thus, no cells of a higher dimension can be incident to a 3-cell and the degree is undefined.

The dual-degree of a 3-cell is defined by the number of times a volume is bounded by faces.

The darts of a volume are obtained by applying the permutations β_1 and β_2 on any dart of the volume in any possible order. β_1 returns the darts belonging to the same face while β_2 returns a dart belonging to the same edge but a different face. This can be used to build an algorithm to calculate the dual-degree that is similar to the algorithm for determining the degree of a vertex: all darts belonging to the same volume are iterated through. If an unmarked dart is encountered, the dual-degree is increased and all darts belonging to the same face are marked. If a marked dart is encountered, the dual-degree is not increased.

Listing 3.3 shows an implementation of such an algorithm.

```
// set the dual-degree to 0 and initialize the flag
// for all darts belonging to the volume
dual_degree = 0;
reset_marks_of_volume(label_of_volume);

// get one dart of the volume
dart = get_dart_of_volume(label_of_volume);

// iterate through all darts belonging to the volume
do {
  // if an unmarked dart is encountered
  if (dart_not_marked(dart)) {
    // increase the dual-degree
    dual_degree++;

    // and mark all darts belonging to the same face
    tmp_dart = dart;
    do {
      mark_dart(tmp_dart);
      tmp_dart = beta_1(tmp_dart);
    } while (tmp_dart != dart);
  }

  // get the next dart of this volume
  // NULL is returned if this was the last dart
  dart = get_next_dart();
} while (dart != NULL);
```

Listing 3.3: Calculation of the dual-degree of a volume

Chapter 4

Contraction and Removal in 3D

In order to simplify a 3D combinatorial map, the two operations introduced in section 2.6, contraction and removal, are extended to 3D. This way six specific operations are introduced: edge contraction, face contraction, volume contraction, vertex removal, edge removal, and face removal. These operations are explained in detail in this chapter.

Additionally, 10 conditions will be defined for these operations. If these conditions are fulfilled, the application of the removal or contraction operation will produce a valid combinatorial map.

One of the goals when simplifying a combinatorial map is to reduce the number of darts and i-cells in the combinatorial map while creating a topologically equivalent map i.e. the resulting map shall encode the same information about adjacencies and inclusions. The conditions presented in this report will ensure that the simplification of a 3D combinatorial map will preserve adjacency and inclusion information and will not change the genus of the bounding surfaces of the encoded objects.

Definition 12 (equivalent combinatorial map) *A combinatorial map G' obtained from a given combinatorial map G is equivalent, if the connected components, the connected component adjacency relations and inclusion relations encoded by G are preserved in G'.*

It will be shown for the operations used to simplify the combinatorial maps (face contraction, volume contraction, edge removal and face removal) that the genus of the bounding surfaces will not change if all conditions are fulfilled. This will be done using the Euler Characteristic. The Euler Characteristic $\chi = v - e + f$ determines the genus of a surface based on the number of vertices v, number of edges e and number of faces f.

Within this report all three contraction operations and two removal operations will be used to merge vertices within a map that belong to the same component and to simplify the resulting maps to their minimal form while maintaining the topological relations. Vertex removal is not used, since in this report vertices are used to represent voxels. The removal of a vertex would therefore remove a voxel from the combinatorial map.

4.1 Contraction Operation

Formally a contraction operation is defined as the removal of the i-cell and the merging of two $(i-1)$-cells - effectively removing one of these $(i-1)$-cells. E.g. when contracting an edge, the two bounding vertices of this edge are merged into a single vertex, removing one of the vertices. The connecting edge is also removed in the process.

Because a contraction operation merges two $(i-1)$-cells, only cells of dimsion $i \geq 1$ may be contracted. Therefore there are 3 contraction operation for a 3D combinatorial map: edge contraction, face contraction, and volume contraction. For each of these operations a definition is provided.

Definition 13 (Edge Contraction) *Let $G = (D, \beta_1, \beta_2, \beta_3)$ be a 3D combinatorial map and let $G' = (D', \beta'_1, \beta'_2, \beta'_3)$ be the resulting map after the contraction of a selected 1-cell $E = (D_e, \beta_2, \beta_3)$, $D_e \subset D$. $G' = G/D_e$ is then defined by*

1. $D' = D \setminus D_e$
2. $\forall d \in D'$,
$$\beta'_1(d) = \begin{cases} \beta_1(d) & : \quad \beta_1(d) \notin D_e \\ \beta_1^n(d) \text{ with } n = \min\{p \in \mathbb{N}^\star | \beta_1^p(d) \notin D_e\} & : \quad \beta_1(d) \in D_e \end{cases}$$
3. $\forall d \in D', \quad \beta'_2(d) = \beta_2(d)$
4. $\forall d \in D', \quad \beta'_3(d) = \beta_3(d)$

Definition 14 (Face Contraction) *Let $G = (D, \beta_1, \beta_2, \beta_3)$ be a 3D combinatorial map and let $G' = (D', \beta'_1, \beta'_2, \beta'_3)$ be the resulting map after the contraction of a selected 2-cell $F = (D_f, \beta_1, \beta_3)$. $G' = G/D_f$ is defined by*

1. $D' = D \setminus D_f$
2. $\forall d \in D', \quad \beta'_1(d) = \beta_1(d)$
3. $\forall d \in D', \quad \beta'_2(d) = \begin{cases} \beta_2(d) & : \quad \beta_2(d) \notin D_f \\ \beta_2(\beta_1(\beta 2(d))) & : \quad \beta_2(d) \in D_f \end{cases}$
4. $\forall d \in D', \quad \beta'_3(d) = \beta_3(d)$

Definition 15 (Volume Contraction) *Let $G = (D, \beta_1, \beta_2, \beta_3)$ be a 3D combinatorial map and let $G' = (D', \beta'_1, \beta'_2, \beta'_3)$ be the resulting map after the contraction of a selected 3-cell $V = (D_v, \beta_1, \beta_2)$. $G' = G/D_v$ is defined by*

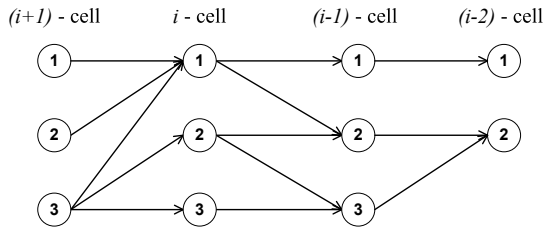
1. $D' = D \setminus D_v$
2. $\forall d \in D', \quad \beta'_1(d) = \beta_1(d)$
3. $\forall d \in D', \quad \beta'_2(d) = \beta_2(d)$
4. $\forall d \in D', \quad \beta'_3(d) = \begin{cases} \beta_3(d) & : \quad \beta_3(d) \notin D_v \\ \beta_3(\beta_2(\beta_3(d))) & : \quad \beta_3(d) \in D_v \end{cases}$

The properties of each contraction operation will be discussed later in this chapter.

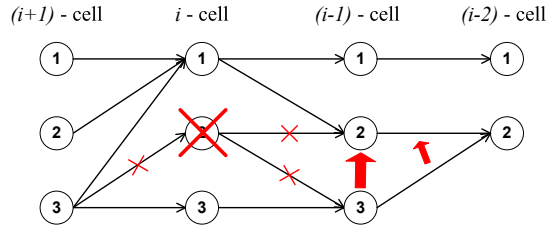
Since the i-cell is effectively removed by the contraction operation all bounding relations from $(i+1)$-cells to the contracted i-cell are removed as well. On the other hand, all incidence relations of the two $(i-1)$-cells that are merged as a result of the contraction operation to other i-cells are inherited by the surviving $(i-1)$-cell. In Fig. 4.1 the bounding relationship diagram of the contraction of an i-cell is shown.

As such a contraction operation can be understood as the reduction of an i-cell to an $(i-1)$-cell. E.g. an edge is contracted into a vertex, a face is contracted into an edge and a volume is contracted into a face.

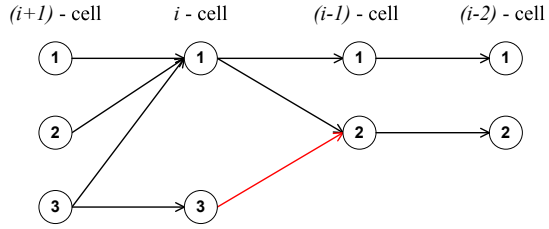
Not every i-cell may be contracted within a given combinatorial map. Instead it must be ensured that applying a contraction operation on an i-cell will again produce a consistent combinatorial map and that the map is topologically equivalent. For this purpose five conditions are presented in this section that must be met by every i-cell that shall be contracted.



a) configuration before contraction



b) contraction of the (i)-cell with index 2



c) final configuration after the contraction

Figure 4.1: Contraction of an i-cell. (a) shows the bounding relationship diagram before the contraction operation. By contracting the i-cell with index 2, the two bounding $(i-1)$-cells with index 2 and 3 are merged. As a result the bounding relation from the $(i+1)$-cell with index 3 to the contracted i-cell is removed as well. (b). In the final configuration (c) the incidence relations from the $(i-1)$-cells 2 and 3 are inherited by the surviving $(i-1)$-cell with index 2.

4.1.1 Condition 1: Dual Degree

[Damiand and Lienhardt, 2003] present the dual-degree as one condition for the contraction operation. An i-cell may only be contracted if it has a dual-degree of 2.

Condition 1 *Let c be an i-cell of a given combinatorial map $G = (D, \beta_1, \beta_2, \beta_3)$. c may only be contracted if*

$$\bar{\delta}(c) = 2.$$

This means that an i-cell can be contracted only if it is bounded exactly two times by $(i-1)$-cells (Either one $(i-1)$-cell bounding the i-cell twice, or two different $(i-1)$-cells, each bounding the i-cell once.). A volume may only be contracted if it is bounded two times by faces, a face may only be contracted if it is bounded two times by edges and an edge may only be contracted if it is bounded two times by vertices. Note that this condition does not need to be validated for edges, as the dual-degree is always two for an edge.

4.1.2 Condition 2: Bounding and Incidence Relations

Strict bounding and incidence relations must be enforced within combinatorial maps due to the way i-cells are encoded. Each cell must bound at least one other cell and must also be bounded by at least one other cell after a contraction operation.

As a result of this, a contraction might remove other cells as a side-effect. Therefore condition 2 is defined to disallow such contractions.

Condition 2 *Let c_i be an i-cell of a given combinatorial map $G = (D, \beta_1, \beta_2, \beta_3)$ and let C_j be the set of all j-cells of G with $j \neq i$. c_i may only be contracted if*

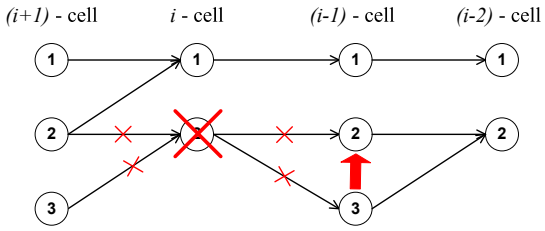
$$\forall c_j \in C_j, \quad D(c_j) \not\subseteq D(c_i).$$

This condition can be derived from the definition of a contraction operation. According to the previous sections, the contraction of an i-cell c_i is the removal of this i-cell and the merging of its two bounding $(i-1)$-cells. All other cells must survive.

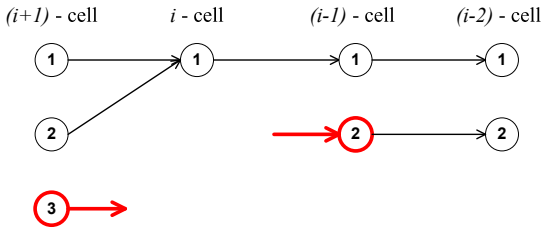
The removal of c_i is equivalent to the removal of all of its darts from the map. Since each dart is associated with exactly one cell of every dimension, the darts belonging to the removed cell will also be removed from all other

cells. Thus the contraction of a specific i-cell c_i might also remove a j-cell c_j (with $i \neq j$) if all darts defining c_j are also part of c_i.

This means that if there is at least one other cell in the combinatorial map that is defined only by darts belonging to c_i, then this cell would get removed as well. This would violate the definition that only c_i is removed by the contraction operation and is therefore not allowed. Fig. 4.2 shows an example of an invalid contraction violating this constraint.



a) contraction of the (i)-cell with index 2



b) configuration after the contraction

Figure 4.2: Contraction of an i-cell that violates the "Bounding and Incidence Relations" condition. (a) shows the bounding relationship diagram before the contraction of the i-cell with index 2. The $(i+1)$-cell with index 3 is only bounded by the contracted i-cell which leads to an invalid configuration (b). In a similar way the contracted i-cell was the only cell being incident to the $(i-1)$-cell with index 2 which also leads to an invalid configuration after the contraction.

4.1.3 Condition 3: Self-Loops

As [Brun and Kropatsch, 1999a] show, the contraction of a self-loop is not allowed. In a general sense, a self-loop is an i-cell that is bounded two or more times by at least one $(i-1)$-cell. This means that a cell may not be contracted if it is bounded by a specific cell more than once. E.g. an edge

that is connected to the same vertex on both endpoints or a face that is bounded twice by the same edge.

Condition 3 *Let c be an i-cell of a combinatorial map G. And let c be bounded by two $(i-1)$-cells b_1 and b_2. c may only be contracted if*

$$b_1 \neq b_2.$$

[Brun and Kropatsch, 1999a] show further that multiple contraction operations may only be applied simultaneously on a combinatorial map if the contracted i-cells do not form a circuit.

4.1.4 Condition 4: Common $(i-2)$-cells

The two $(i-1)$-cells bounding the i-cell are merged during a contraction. The fourth condition validates this merging process by ensuring that the resulting $(i-1)$-cell is non-ambiguous.

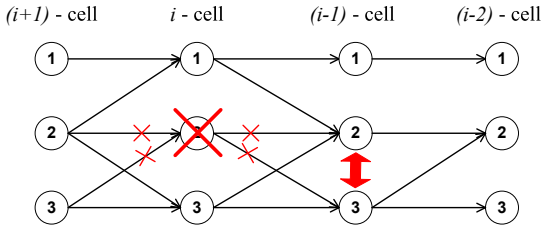
Condition 4 *Given a combinatorial map G. Let c_i be an i-cell of G with $i \geq 2$ that is bounded by exactly two $(i-1)$-cells c_1 and c_2. Also let B_1 be the set of $(i-2)$-cells bounding c_1 and let B_2 be the set of $(i-2)$-cells bounding c_2. c_i may only be contracted if*

$$B_1 = B_2.$$

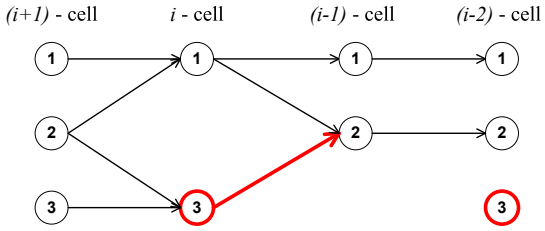
This condition can also be derived from the definition of the contraction operation: a contraction of an i-cell consists of the removal of the i-cell and the merging of the two bounding $(i-1)$-cells. The merging removes one of these two $(i-1)$-cells. The surviving cell inherits the incidence relations of both $(i-1)$-cells, but the bounding relations are unchanged.

This means that the bounding relations of the surviving cell would depend on which of the two cells survives the contraction operation if they have different bounding relations. The result of the contraction would therefore be ambiguous since the contraction operation does not define which cell should survive. This is not allowed.

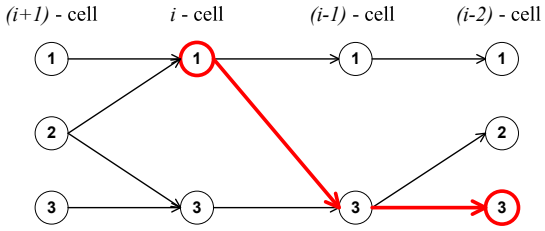
The bounding relationship diagram of such an invalid contraction is shown in Fig. 4.3. The i-cell with index 2 is chosen for contraction although its



a) contraction of the (i)-cell with index 2



b) $(i-1)$-cell with index 2 is chosen to survive the contraction



c) $(i-1)$-cell with index 3 is chosen to survive the contraction

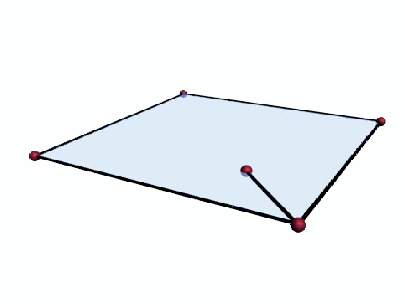
Figure 4.3: Contraction of an i-cell that violates the "common $(i-2)$-cells" condition. (a) shows the bounding relationship diagram before the contraction of the i-cell with index 2. The $(i-1)$-cells with index 2 and 3 are merged during the operation but have different bounding relations. (b,c) show the different results of the configuration depending on which of these two cells is chosen to survive.

bounding $(i-1)$-cells have different bounding relations. This leads to different and invalid results of the contraction operation if either of the two cells is chosen to survive.

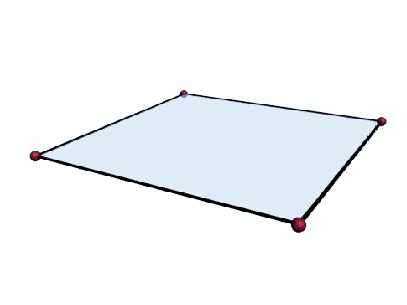
If the $(i-1)$-cell with index 2 is chosen (Fig. 4.3b), then the bounding relation to the $(i-2)$ cell with index 3 is lost and this cell gets disconnected. If the $(i-1)$-cell with index 3 is chosen, then the i-cell with index 1 becomes bounded by the $(i-2)$-cell with index 3 via the surviving $(i-1)$-cell with index 3 (Fig. 4.3c).

Fig. 4.4 gives an example of such an i-cell that may not be contracted because of this condition: a volume that is bounded by two faces. This volume cannot be contracted since the bounding faces do not share the same edges - one face has an additional edge. If the volume would be contracted the resulting face would either have this additional edge or not, depending on which face would be chosen to survive.

a) volume before the contraction



b) result of contraction if the upper face survives



c) result of contraction if the lower face survives

Figure 4.4: (a) shows an example of a volume that is not contractible because the two bounding faces are not bounded by the same edges. In this example the upper face has an additional edge. The face resulting from the contraction will be different depending on which of the two faces the volume is contracted to (b,c).

4.1.5 Condition 5: Background vertex

The fifth condition only applies to edges. When encoding the data as vertices, an additional vertex is added to the map representing the infinite background (see section 2.5). All vertices on the border of the initial grid-like map are adjacent to the background and therefore connected via an edge to this background vertex.

Contracting an edge is equivalent with merging two data nodes when using this representation of data. So the contraction of an edge that is connected to the background vertex would result in merging a data node with the background. Within this report it is assumed that the data encoded by a 3D combinatorial map is read from a three dimensional matrix encoding the voxels of a 3D image. Each of these voxels is part of one connected component and none is part of the infinite background region. For this reason condition 5 prohibits the contraction of an edge that is connected to the background vertex.

Condition 5 *Let e be a 1-cell of a combinatorial map G that is bounded by the two 0-cells v_1 and v_2. Also let v_{BG} be the 0-cell representing the background. e may only be contracted if*

$$v_1 \neq v_{BG} \quad \wedge \quad v_2 \neq v_{BG}$$

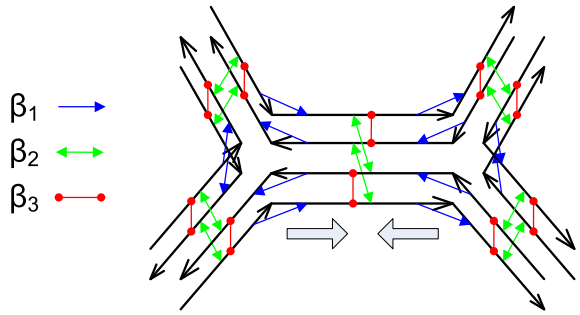
Note that this condition is not necessary if the 3D image is allowed to contain voxels that belong to the infinite background. This can be implemented by assigning specific values (e.g. a specific color) to all voxels belonging to the background region as well as to the background vertex itself.

4.1.6 Edge Contraction

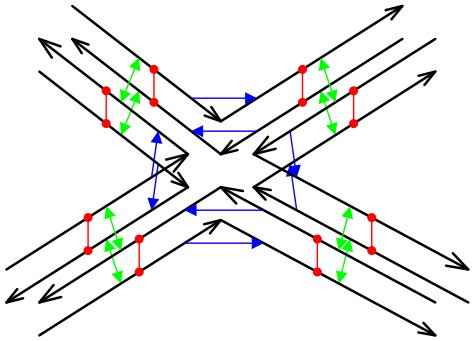
The contraction of an edge consists of removing the selected edge and merging the two vertices the edge connects. Since the data representation used within this report encodes the voxels as vertices, the edge contraction is used to merge two data nodes that belong to the same component. E.g. two voxels that have the same value when implementing a connected component algorithm.

Condition 1 (Dual Degree) does not need to be checked since the dual-degree of an edge is always 2 (An edge is always bounded exactly twice by

vertices). Condition 4 (Common $(i-2)$-cells) only applies to i-cells with $i \geq 2$ and is therefore not relevant for 1-cells. Conditions 2, 3 and 5 must be validated before contracting the edge: An edge may only be contracted if the operation removes no other i-cell from the map but the contracted edge itself and one of the vertices bounding the edge. An edge may not be contracted if it is a self-loop or if it is connected to the background vertex.



a) initial configuration



b) configuration after the edge contraction

Figure 4.5: Contraction of an edge. (a) shows the initial combinatorial map before the contraction. After the contraction the two vertices are merged (b).

Fig. 4.5 shows an example of an edge contraction. All darts belonging to the edge are removed and the permutations of the surviving darts belonging to the two vertices are adjusted to form a single vertex.

Applying a contraction operation will remove all darts of the contracted i-cell from the set of darts D. In addition the permutations β_1, β_2 and β_3 are adjusted to encode the relations of the surviving cells.

In the case of an edge contraction β_2 and β_3 will remain unchanged, since

a 1-cell D_e is defined as the set of all darts that can be obtained by applying β_2 and β_3 in any possible combination on any dart of that 1-cell. Therefore β_2 or β_3 will return a surviving (non-surviving) dart when applied on a surviving (non-surviving) dart. β_1 is adjusted for all darts whose β_1 successor is a dart of the contracted edge. In this case the first dart of the β_1 cycle that does not belong to the contracted edge will be selected as the new β_1 successor.

Proposition 4.1 *Given a combinatorial map G and an edge $D_e \in G$. The contraction of the edge $G' = G/D_e$ produces a valid combinatorial map if conditions 2, 3 and 5 are fulfilled.*

Proof: G' is a valid combinatorial map as it complies to definition 10:

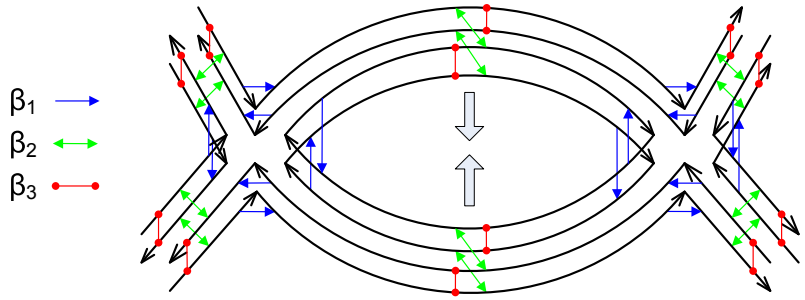
- D' is a finite set of darts as D' is a subset of D.
- $\beta'_1(d)$, $d \in D'$ is a permutation on D' since it is either equal to $\beta_1(d)$ (if $\beta_1(d)$ points to a surviving dart) or equal to the first dart of the cycle $\beta_1^\star(d)$ that does not belong to the contracted edge. There is always such a surviving dart in the cycle $\beta_1^\star(d)$, since condition 3 disallows the contraction of self-loops and cycles and because condition 2 disallows the contraction of an edge if it is the only edge bounding a face. Thus there must be a surviving edge for each face in G after an edge contraction and therefore also a surviving dart in each cycle $\beta_1^\star(d)$.
- $\beta'_2(d)$ and $\beta'_3(d)$, $d \in D'$, are involutions since $\beta'_2(d) = \beta_2(d)$ and $\beta'_3(d) = \beta_3(d)$.

□

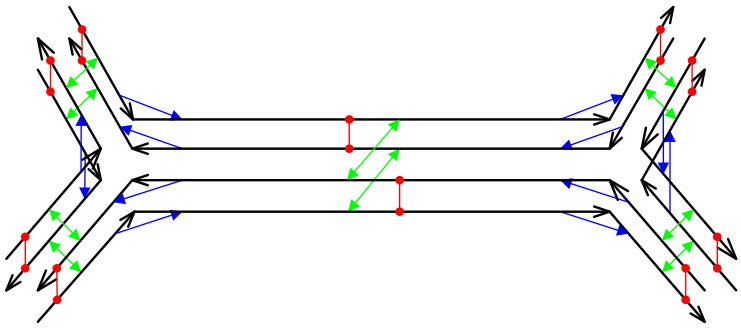
4.1.7 Face Contraction

A face that is bounded by exactly two edges can be reduced to a single edge that is topologically equivalent to the face. Such a face contraction is used to reduce the complexity of a combinatorial map. The operation removes the face and merges the bounding edges.

Conditions 1 to 4 have to be checked when selecting a face for contraction: Only faces bounded by exactly two different edges may be contracted, the face itself and one of the two bounding edges must be the only i-cells eliminated from the map by the contraction, and both edges bounding the face must be



a) initial configuration



b) configuration after the face contraction

Figure 4.6: Contracting a face. The face bounded by two edges (a) is contracted into a single edge (b).

bounded by the same vertices. Condition 5 (Background vertex) is irrelevant as it only applies to edges. An example of a face contraction can be seen in Fig. 4.6.

As with the edge contraction, only one permutation must be adjusted for the contraction of a face D_f. β_1 and β_3 remain unchanged as a face is defined by $F = (D_f, \beta_1, \beta_3)$.

Since only faces with a dual-degree of two may be contracted, the β_2 successor can always be set to the β_2-β_1-β_2 successor for any dart having a non-surviving β_2 successor (See Fig. 4.6).

That way the first face around the opposite edge of the contracted face is selected.

Proposition 4.2 *Given a combinatorial map G and a face $D_f \in G$. The contraction of the face $G' = G/D_f$ produces a valid combinatorial map if conditions 1 to 4 are fulfilled.*

Proof: G' is a valid combinatorial map as it complies to definition 10:

- D' is a finite set of darts as D' is a subset of D.
- $\beta'_1(d)$, $d \in D'$, is a permutation on D' since $\beta'_1(d) = \beta_1(d)$.
- $\beta'_2(d)$, $d \in D'$, is an involution as it is either set to $\beta_2(d)$ or to $\beta_2(\beta_1(\beta_2(d)))$. Condition 1 allows the contraction of a face only if it is bounded exactly twice by edges. Further, condition 3 ensures that the face is not bounded by the same edge twice. For such faces β_1 is an involution (see Fig. 4.6). Therefore $\beta_2(\beta_1(\beta_2(d)))$ is an involution if $\beta_2(d) \in D_f$. Thus β'_2 is an involution in both cases.
- $\beta'_3(d)$, $d \in D'$, is an involution since $\beta'_3(d) = \beta_3(d)$.

□

Proposition 4.3 *Given a combinatorial map G and a face $D_f \in G$. The contraction of the face $G' = G/D_f$ will not change the genus of the bounding surfaces if conditions 1 to 4 are fulfilled.*

Proof: It can be shown that a face contraction will produce a topologically equivalent map if all conditions are fulfilled by comparing the Euler characteristic of the bounding surfaces. This is done by comparing the Euler Characteristic $\chi = v - e + f$ for the initial map to the Euler Characteristic $\chi' = v' - e' + f'$ after the operation has been applied.

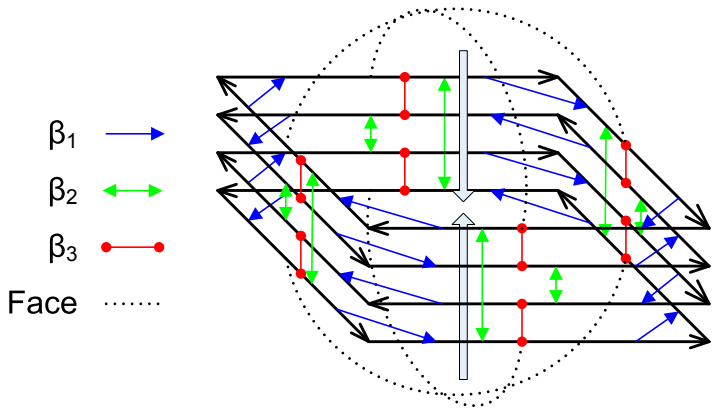
Since vertices are used to represent voxels in this report, the contraction of a face corresponds to the removal of an edge. Because of condition 1 this edge is incident to a face twice, and because of condition 3 it must be incident to two different faces. Further the removal of an edge is defined as the removal of this edge and the merging of its two incident faces. No other i-cells will be removed as this would violate condition 2. Thus $v' = v$, $e' = e - 1$, and $f' = f - 1$. χ' is therefore equivalent to χ:

$$\chi' = v' - e' + f' = v - (e - 1) + (f - 1) = v - e + f = \chi.$$

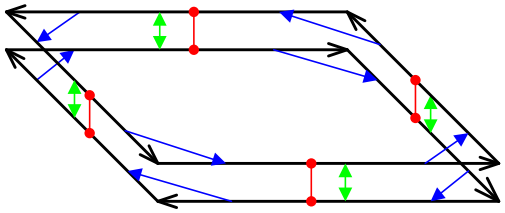
Therefore the genus of the bounding surfaces is not changed by the operation. □

4.1.8 Volume Contraction

A volume may be contracted if it is bounded by exactly two faces. This operation can be imagined as "flattening" a pillow-like object. Like the face contraction this operation is used to simplify a given combinatorial map without changing the topological relations within the map.



a) initial configuration



b) configuration after the volume contraction

Figure 4.7: The pillow-like volume in the combinatorial map is selected for contraction (a). The outline of the bounding faces is visualized by the dotted lines. (b) shows the resulting map after the contraction.

As with the contraction of a face, conditions 1 to 4 have to be met by a volume in order to be contractible. Therefore only volumes that are bounded by exactly two different faces may be contracted and the the volume itself and one of the two bounding faces must be the only i-cells eliminated from the map by the contraction. Further, both faces bounding the volume must be bounded by the same edges.

Fig. 4.7 displays a combinatorial map before and after a volume contrac-

tion.

A volume D_v is defined by the two permutations β_1 and β_2. Thus these permutations will not change due to a volume contraction. β_3 is adjusted for all darts having a non-surviving β_3 successor.

Due to the fact that only volumes that are bounded by exactly two faces may be contracted, β_3 is set to the β_3-β_2-β_3 successor in these cases. This corresponds to β_3-connecting the two outer sides of both faces of the volume, thus creating a new face. The inner sides of the faces are removed as being part of the contracted volume.

Proposition 4.4 *Given a combinatorial map G and a volume $D_v \in G$. The contraction of the volume $G' = G/D_v$ produces a valid combinatorial map if conditions 1 to 4 are fulfilled.*

Proof: G' is a valid combinatorial map as it complies to definition 10:

- D' is a finite set of darts as D' is a subset of D.
- $\beta'_1(d)$, $d \in D'$, is a permutation on D' since $\beta'_1(d) = \beta_1(d)$.
- $\beta'_2(d)$, $d \in D'$, is an involution since $\beta'_2(d) = \beta_2(d)$.
- $\beta'_3(d)$, $d \in D'$, is an involution since $\beta_3(\beta_2(\beta_3(d)))$ is an involution and $\beta'_3(d)$ is either set to β_3 or $\beta_3(\beta_2(\beta_3(d)))$.

□

Proposition 4.5 *Given a combinatorial map G and a volume $D_v \in G$. The contraction of the volume $G' = G/D_v$ will not change the genus of the bounding surfaces if conditions 1 to 4 are fulfilled.*

Proof: Again the dual operation - the removal of a vertex - is examined.

Because of condition 1 this vertex is incident to edges twice, and because of condition 3 it must be incident to two different edges. Further the removal of a vertex is defined as the removal of this vertex and the merging of its two incident edges. No other i-cells will be removed as this would violate condition 2. Thus $v' = v-1$, $e' = e-1$, and $f' = f$. χ' is therefore equivalent to χ:

$$\chi' = v' - e' + f' = (v-1) - (e-1) + f = v - e + f = \chi.$$

Therefore the genus of the bounding surfaces is not changed by the operation. □

4.1.9 Simplification of a Cube

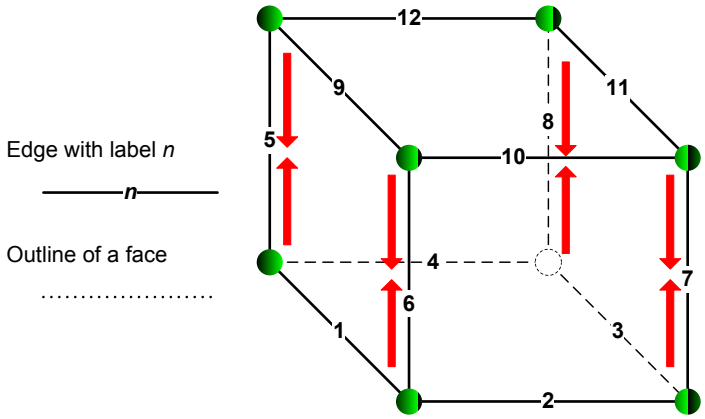
This section describes how a combinatorial map can be simplified using contraction operations in the example of a $2 \times 2 \times 2$ cube being reduced to a single vertex (Fig. 4.8). In this example six steps are necessary to reduce the configuration and all three defined contraction operations are applied to the map during this simplification process.

1. The initial configuration is a cube consisting of 8 vertices, 12 edges, 6 faces, and 2 volumes - one volume inside and one volume outside of the cube (see Fig. 4.8a). In this configuration there are no contractible faces, as each of the faces has a dual-degree of four - it is bounded by four edges. The inner volume is not contractible as its dual-degree is six, being bounded by all six faces. Finally all edges are contractible as they fulfill all conditions. The four vertical edges with index 5, 6, 7 and 8 are chosen for contraction as the first simplification step.

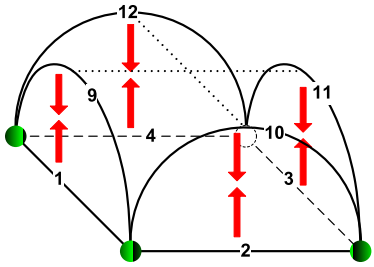
2. The contraction of the edges reduced the number of edges to eight and the number of vertices to four. Further, it reduced the dual-degree of the four vertical faces to two. They are now bounded only by two edges: 1&9, 2&10, 3&11 and 4&12. As they also fulfill the other conditions they can be contracted in the next simplification step (see Fig. 4.8b).

3. After the contraction of the four faces there are four edges and two faces left. Additionally the inner volume is now bounded only by the two remaining faces, meaning that its dual-degree is two and is eligible for contraction (see Fig. 4.8c).

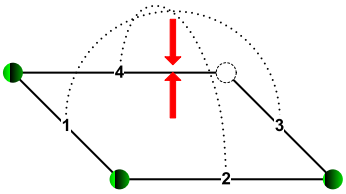
4. The volume contraction removed the inner volume from the configuration and reduced the number of faces to one. This face is not contractible as its dual-degree is four, but each of the edges may be contracted (see Fig. 4.8d). The edges with index 1 and 3 are contracted in this step.



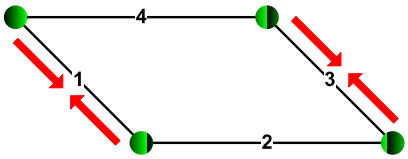
a) Step 1: the 4 vertical edges of the cube are selected for contraction.



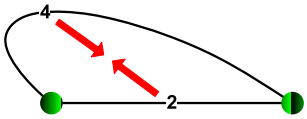
b) Step 2: the 4 vertical faces are contracted.



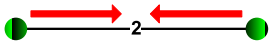
c) Step 3: the volume is contracted.



d) Step 4: 2 edges are selected for contraction.



e) Step 4: The final face gets contracted.



f) Step 5: The remaining edge is selected for contraction.

c) Final configuration: The cube is reduced to a single vertex

Figure 4.8: Example of the simplification of a cube using contraction operations. The initial configuration (a) is reduced in six steps to a single vertex (g). This is done by the successive application of edge contractions (a,d,f), face contractions (b,e) and one volume contraction (c).

5. As a result of the two edge contractions, the final face now has a dual-degree of two, being bounded only by the two remaining edges. It is therefore chosen for contraction (see Fig. 4.8e).

6. The cube is now reduced to a single edge connecting two vertices (see Fig. 4.8f). This edge is contracted in the last simplification step, leaving a single vertex (see Fig. 4.8g).

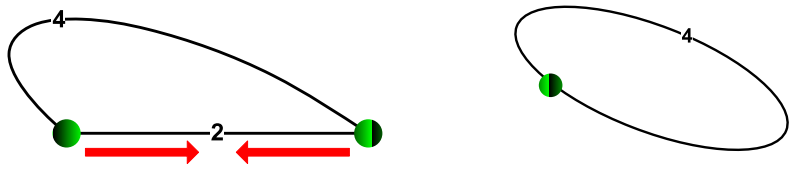
Table 4.1 shows the number of i-cells in this example after each simplification step. The initial configuration consists of 28 i-cells (8 vertices, 12 edges, 6 faces and 2 volumes). This number gets reduced to two remaining i-cells (one vertex and one volume) in the final configuration.

	Vertex	Edge	Face	Volume
Initial configuration	8	12	6	2
Step 1	4	8	6	2
Step 2	4	4	2	2
Step 3	4	4	1	1
Step 4	2	2	1	1
Step 5	2	1	0	1
Step 6	1	0	0	1

Table 4.1: Number of i-cells after each simplification step.

Note that step 5 and step 6 actually violate condition 2 by removing the last face and edge from the combinatorial map and are therefore not allowed. For the clarity of this example it is assumed that this configuration is part of a larger combinatorial map in which the remaining vertex is still incident to other edges and faces. A more detailed discussion about minimal configuration can be found later in this report.

When selecting candidates for a simplification there is often more than one possibility. Fig. 4.9 presents such an alternative way to simplify the initial configuration. In step 4b (Fig. 4.9a) edge 2 is selected for contraction instead of the face. This is allowed since edge 2 is bounded by two vertices. This leads to an alternative final configuration consisting of one vertex, one edge, one face and one volume (Fig. 4.9b). The cube is represented by the single remaining vertex which is incident to edge 4. This edge is a self-loop that bounds the only face.



a) Step 4b: edge 2 is selected for contraction. b) Final configuration

Figure 4.9: Alternative example for the simplification of the cube from Fig. 4.8. Instead of selecting the face for contraction, edge 2 is selected in step 4 (a). This reduces the cube to a final configuration consisting of one vertex that is incident to one edge, one face and one volume (b).

4.2 Removal Operation

According to [Damiand and Lienhardt, 2003, Brun and Kropatsch, 1999a] the removal operation is the dual counterpart to a contraction. Each contraction operation is equivalent to a removal operation in the dual graph and each removal operation is equivalent to a contraction operation in the dual graph. Table 4.2 shows this relation between both operations.

Contraction Operation	Removal Operation in the dual graph
Volume Contraction	Vertex Removal
Face Contraction	Edge Removal
Edge Contraction	Face Removal

Table 4.2: Relation of the contraction and removal operations in the primal and dual graph.

Thus a removal operation is defined as the removal of the i-cell and the merging of two $(i+1)$-cells being incident to this i-cell. When merging the two $(i+1)$-cells, one of them gets effectively removed from the combinatorial map. E.g. when removing a face, the two incident volumes are merged into a single volume.

A removal operation merges two $(i+1)$-cells. Therefore only cells of dimension $i \leq 2$ can be removed in a 3D combinatorial map. The three possible removal operations are: vertex removal, edge removal, and face removal. Specific definitions are provided for these three removal operations.

Definition 16 (Vertex Removal) *Let $G = (D, \beta_1, \beta_2, \beta_3)$ be a 3D combinatorial map and let $G' = (D', \beta'_1, \beta'_2, \beta'_3)$ be the resulting map after the removal of a selected 0-cell $V = (D_v, \beta_2\beta_1, \beta_3\beta_1)$. $G' = G \setminus D_v$ is defined by*

1. $D' = D \setminus D_v$
2. $\forall d \in D', \quad \beta'_1(d) = \begin{cases} \beta_1(d) & : \quad \beta_1(d) \notin D_v \\ \beta_1(\beta_1(d)) & : \quad \beta_1(d) \in D_v \end{cases}$
3. $\forall d \in D', \quad \beta'_2(d) = \begin{cases} \beta_2(d) & : \quad \beta_2(d) \notin D_v \\ \beta_2(\beta_1(d)) & : \quad \beta_2(d) \in D_v \end{cases}$
4. $\forall d \in D', \quad \beta'_3(d) = \begin{cases} \beta_3(d) & : \quad \beta_1(d) \notin D_v \\ \beta_3(\beta_1(d)) & : \quad \beta_3(d) \in D_v \end{cases}$

Definition 17 (Edge Removal) *Let $G = (D, \beta_1, \beta_2, \beta_3)$ be a 3D combinatorial map and let $G' = (D', \beta'_1, \beta'_2, \beta'_3)$ be the resulting map after the removal of a selected 1-cell $E = (D_e, \beta_2, \beta_3)$. $G' = G \setminus D_e$ is defined by*

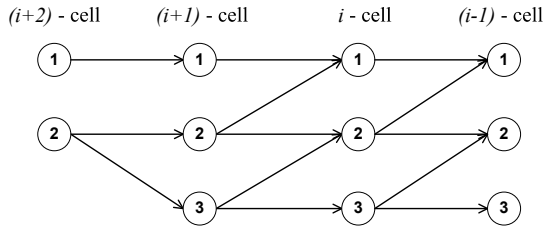
1. $D' = D \setminus D_e$
2. $\forall d \in D', \quad \beta'_1(d) = \begin{cases} \beta_1(d) & : \quad \beta_1(d) \notin D_e \\ \beta_1(\beta_2(\beta_1(d))) & : \quad \beta_1(d) \in D_e \end{cases}$
3. $\forall d \in D', \quad \beta'_2(d) = \beta_2(d)$
4. $\forall d \in D', \quad \beta'_3(d) = \beta_3(d)$

Definition 18 (Face Removal) *Let $G = (D, \beta_1, \beta_2, \beta_3)$ be a 3D combinatorial map and let $G' = (D', \beta'_1, \beta'_2, \beta'_3)$ be the resulting map after the removal of a selected 2-cell $F = (D_f, \beta_1, \beta_3)$. $G' = G \setminus D_f$ is defined by*

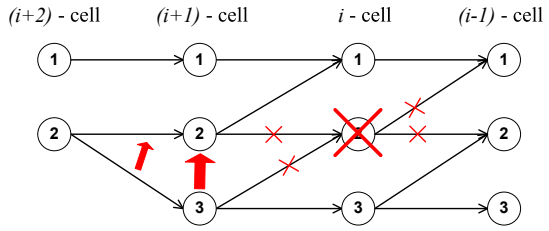
1. $D' = D \setminus D_f$
2. $\forall d \in D', \quad \beta'_1(d) = \beta_1(d)$
3. $\forall d \in D', \quad \beta'_2(d) = \begin{cases} \beta_2(d) & : \quad \beta_2(d) \notin D_f \\ \beta_2(\beta_3(\beta_2(d))) & : \quad \beta_2(d) \in D_f \end{cases}$
4. $\forall d \in D', \quad \beta'_3(d) = \beta_3(d)$

The properties of these 3 operations will be discussed later in this chapter.

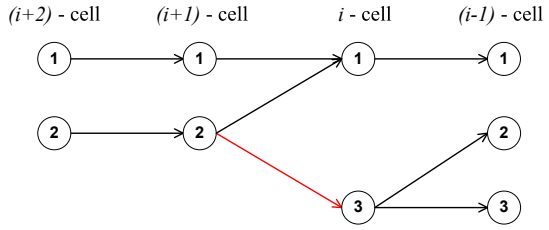
The bounding relations from the merged $(i+1)$-cells to the removed i-cell get removed while the bounding relations to all other i-cells are inherited by the newly created $(i+1)$-cell. Fig. 4.10 shows the bounding relationship diagram of the removal of an i-cell.



a) configuration before the removal



b) removal of the (i)-cell with index 2



c) final configuration after the removal

Figure 4.10: Removal of an i-cell. (a) shows the bounding relationship diagram before the removal operation. The two $(i+1)$ cells with index 2 and 3 are merged. In the final configuration (c) the bounding relations from the $(i+1)$-cells 2 and 3 are inherited by the surviving cell with index 2.

A removal operation can be imagined as the merging of two neighboring $(i+1)$-cells by removing the boundary between these two cells. E.g. creating a larger volume by removing the face that separates two volumes or creating a larger face by removing the edge that separates two faces.

Due to the dual relation between the contraction and the removal operation, the conditions defined for the contraction operation apply to the removal operation accordingly.

4.2.1 Condition 6: Degree

This condition corresponds to condition 1 for contraction operations, the "dual-degree" condition. An i-cell may only be removed if it is incident exactly two times to $(i+1)$-cells. E.g. an edge may only be removed if it is incident exactly twice to faces. Note that a face always fulfills this condition in 3D as it is always incident exactly two times to volumes.

Condition 6 *Let c be an i-cell of a given combinatorial map $G = (D, \beta_1, \beta_2, \beta_3)$. c may only be removed if*

$$\delta(c) = 2.$$

4.2.2 Condition 7: Bounding and Incidence Relations

Condition 2 (Bounding and Incidence Relations) also applies to removal operations. When selecting an i-cell for removal it must be ensured that only this cell gets removed from the map as a result of the operation. Thus condition 2 is extended to be valid for removal operations as well.

Condition 7 *Let c_i be an i-cell of a given combinatorial map $G = (D, \beta_1, \beta_2, \beta_3)$ and let C_j be the set of all j-cells of G with $j \neq i$. c_i may only be contracted or removed if*

$$\forall c_j \in C_j, \quad D(c_j) \not\subseteq D(c_i).$$

4.2.3 Condition 8: Bridges

A bridge is an i-cell that is the only path between two connected components of the combinatorial map. Thus the removal of such a bridge

would disconnect the two connected components and is therefore not allowed [Brun and Kropatsch, 1999a].

The representation of a bridge in the dual combinatorial map is a self-loop. Therefore removing a bridge is equivalent to the contraction of a self-loop in the dual combinatorial map [Brun and Kropatsch, 1999a], which is disallowed by condition 3. Thus the "Bridges" condition is the symmetrical counterpart to the "Self-Loops" condition.

A bridge can be detected be examining the two incident $(i+1)$-cells when validating this condition. Due to condition 6, the degree of the i-cell to be removed must be 2 which means that the i-cell is two times incident to $(i+1)$-cells. If it is incident to the same $(i+1)$-cell twice, then the i-cell is a bridge and cannot be removed.

Condition 8 *Let c be an i-cell of a combinatorial map G. And let c be incident to exactly two $(i+1)$-cells b_1 and b_2. c may be removed only if*

$$b_1 \neq b_2.$$

4.2.4 Condition 9: Common $(i+2)$-cells

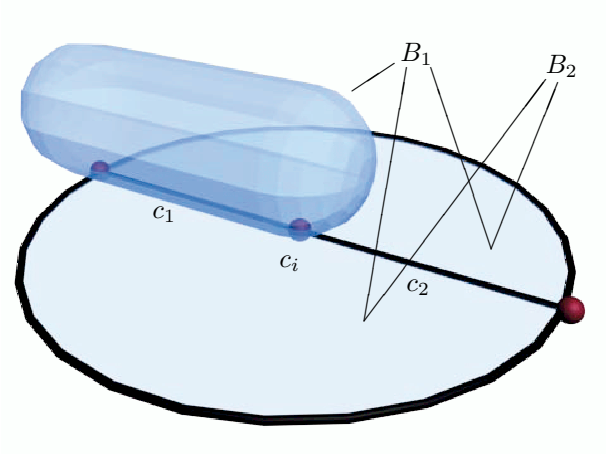
The ninth condition can be directly deduced from condition 4 using the symmetry between contraction and removal operations:

Condition 9 *Given a combinatorial map G. Let c_i be an i-cell of G with $i \leq 1$ that is incident to exactly two $(i+1)$-cells c_1 and c_2. Also let B_1 be the set of $(i+2)$-cells being incident to c_1 and let B_2 be the set of $(i+2)$-cells being incident to c_2. c_i may only be removed if*

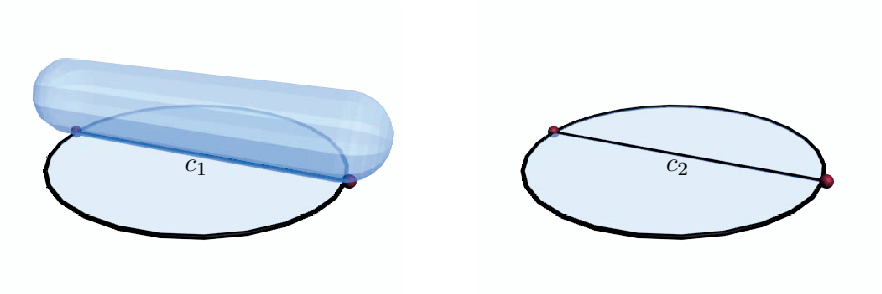
$$B_1 = B_2.$$

Fig. 4.11 shows an i-cell that cannot be removed because of condition 9. The middle vertex is incident to two edges and is therefore a candidate for removal, but the two edges are not incident to the same faces and therefore the removal is not allowed: The left edge is incident to an additional face (the capsule-like face only being bounded by the left edge twice). When removing the vertex this would lead to an ambiguous result as the new configuration

would either have this additional face or not, depending on which edge is chosen to survive (Fig. 4.11b,c).



a) configuration before the removal of vertex c_i



b) result of removal if edge c_1 survives c) result of removal if edge c_2 survives

Figure 4.11: (a) shows an example of a vertex c_i that cannot be removed as it would violate condition 9. The left edge (c_1) is incident to an additional face. Thus the result would be different depending on which edge would survive as the merged edge (b,c).

4.2.5 Condition 10: Connected Vertices

The tenth condition only applies to the removal operation and has no counterpart regarding the contraction operation. It ensures that vertices do not "break apart" as the result of a removal operation.

Condition 10 *Let G be a combinatorial map $G = (D, \beta_1, \beta_2, \beta_3)$, V be 0-cell $V = (D_v, \beta_2\beta_1, \beta_3\beta_1)$ and F be a 2-cell $F = (D_f, \beta_1, \beta_3)$. F may only be removed if*

$$(D_v \setminus D_f, \beta_2\beta_1, \beta_3\beta_1) \text{ is connected.}$$

As described in section 3.1, a vertex is defined as $V = (D', \beta_2\beta_1, \beta_3\beta_1)$ given a 3D combinatorial map $G = (D, \beta_1, \beta_2, \beta_3)$. As such, a vertex is defined as the set of all darts that can be reached by any combination of the permutations $\beta_2\beta_1$ and $\beta_3\beta_1$.

For a combinatorial map to be consistent, the opposite must be true as well: all darts belonging to the same vertex must be reachable from any dart of this vertex by a combination of the permutations $\beta_2\beta_1$ and $\beta_3\beta_1$. The removal of a face might remove the only path from one dart of the vertex to another using these two permutations, effectively dividing the vertex and making the combinatorial map inconsistent. Thus such a removal is not allowed.

This special case will be discussed in more detail in section 6.4.

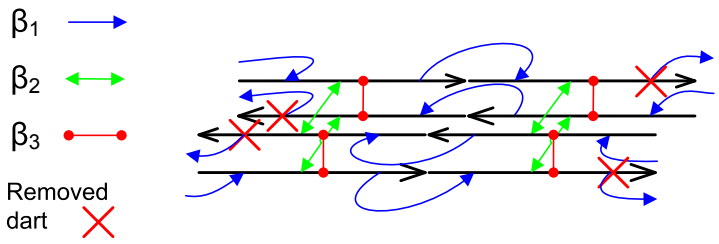
4.2.6 Vertex Removal

The removal of a vertex consists of removing this vertex from the combinatorial map and merging the two edges it separates. Conditions 6 to 9 must be checked before removing a vertex: The vertex must be incident to exactly two different edges, the vertex and one of the two incident edges must be the only i-cells removed from the map as a result of the removal operation, and both edges incident to the vertex must themselves be incident to the same faces.

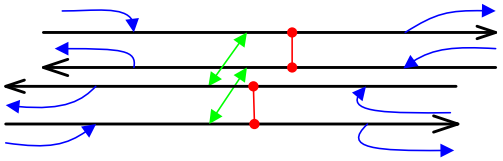
If the data that the combinatorial map describes is encoded by vertices, then the removal of vertices is not allowed at all. Removing a vertex in this case would remove a data node from the map.

An example of a vertex removal can be seen in Fig. 4.12. All darts associated with the vertex are removed from the map and the remaining darts of the two edges are connected to form a single edge.

Due to the way a vertex D_v is defined, all three permutations have to be adjusted for this operation (See Fig. 4.12).



a) initial configuration



b) configuration after the vertex removal

Figure 4.12: Removal of a vertex. The initial configuration shows two edges that are separated by a vertex. This vertex is only incident to these two edges (a). After the removal the two edges are merged into a single edge and all darts belonging to the vertex are removed from the map (b).

Proposition 4.6 *Given a combinatorial map G and a vertex $D_v \in G$. The removal of the vertex $G' = G \setminus D_v$ produces a valid combinatorial map if conditions 6 to 9 are fulfilled.*

Proof: G' is a valid combinatorial map as it complies to definition 10:

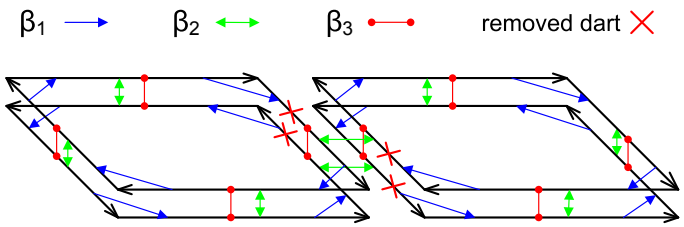
- D' is a finite set of darts as D' is a subset of D.
- $\beta'_1(d)$, $d \in D'$, is a permutation on D' since it is either equal to $\beta_1(d)$ (if $\beta_1(d)$ points to a surviving dart) or equal to $\beta_1(\beta_1(d))$. Condition 6 allows a removal of a vertex only if it is incident exactly twice to an edge and condition 8 ensures that it is incident to two different edges. Thus D_v cannot be the only vertex of an edge self-loop and $\beta_1(\beta_1(d))$ is defined and a valid permutation.
- $\beta'_2(d)$, $d \in D'$, is an involution on D' as it is either equal to $\beta_2(d)$ or equal to $\beta_2(\beta_1(d))$. $\beta_2(\beta_1(d))$ is an involution if the vertex that d belongs to, is incident twice to an edge. This is ensured by condition 6.

- $\beta_3'(d)$, $d \in D'$, is an involution on D' because it is either equal to $\beta_3(d)$ or equal to $\beta_3(\beta_1(d))$ which is also an involution.

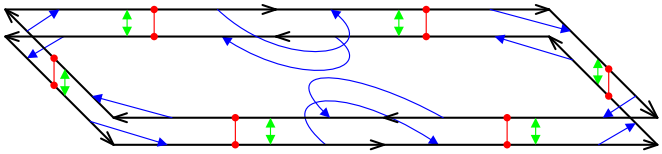
□

4.2.7 Edge Removal

An edge may be removed only if it is incident to exactly two faces. By removing the separating edge these two faces will become a single face. Conditions 6 to 9 must be fulfilled before an edge may be selected for removal. This means that the edge must be incident to exactly two different faces, the removal of the face may not remove any other i-cells from the map but the edge itself and one of the two faces incident to the edge. Finally both faces incident to the removed edge must be incident to the same volumes. Fig. 4.13 shows a combinatorial map before and after this operation.



a) configuration before the removal of the edge.



b) configuration after the removal.

Figure 4.13: Removal of an edge. (a) shows an edge that is incident to exactly two faces. It can be removed to merge the two faces into a single face (b).

As with the edge contraction, only the permutation β_1 needs to be adjusted for the removal of an edge D_e. All pairs of β_2- and β_3-connected darts are either removed completely as being part of the removed edge or will completely survive the operation. β_1 is set to the β_1-β_2-β_1 successor for those

darts whose β_1 successor is part of the removed edge. This corresponds to selecting the second edge around a vertex if the first edge was removed.

Proposition 4.7 *Given a combinatorial map G and an edge $D_e \in G$. The removal of the edge $G' = G \setminus D_e$ produces a valid combinatorial map if conditions 6 to 9 are fulfilled.*

Proof: G' is a valid combinatorial map as it complies to definition 10:

- D' is a finite set of darts as D' is a subset of D.
- $\beta'_1(d)$, $d \in D'$, is a permutation on D' since it is either equal to $\beta_1(d)$ or equal to $\beta_1(\beta_2(\beta_1(d)))$ which is an involution if the edge is incident exactly twice to a face. This is always the case because an edge removal is only allowed if condition 6 is fulfilled.
- $\beta'_2(d)$ and $\beta'_3(d)$, $d \in D'$, are involutions on D' since they are not changed by the operation.

□

Proposition 4.8 *Given a combinatorial map G and an edge $D_e \in G$. The removal of the edge $G' = G \setminus D_e$ will not change the genus of the bounding surfaces if conditions 6 to 9 are fulfilled.*

Proof: To show that an edge removal does not change the genus of the bounding surfaces, the corresponding contraction of a face in the dual is examined by comparing the Euler Characteristic before and after the operation.

Because of condition 6 this face is bounded by edges twice, and because of condition 8 it must be bounded by two different edges. Further, the contraction of a face is defined as the removal of this face and the merging of its two bounding edges. No other i-cells will be removed as this would violate condition 7. Thus $v' = v$, $e' = e - 1$, and $f' = f - 1$. χ' is therefore equivalent to χ:

$$\chi' = v' - e' + f' = v - (e - 1) + (f - 1) = v - e + f = \chi.$$

Therefore the genus of the bounding surfaces is not changed by the operation. □

4.2.8 Face Removal

The last operation described in this section is the face removal. This operation joins two volumes that are separated by a face. Note that condition 6 (Degree) does not need to be checked since a face is always incident to volumes exactly twice in a connected 3D combinatorial map. Condition 9 (Common $(i+2)$-cells) is not relevant for 2-cells as well since it applies only to i-cells with $i \leq 1$. Conditions 7 (Bounding and Incidence Relations), 8 (Bridges) and 10 (Connected Vertices) must be validated before selecting a face for removal: The removed face and one of the incident volumes must be the only i-cells removed from the map, the face must be incident to two different volumes, and it must be ensured that the removal of the face will not create an invalid vertex.

Fig. 4.14 shows an example of such a face removal. In this figure only the removed face is drawn in complete detail. The other faces bounding the merged volumes are simplified for the clarity of the example. All darts of the face are removed and the β_2 involutions are corrected to create a single volume.

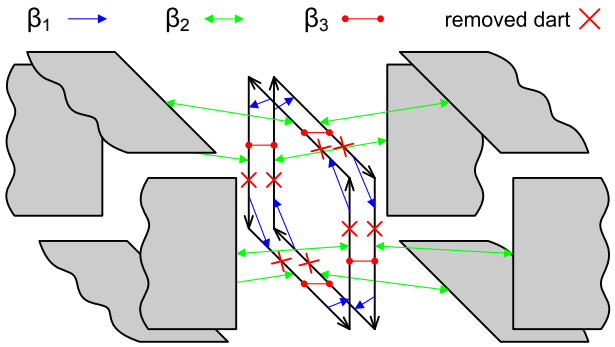
β_2 is the only permutation changed by the removal of a face $F = (D_f, \beta_1, \beta_3)$ as it is the only permutation that is not part of the definition of a face. β_2-β_3-β_2 is used wherever β_2 points to a dart that is part of the removed face. This corresponds to skipping the removed face and selecting the next face around the edge.

Proposition 4.9 *Given a combinatorial map G and a face $D_f \in G$. The removal of the face $G' = G \setminus D_f$ produces a valid combinatorial map if conditions 7, 8, and 10 are fulfilled.*

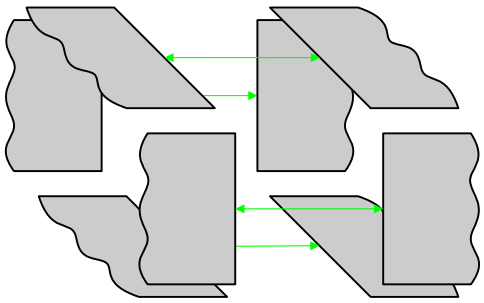
Proof: G' is a valid combinatorial map as it complies to definition 10:

- D' is a finite set of darts as D' is a subset of D.
- $\beta'_1(d)$, $d \in D'$, is a permutation on D' as it is equal to $\beta_1(d)$.
- $\beta'_2(d)$, $d \in D'$, is an involution as it is either equal to $\beta_2(d)$ or equal to $\beta_2(\beta_3(\beta_2(d)))$ which is also an involution.
- $\beta'_3(d)$, $d \in D'$, is a involution on D' as it is equal to $\beta_3(d)$.

□



a) configuration before the removal of the edge.



b) configuration after the removal.

Figure 4.14: Removal of a face. The face between the two volumes gets selected for removal (a). All darts of this face are removed and the two incident volumes are joined by setting the β_2 involutions accordingly (b).

Proposition 4.10 *Given a combinatorial map G and a face $D_f \in G$. The removal of the face $G' = G \setminus D_f$ will not change the genus of the bounding surfaces if conditions 6 to 9 are fulfilled.*

Proof: In a similar way as for an edge removal it can be shown that a face removal will not change the genus of the bounding surfaces if all conditions are fulfilled. This is done by examining the dual operation of an edge contraction.

Because of condition 6 this edge is bounded by vertices twice, and because of condition 8 it must be bounded by two different vertices. Further, the contraction of an edge is defined as the removal of this edge and the merging

of its two bounding vertices. No other i-cells will be removed as this would violate condition 7. Thus $v' = v - 1$, $e' = e - 1$, and $f' = f$. χ' is therefore equivalent to χ:

$$\chi' = v' - e' + f' = (v - 1) - (e - 1) + f = v - e + f = \chi.$$

Therefore the genus of the bounding surfaces is not changed by the operation. $\square$

Chapter 5

3D Combinatorial Pyramids

This chapter describes the process of building a 3D combinatorial pyramid using the operations defined in the previous chapter.

Section 5.1 shows how an initial map is created from 3-dimensional data. In section 5.2 an algorithm for constructing the complete pyramid from such an initial map is introduced, using contraction and removal operations. Finally, the last section of this chapter extends the concept of connecting walks introduced by [Brun and Kropatsch, 1999b], thus presenting a method for calculating the receptive field of any vertex in the top level of a 3D combinatorial pyramid.

5.1 Describing 3D Data

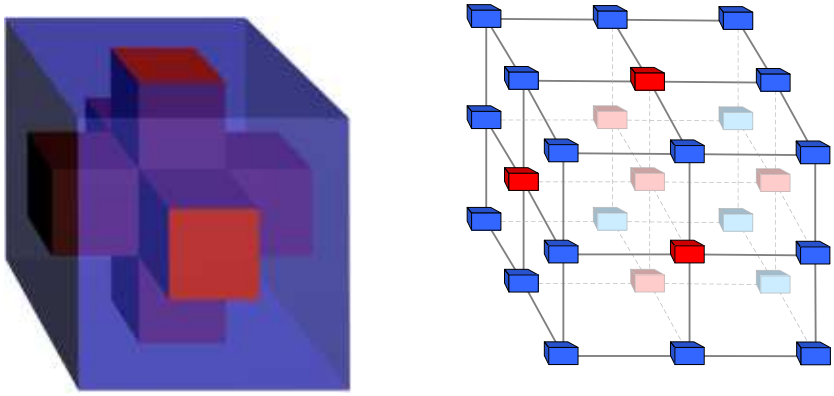
The first step in building a combinatorial pyramid is the creation of an initial map representing the data. This map is the base level of the pyramid.

In 2D this is done by converting the data in a grid-like 2D combinatorial map (see section 2.5). This method is suitable for data that is provided as a labaled 2D matrix (e.g. images). Each pixel is represented by a vertex in such a grid-like combinatorial map.

This method can be extended for labeled volumetric data, if the data can be accessed as a 3D matrix, e.g. stacks of images, the output of a 3D scanner, or movies. Time or the index of the frame is used as the third dimension when using movies as input data.

In these cases a grid-like 3D combinatorial map can be built where each voxel is represented by a vertex in the map. The adjacency of the six neighbors of each voxel is encoded by edges connecting the corresponding vertices within the map. The background in which the object is embedded is represented by an additional vertex. Each vertex on the border of the grid is connected to this background-vertex by one edge.

Fig. 5.1 shows an example of such a conversion. A $3 \times 3 \times 3$ object is transformed into a $3 \times 3 \times 3$ grid-like combinatorial map. The background vertex is not shown in this figure for the clarity of this example.

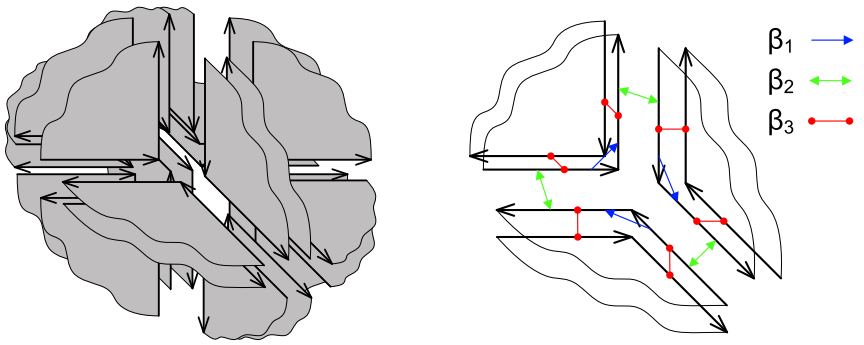


a) a 3x3x3 object b) 3D combinatorial map of the object

Figure 5.1: Example of a grid-like 3D combinatorial map representing volumetric data. A $3 \times 3 \times 3$ object (a) is converted into a combinatorial map (b). Each voxel is encoded by a vertex in the grid. Neighboring voxels are connected by an edge to encode this adjacency.

Each vertex that does not lie on the border of such a grid is incident to 6 edges, 12 faces and 8 volumes. Fig 5.2a shows the configuration of such a vertex within the grid-like combinatorial map. Each vertex is represented by 24 darts when using a 3D combinatorial map $G = (D, \beta_1, \beta_2, \beta_3)$. Two darts are needed for each face: one for each edge that bounds the face and is incident to the vertex.

In Fig. 5.2b the correct β_1-, β_2- and β_3-connections for such a grid can be seen in the example of one corner of a volume being incident to the vertex. β_3 is set to the dart belonging to the same face but the opposite vertex of the edge. β_2 connects the two darts belonging to two subsequent faces around an edge and the opposite vertices of the edge. Finally the β_1 successor is set



a) one vertex within the initial map b) β_1-, β_2- and β_3-connections

Figure 5.2: Detailed view of a vertex within the initial 3D combinatorial map. (a) shows the darts needed to encode the 6 edges, 12 faces and 8 volumes each vertex is incident to. (b) shows the β_1-, β_2- and β_3-connections within the grid in the example of 3 faces defining the corner of one volume the vertex is incident to.

to the dart belonging to the next edge around the border of the face.

5.1.1 Size of the Initial Combinatorial Map

Since the number of darts needed to represent each vertex within the grid is known as well as the number of edges, faces and volumes each vertex is incident to, the size of such a grid-like map depends only on the size of the volumetric data. Thus the size of the initial map for a $n_1 \times n_2 \times n_3$ object can be calculated using only n_1, n_2 and n_3.

The number of such vertices is equal to the number of elements in the $n_1 \times n_2 \times n_3$ matrix plus one additional vertex representing the background. Therefore the number of vertices $n_{ve}(G)$ for a combinatorial map G representing a $n_1 \times n_2 \times n_3$ object is defined by

$$n_{ve}(G) = n_1 * n_2 * n_3 + 1.$$

The number of darts $n_d(G)$ cannot be directly calculated by multiplying the number of vertices by 24 (the number of darts needed to represent a vertex within the grid) as the number of darts is different for vertices on the faces, edges and corners of the grid as well as for the background vertex. So

the total number of darts is determined by calculating the number of darts for each type of vertex and adding up these numbers.

The number of darts of a vertex that does not lie on the border of the grid is 24. There are $(n_1 - 2) * (n_2 - 2) * (n_3 - 2)$ such vertices. The number of darts for a vertex on a face of the grid is also 24. There are $2 * (n_1 - 2) * (n_2 - 2) + 2 * (n_1 - 2) * (n_3 - 2) + 2 * (n_2 - 2) * (n_3 - 2)$ such vertices in a grid. A vertex on an edge of the grid is only incident to 5 edges and therefore consists of 18 darts. The number of such vertices on an edge of the grid is calculated by $4 * ((n_1 - 2) + (n_2 - 2) + (n_3 - 2))$. Finally there are 8 vertices on the corners of the grid which consist of 12 darts.

The number of darts of the background vertex can be calculated in similar way. It consists of 4 darts for each vertex on a face, 4 darts for each vertex on an edge and 3 darts for each vertex on a corner of the grid. Thus the total number of darts $n_d(G)$ in an initial grid is given by

$$\begin{aligned} n_d(G) &= 24 * (n_1 - 2) * (n_2 - 2) * (n_3 - 2) \\ &\quad +(24 + 4) * [2 * (n_1 - 2) * (n_2 - 2) \\ &\qquad\qquad +2 * (n_1 - 2) * (n_3 - 2) \\ &\qquad\qquad +2 * (n_2 - 2) * (n_3 - 2)] \\ &\quad +(18 + 4) * 4 * [(n_1 - 2) + (n_2 - 2) + (n_3 - 2)] \\ &\quad +(12 + 3) * 8. \end{aligned}$$

Additionally, an upper bound can be specified for $n_d(G)$ approximating the real number of darts. This is done by calculating the number of vertices needed for a $(n_1 + 1) \times (n_2 + 1) \times (n_3 + 1)$ grid without a background vertex and multiplying this number by 24.

$$n_d(G) \approx (n_1 + 1) * (n_2 + 1) * (n_3 + 1) * 24$$

The number of edges $n_e(G)$ is determined by calculating the number of edges for a grid in which each vertex is incident to 6 edges and then subtracting 1 edge for each vertex on an edge of the grid and 2 edges for an edge on the corner of the grid.

$$
\begin{aligned}
n_e(G) &= (n_1+1)*n_2*n_3 \\
&\quad +n_1*(n_2+1)*n_3 \\
&\quad +n_1*n_2*(n_3+1) \\
&\quad -4*(n_1+n_2+n_3)+8
\end{aligned}
$$

The number of faces $n_f(G)$ and the number of volumes $n_{vo}(G)$ are calculated in the same way.

$$
\begin{aligned}
n_f(G) &= (n_1+1)*(n_2+1)*n_3 \\
&\quad +(n_1+1)*n_2*(n_3+1) \\
&\quad +n_1*(n_2+1)*(n_3+1) \\
&\quad -4*(n_1+n_2+n_3) \\
&\quad -4*(n_1-1+n_2-1+n_3-1) \\
n_{vo}(G) &= (n_1+1)*(n_2+1)*(n_3+1) \\
&\quad -4*(n_1-1+n_2-1+n_3-1)-8
\end{aligned}
$$

The number of edges $n_e(G)$, the number of faces $n_f(G)$, and the number of volumes $n_{vo}(G)$ can be approximated in a similar way, namely by the number of edges, faces and volumes within a $n_1 \times n_2 \times n_3$ grid (without a background vertex) where each vertex is incident to 6 edges, 12 faces and 8 volumes. Thus the subtraction of edges, faces or volumes for vertices on the border of the grid is removed from the original formula.

$$
\begin{aligned}
n_e(G) &\approx (n_1+1)*n_2*n_3 \\
&\quad +n_1*(n_2+1)*n_3 \\
&\quad +n_1*n_2*(n_3+1) \\
n_f(G) &\approx (n_1+1)*(n_2+1)*n_3 \\
&\quad +(n_1+1)*n_2*(n_3+1) \\
&\quad +n_1*(n_2+1)*(n_3+1) \\
n_{vo}(G) &\approx (n_1+1)*(n_2+1)*(n_3+1)
\end{aligned}
$$

Table 5.1 contains the exact values of n_d, n_{ve}, n_e, n_f and n_{vo} for initial maps of different sizes. As can be seen in the fourth row approximately 24 million darts are needed to represent a 100x100x100 object. The grid for one second of a movie with 25 frames per second and a resolution of 640x480 consists of about 185 million darts.

These numbers illustrate the importance of reducing such initial maps to their minimal, topologically equivalent form.

size	n_d	n_{ve}	n_e	n_f	n_{vo}
2x2x2	120	9	20	18	7
3x3x3	576	28	80	84	32
10x10x10	25.272	1.001	3.188	3.402	1.215
100x100x100	24.228.072	1.000.001	3.028.808	3.057.912	1.029.105
640x480x25	186.955.872	7.680.001	23.370.628	23.702.397	8.011.770

Table 5.1: Number of darts, vertices, edges, faces and volumes in the initial grid-like maps of different sizes.

The approximated numbers of n_d, n_{ve}, n_e, n_f and n_{vo} for the same maps are listed in Table 5.2. As can be seen, the error of the approximation is large for small maps (n_d is overestimated by a factor of 5,5 for the $2 \times 2 \times 2$ map), but becomes less significant for larger maps.

size	n_d	n_{ve}	n_e	n_f	n_{vo}
2x2x2	648	9	36	54	27
3x3x3	1.536	28	108	144	64
10x10x10	31.944	1.001	3.300	3.630	1.331
100x100x100	24.727.224	1.000.001	3.030.000	3.060.300	1.030.301
640x480x25	192.392.304	7.680.001	23.375.200	23.711.545	8.016.346

Table 5.2: Approximated number of darts, vertices, edges, faces and volumes in the initial grid-like maps of different sizes.

5.2 Building the Pyramid

This section describes the process of building a 3D combinatorial pyramid using the contraction and removal operations defined in the previous chapter.

The pyramid is built level by level. Each level is a reduced, but topologically equivalent map based on the previous level. Once a level is reached that cannot be simplified anymore, the pyramid is complete. Thus the top level contains the minimal combinatorial map which represents the original data topologically correct [Grasset-Simon et al., 2005, Brun and Kropatsch, 1999b].

The simplification of each level is divided into two parts:

1. Merging of adjacent voxels that belong to the same component
2. Eliminating redundant i-cells

The first step will be referred to as an "algorithm step" throughout this report, while the second step will be called a "simplification step".

While it is possible to apply each contraction or removal operation immediately to a map, [Kropatsch, 1995] suggests the use of contraction and removal kernels as a more effective method. This concept can be used in the following way for building 3D combinatorial pyramids:

During each step, i-cells are selected for contraction (removal) and added to a contraction (removal) kernel. After a step has selected all i-cells, this contraction (removal) kernel is applied to the map to create a temporary combinatorial map. The next step will then select other i-cells for contraction (removal) based on the temporary map and will again build a contraction (removal) kernel. Once all steps have been completed for a level, the last temporary map built this way will be used as the next level of the pyramid. All other temporary maps are discarded.

5.2.1 Algorithm Step

Adjacent voxels are connected by an edge within the combinatorial map. Merging two such voxels can therefore be achieved by contracting the connecting edge.

```
comb_map *algorithm_step(comb_map *map) {
  // contraction kernel to hold all selected edges
  contraction_kernel    c_kernel;

  // Create an Iterator that will process all edges
  // present in the map
  edge_iterator    edge_it = map->edges.begin();

  // iterate through all edges
  for(edge_it; edge_it!=map->edges.end(); edge_it++) {
    // retrieve the values of both vertices
    value value1 = edge_it->value();
    value value2 = edge_it->get_next_vertex()->value();

    // algorithm specific decision to contract an edge
    // e.g. a connected component algorithm compares
    // the values of two vertices
    if ( value1 == value2 ) {

      // if the edge was selected by the algorithm
      // the other conditions must be checked as well
      // the conditions are checked with respect to the
      // edges already in the contraction kernel.
      if ( edge_it->validate(EDGE_CONTRACTION, c_kernel) ) {

        // and add the edge to the kernel if the
        // conditions were met
        // this function also checks that the
        // edges form a forest
        c_kernel->add( edge_it );
      }
    }
  }

  // finally create a temporary map by applying the
  // contraction kernel if it is not empty
  if ( ! c_kernel->empty() ) {
    comb_map *tmp_map = new comb_map(map, c_kernel);
  } else {
    // otherwise return the original map
    tmp_map = map;
  }

  return tmp_map;
}
```

Listing 5.1: Algorithm step

The selection of edges to be contracted in this step depends solely on the algorithm used. E.g. when implementing a connected component algorithm based on labeled voxels, the algorithm will select all edges connecting two vertices that represent voxels with the same label.

Additionally, the conditions presented in section 4.1 have to be validated to ensure that the selected edges are eligible for contraction. Because a kernel is used to apply multiple edge contractions simultaneously to the combinatorial map, it is not sufficient to check these conditions for each edge independently. Instead, the conditions must be validated for the candidate edge with respect to the edges already in the kernel and it must be ensured that adding the candidate edge does not invalidate the conditions for an edge that is already part of the kernel.

Condition 1 (Dual-Degree): Contracting an edge will not change the dual-degree of any other edge of the map. This condition can therefore be validated locally for each edge.

Condition 2 (Bounding and Incidence Relations): This condition must be validated with respect to the edges already in the kernel. To ensure that the contraction of a candidate edge will remove only the edge itself and one of the two bounding vertices, all incident i-cells are examined. If any of these incident i-cells consists only of darts that either belong to the candidate edge or that are already in the contraction kernel, then this i-cell would be removed as a side effect and therefore the operation is disallowed.

Condition 3 (Self-Loops): This condition must be validated with respect to the edges already in the kernel. [Brun and Kropatsch, 1999a] show that the edges in the kernel must be a forest to validate this condition.

Condition 4 (Common $(i-2)$-cells): This condition does not apply to edges as it is only defined for i-cells with $i \geq 2$.

Condition 5 (Background vertex): It is sufficient to validate this condition for each edge locally, as the contraction of an edge does not change which edges are connected to the background vertex.

Listing 5.1 gives a C++ implementation of the algorithm step. The function `algorithm_step` takes an existing combinatorial map as its argument and returns the temporary map created after applying the contraction kernel. The contraction kernel is built by first selecting all candidate edges by the algorithm. If they also fulfill all conditions defined for an edge contraction, they are added to the kernel.

5.2.2 Simplification Step

Simplification steps are used to eliminate redundant i-cells from a given combinatorial map. For 2D combinatorial maps [Brun and Kropatsch, 1999a] showed that such redundant i-cells are either parallel edges or edges that are empty self-loops. Empty self-loops are eliminated by an edge removal while parallel edges can either be eliminated by an edge removal or by a face contraction.

When extending this concept to 3D, parallel faces and empty face self-loops are also considered redundant cells. Therefore parallel cells and empty self-loops are defined for cells of any dimension.

Definition 19 (Parallel i-Cell) *Given a combinatorial map G and an i-cell $c \in G$. c is a parallel i-cell if there is an i-cell $c' \in G$ such that*

- $\exists$ *$(i+1)$-cell $c_i \in G$, $\bar{\delta}(c_i) = 2$, c_i is bounded by c and c';*
- *$B_c = B_{c'}$ where B_c is the set of $(i-1)$-cells bounding c and $B_{c'}$ is the set of $(i-1)$-cells bounding c'.*

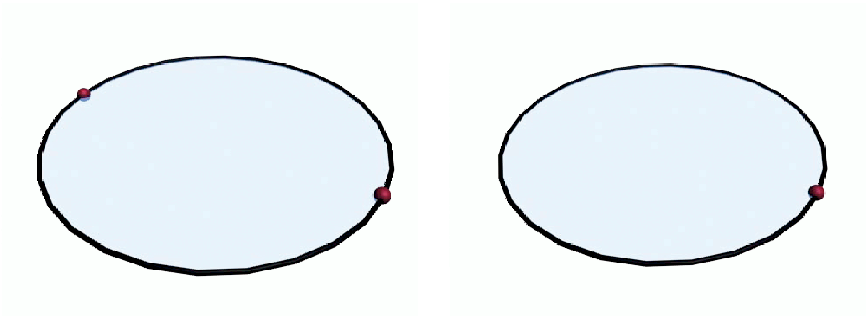
Definition 20 (Empty i-Cell Self-Loop) *Given a combinatorial map G and an i-cell $c \in G$. c is an empty self-loop if there is an $(i+1)$-cell $c_i \in G, \bar{\delta}(c_i) = 1$ such that*

- *c bounds c_i;*
- *$\bar{\delta}(c) = 2$;*
- *c is bounded by exactly one $(i-1)$-cell $c_j \in G$.*

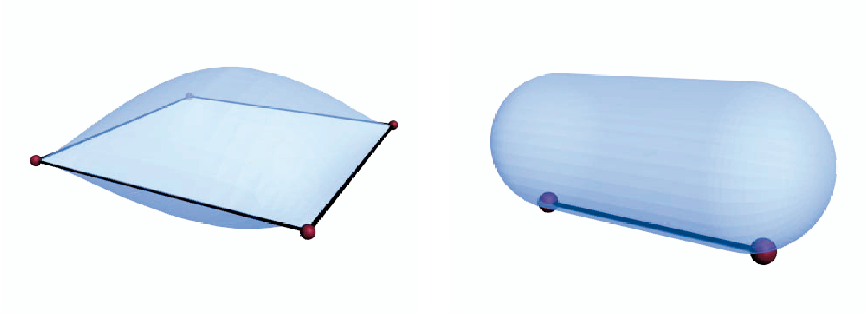
Using these two definitions a redundant i-cell is defined as an i-cell that is either a parallel i-cell or an empty i-cell self-loop:

Definition 21 (Redundant i-Cell) *Given a combinatorial map G and an i-cell $c \in G$. c is a redundant i-cell if it is either a parallel i-cell or an empty i-cell self-loop.*

Based on these definitions, four kinds of redundant i-cells exist in a 3D combinatorial map: parallel edges, parallel faces, empty edge self-loops and empty face self-loops. Fig 5.3 shows examples of these four types of redundant i-cells.



a) parallel edge b) empty edge self-loop

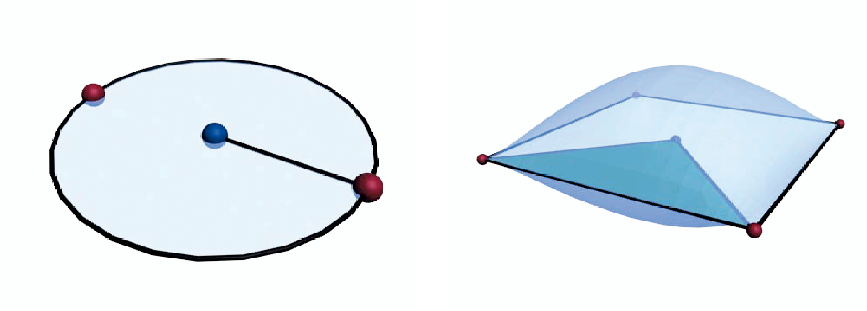


c) parallel face d) empty face self-loop

Figure 5.3: Examples of redundant i-cells: an empty face bounded by parallel edges (a), an empty face bounded by an edge self-loop (b), an empty volume bounded by parallel faces (c), and an empty volume bounded by a face self-loop (d).

For reasons of completness examples of non-empty cells are shown as well even though they are not considered redundant cells according to definition 21. Non-empty parallel i-cells can be seen in Fig. 5.4 and examples of non-empty self-loops are shown in chapter 6 in Fig. 6.5.

Parallel edges and empty edge self-loops are eliminated by the contraction of a face and the removal of an edge as introduced by [Brun and Kropatsch, 1999a]. To deal with parallel faces and empty face self-loops two additional operations are used. The contraction of a volume is introduced to deal with parallel faces and the removal of a face is used to remove faces that form an empty self-loop. Therefore the four operations edge removal, face removal, face contraction, and volume contraction are used to simplify a combinatorial map. Table 5.3 contains an overview of these four operations.



a) non-empty parallel edges d) non-empty face self-loop

Figure 5.4: non-empty parallel i-cells: non-empty parallel edges (a) and non-empty parallel faces (b).

operation	i-cell eliminated
edge removal	empty edge self-loops
face contraction	parallel edges
face removal	empty face self-loops
volume contraction	parallel faces

Table 5.3: Operations used to eliminate different kinds of redundant i-cells in a 3D combinatorial map.

5.2.3 Simplification with Contraction Operations

Parallel edges and parallel faces can both be identified using the dual-degree of an i-cell. The dual-degree of an i-cell that is bounded by parallel $(i-1)$-cells equals two. This means that the same algorithm can be used when simplifying a map using either face contractions or volume contractions.

Again it must be ensured that the conditions for a contraction are valid with respect to the i-cells already in the contraction kernel.

Condition 1 (Dual-Degree): Contracting an i-cell will not change the dual-degree of any other i-cell of the map with the same dimension. This condition can therefore be validated locally for each i-cell.

Condition 2 (Bounding and Incidence Relations): This condition must be validated with respect to the i-cells already in the kernel. To ensure that the contraction of a candidate i-cell will remove only the i-cell itself and one of the two bounding $(i-1)$-cells, all incident cells are examined. If any of these incident cells consists only of darts that either belong to the candidate i-cell

or that are already in the contraction kernel, then this cell would be removed as a side effect and therefore the operation is disallowed.

Condition 3 (Self-Loops): This condition must be validated with respect to the i-cells already in the kernel. The the i-cells in the kernel must be a forest to validate this condition [Brun and Kropatsch, 1999a].

Condition 4 (Common $(i-2)$-cells): This condition can be validated locally as the contraction of an i-cell will not remove any other cell but the i-cell itself and one of the two bounding $(i-1)$-cells. Thus the $(i-2)$-cells bounding a cell will not be changed by the contraction of the i-cells in the kernel.

Condition 5 (Background vertex): This condition is only relevant for edges and does not need to be checked for an i-cell of a higher dimension.

A C++ version of a function `simplify_by_contraction` implementing such an algorithm is displayed in listing 5.2. This function takes the combinatorial map that is to be reduced and the type of i-cells as its arguments and returns a simplified map. First, the i-cells that are candidates for a contraction are selected by iterating through all i-cells of the map and choosing those with a dual-degree of two. If these candidates meet the other conditions for a contraction operation, they are added to the contraction kernel. After all i-cells have been processed this way, the contraction kernel is applied to the map to create the reduced map.

5.2.4 Simplification with Removal Operations

Both removal operations are used to eliminate empty self-loops from the combinatorial map. Self-loops are defined as i-cells that are bounded by the same $(i-1)$-cell twice. E.g. an edge with the same vertex on both endpoints or a volume bounded by a single face that itself is bounded by one edge twice.

This means that the dual-degree of an empty self-loop equals two, being bounded by one $(i-1)$-cell twice. Additionally, the degree of the i-cell must also equal two (see condition 6). E.g. an edge cannot be removed if it is incident to three or more faces, even if it is an empty self-loop.

Because a removal kernel is used it must be ensured that the conditions are valid with respect to the i-cells already in the kernel when adding an i-cell. This is done in an analogous way to the contraction operation and the use of a contraction kernel.

```
comb_map *simplify_by_contraction(comb_map *map, int i) {
  // contraction kernel to hold all selected i-cells
  contraction_kernel    c_kernel;

  // Create an Iterator that will process all cells of
  // dimension i present in the map
  cell_iterator     cell_it = map->cells(i).begin();

  // iterate through all i-cells
  for(cell_it; cell_it!=map->cells(i).end(); cell_it++) {
    // select those i-cells that have a dual_degree
    // of 2 as candidates for a contraction
    if ( cell_it->dual_degree() == 2 ) {

      // if the cell was selected by the algorithm
      // the other conditions for a contraction of
      // a cell of dimension i must be checked as well.
      // the conditions are checked with respect to the
      // cells already in the kernel
      if ( cell_it->validate(CONTRACTION,i,c_kernel) ) {

        // and add the cell to the kernel if the
        // conditions were met
        // this function also ensures that the cells
        // in the kernel form a forest
        c_kernel->add( cell_it );
      }
    }
  }

  // finally create a temporary map by applying the
  // contraction kernel if it is not empty
  if ( ! c_kernel->empty() ) {
    comb_map *tmp_map = new comb_map(map, c_kernel);
  } else {
    // otherwise return the original map
    tmp_map = map;
  }

  return tmp_map;
}
```

Listing 5.2: Simplification step using contraction operations

Condition 6 (Degree): Removing an i-cell will not change the degree of any other i-cell of the map with the same dimension. This condition can therefore be validated locally for each i-cell.

Condition 7 (Bounding and Incidence Relations): This condition must be validated with respect to the i-cells already in the kernel. To ensure that the removal of a candidate i-cell will remove only the i-cell itself and one of the two incident $(i+1)$-cells, all incident cells are examined. If any of these incident cells consists only of darts that either belong to the candidate i-cell or that are already in the removal kernel, then this cell would be removed as a side effect and therefore the operation is disallowed.

Condition 8 (Bridges): This condition must be validated with respect to the edges already in the kernel. [Brun and Kropatsch, 1999a] show that it is sufficient to ensure that the dual-cells of the cells in the removal kernel are a forest.

Condition 9 (Common $(i+2)$-cells): This condition can be validated locally as the removal of an i-cell will not remove any other cell but the i-cell itself and one of the two incident $(i+1)$-cells. Thus the $(i+2)$-cells incident to a cell will not be changed by the removal of the i-cells in the kernel.

Condition 10 (Connected Vertices): This condition must be checked with respect to the i-cells already in the kernel, but it is sufficient to validate it in the case of face removals. When testing whether a vertex is still connected after the removal of a face, only those darts may be considered that are not already in the kernel. This means that each surviving dart of the vertex must be reachable from every other surviving dart of the vertex using any combination of $\beta_2\beta_1$ and $\beta_3\beta_1$ without traversing a dart that belongs to the removed face or that is already in the kernel.

The algorithm for simplifying a combinatorial map with removal operations can then be designed in a similar way to the simplification using contraction operations. This is done by selecting empty self-loops instead of i-cells bounded by parallel $(i-1)$-cells and using a removal kernel instead of a contraction kernel.

Listing 5.3 contains a C++ version of this algorithm. The function `simplify_by_removal` takes a combinatorial map and the type of i-cells to be removed as its arguments and returns a combinatorial map in which all removable and empty self-loops have been eliminated. Again, candidate i-cells are selected by iterating through all i-cells of the map. If empty self-loops are detected, they are tested against the other conditions for the removal of

```
comb_map *simplify_by_removal(comb_map *map, int i) {
  // removal kernel to hold all selected i-cells
  removal_kernel     r_kernel;

  // Create an Iterator that will process all cells of
  // dimension i present in the map
  cell_iterator      cell_it = map->cells(i).begin();

  // iterate through all i-cells
  for(cell_it; cell_it!=map->cells(i).end(); cell_it++) {
    // select those i-cells that have a dual_degree
    // of 2, a degree of 2 and that are bounded by
    // the same (i-1)-cell twice
    if ( ( cell_it->dual_degree() == 2 )
      && ( cell_it->degree() == 2 )
      && ( cell_it->bounding_cells_with_same_label() )) {

      // if the cell was selected by the algorithm
      // the other conditions for a removal of a
      // cell of dimension i must be checked as well.
      // the conditions are checked with respect to the
      // cells already in the kernel
      if ( cell_it->validate(REMOVAL,i,r_kernel) ) {

        // and add the cell to the kernel if the
        // conditions were met
        // this function also checks that the dual-cells
        // of the cells in the kernel are a forest
        r_kernel->add( cell_it );
      }
    }
  }

  // finally create a temporary map by applying the
  // removal kernel if it is not empty
  if ( ! r_kernel->empty() ) {
    comb_map *tmp_map = new comb_map(map, r_kernel);
  } else {
    // otherwise return the original map
    tmp_map = map;
  }

  return tmp_map;
}
```

Listing 5.3: Simplification step using removal operations

an i-cell and then added to the removal kernel. Finally, the removal kernel is applied to the map and the resulting combinatorial map is returned.

5.2.5 Building the next level

Using the algorithm step and the simplification steps introduced in the previous sections, an algorithm for building the next level of a pyramid can be designed.

1. Contract all edges that connect vertices belonging to the same component (algorithm step)
2. Contract all faces bounded by parallel edges (simplification step)
3. Contract all volumes bounded by parallel faces (simplification step)
4. Remove all empty edge self-loops (simplification step)
5. Remove all empty face self-loops (simplification step)

The algorithm step will only be executed once when building a new level. Steps 2-5 will be repeated until no more candidates for a simplification can be found within the map. This way it is ensured that each new level will not contain any redundant cells.

Listing 5.4 shows a C++ implementation of this algorithm. The function `build_next_level` will create the next level of the pyramid based on the map it receives as its argument. This is done by applying all steps as described above. Each step produces a temporary map that is discarded after the next step has been executed. The last of these temporary maps is kept and returned as the next level of the pyramid.

```
comb_map *build_next_level(comb_map *map) {
  // temporary maps holding the intermediate maps
  comb_map *tmp_map, *tmp_map2;

  // the algorithm_step is applied once
  tmp_map = algorithm_step(map);

  // flag set to true if a simplification was performed
  bool simplified;

  // apply the simplification steps until no more
  // candidates are found
  do {
    simplified = false;

    // simplify map using face and volume contractions
    for (int=2;i<=3;i++) {
      tmp_map2 = simplify_by_contraction(tmp_map, i);

      //check whether the map has been changed
      if (tmp_map2 != tmp_map) {
        // set flag and discard old temporary map
        simplified = true;
        delete (tmp_map);
        tmp_map = tmp_map2;
      }
    }

    // simplify map using edge and face removals
    for (int=1;i<=2;i++) {
      tmp_map2 = simplify_by_removal(tmp_map, i);

      //check whether the map has been changed
      if (tmp_map2 != tmp_map) {

        // set flag and discard old temporary map
        simplified = true;
        delete (tmp_map);
        tmp_map = tmp_map2;
      }
    }
  } while (simplified);

  // return the last temporary map as the
  // next level of the pyramid
  return tmp_map;
}
```

Listing 5.4: Building the next level of a pyramid

5.3 Receptive Fields

Once a combinatorial pyramid has been built, connecting walks can be used to calculate the receptive field of any dart within the pyramid [Brun and Kropatsch, 2003].

The concept of connecting walks has been introduced by [Brun and Kropatsch, 1999b] for 2D combinatorial pyramids. It is used when creating a new combinatorial map G' by applying multiple contraction or removal operations simultaneously onto a combinatorial map G using a contraction- or removal-kernel. Each surviving dart is associated with another surviving dart by traversing some non-surviving darts. This traversal is non-ambiguous and is used to find the β_n-successors for each dart in G'.

At the same time, a connecting walk associates one surviving dart unambiguously with each non-surviving dart. Each non-surviving dart traversed during a connecting walk is assigned to the surviving dart at the end of the connecting walk. This mapping of non-surviving darts to surviving darts is used by [Brun and Kropatsch, 2003] to calculate the receptive field of darts within a combinatorial pyramid.

[Damiand and Lienhardt, 2003] extend this concept to any dimension and multiple operations. The dimension i of a non-surviving cell and the type of operation (contraction or removal) that removed the i-cell from the map is saved for each dart of the cell. This information is then used to calculate the connecting walk for each dart d and permutation β_n, using the following algorithm.

operation	dimension of the cell	permutation	sequence
contraction	1	β_1	$\beta_1^p(d)$
contraction	2	β_2	$\beta_2(\beta_1(d))$
contraction	3	β_3	$\beta_3(\beta_2(d))$
removal	1	β_1	$\beta_1(\beta_2(d))$
removal	2	β_2	$\beta_2(\beta_3(d))$

Table 5.4: Permutation sequences for the construction of a connecting walk.

Starting from a surviving dart d the β_n successor is checked. If it is a surviving dart, the connecting walk is complete. If it is a non-surviving dart, then the information about the dimension of the i-cell and the type of operation is used to select the correct permutation sequence from Table 5.4

```
dart_type *surviving_dart(dart_type *dart,int perm_index){

  // retrieve the successor of the permutation specified
  dart_type *tmp_dart = dart->permutation(perm_index);

  // check whether it was removed or contracted
  // these functions return the dimension of the
  // removed/contracted cell or NOT_REMOVED,
  // NOT_CONTRACTED otherwise
  int removed    = tmp_dart->removed_by_cell();
  int contracted = tmp_dart->contracted_by_cell();

  // Apply the correct permutation sequences
  // until a surviving dart is returned
  while (removed!=NOT_REMOVED
         || contracted!=NOT_CONTRACTED) {

    // call removal_successor or contraction_successor
    // recursively depending on the operation
    if (removed!=NOT_REMOVED)
      tmp_dart = removal_successor(tmp_dart, removed);
    else
      tmp_dart = contraction_successor(tmp_dart,
                                       contracted);

    removed    = tmp_dart->removed_by_cell();
    contracted = tmp_dart->contracted_by_cell();
  }

  return tmp_dart;
}
```

Listing 5.5: Retrieving the surviving dart using connecting walks

(these permutation sequences correspond to the permutations defined for each operation in chapter 4). The connecting walk is completed if applying these permutations returns a surviving dart. Otherwise, the procedure is repeated until a surviving dart is found.

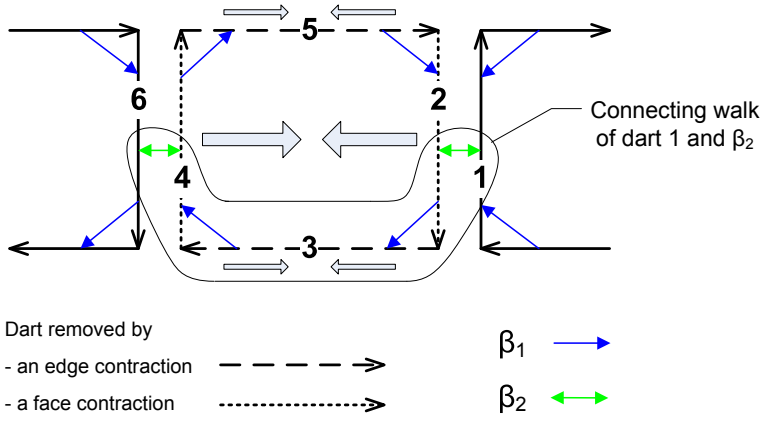
Since the permutation sequences in Table 5.4 consist of more than one subsequent permutation, the darts have to be checked after the application of each permutation. If such an intermediate dart has been removed from the map due to a different operation, the correct sequence for this operation has to be applied before continuing with the original sequence. This leads to the recursive algorithm presented in listings 5.5 & 5.6.

```
dart_type *removal_successor(dart_type *dart,
                             int dimension) {

  // apply the correct sequence for removed i-cells
  // based on the dimension of the cell
  switch ( dimension ) {
    case 1:
      // apply the first permutation
      tmp_dart = tmp_dart->permutation(BETA_2);

      // check if the returned dart was removed due to
      // the same operation and dimension
      // these functions return the dimension of the
      // removed/contracted cell or NOT_REMOVED,
      // NOT_CONTRACTED otherwise
      int removed    = tmp_dart->removed_by_cell();
      int contracted = tmp_dart->contracted_by_cell();

      // if not, the surviving_dart function is called
      // recursively until the second dart of removal
      // operation for this dimension is returned
      while (removed != 1) {
        if (removed!=NOT_REMOVED)
          tmp_dart = removal_successor(tmp_dart, removed);
        else
          tmp_dart = contraction_successor(tmp_dart,
                                           contracted);

        removed    = tmp_dart->removed_by_cell();
        contracted = tmp_dart->contracted_by_cell();
      }

      // apply the second permutation and return the dart
      return (tmp_dart->permutation(BETA_1));
    case 2:
      ...
  }
}
```

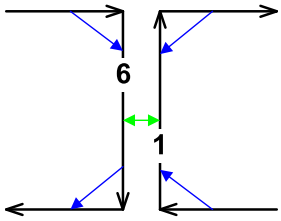
Listing 5.6: Recursively calculating the successor for a removal operation

Listing 5.5 defines a function `surviving_dart` that will return the surviving dart for a specified dart and permutation β_n by implementing the connecting walk. Depending on the contraction or removal operations encountered, the recursive functions `removal_successor` and `contraction_successor` are called until a surviving dart is found.

`removal_successor` and `contraction_successor` will apply the permutation sequence from Table 5.4 depending on the dimension of the removed or contracted cell. `removal_successor` and `contraction_successor` are called recursively if the sequence contains a dart that was eliminated by a different operation. Listing 5.6 shows the implementation of the function `removal_successor` for dimension 1.



a) 2 edges and 1 face are selected for contraction



b) configuration after the contractions

Figure 5.5: Connecting walk of a dart. (a) shows a map that is reduced by two edge contraction (darts 3 & 5) and one face contraction (darts 2 & 4). The β_2 successor of dart 1 is set to dart 6 (b) using the connecting walk.

Fig. 5.5 shows an example of such a connecting walk. The original configuration has been reduced using an edge contraction and a face contraction. The β_2 permutation of the dart with index 1 is set to the surviving dart with index 6 using the connecting walk. The original β_2 successor - dart 2 - was removed from the map by the face contraction. Therefore the permutation sequence $\beta_2(\beta_1(d))$ is used to construct the connecting walk. By applying β_1 on dart 2, the dart with index 3 is obtained. Since this dart was removed

by an edge contraction the second permutation from the $\beta_2(\beta_1(d))$ sequence is not immediately applied. Instead, the sequence of an edge contraction - $\beta_1^n(d)$ is used first, which leads to the dart with index 4. As this dart was removed by the face contraction, β_2 (the second permutation of the $\beta_2(\beta_1(d))$ sequence) can be applied now. The resulting dart 6 is a surviving dart. Thus the connecting walk is complete and the β_2 successor of dart 1 is set to dart 6.

Chapter 6

Configurations and Topological Relations

The previous chapter described the process of building a combinatorial pyramid with the operations defined in chapter 4.

In the first section of this chapter, minimal combinatorial maps obtained by these operations are discussed. It will be shown that these operations are suitable to create minimal maps where each connected component is reduced to a single vertex if the original configuration does not contain face self-loops. The second and third section studies the properties of minimal configurations and provides the means for interpreting them. Finally the last section will examine configurations that cannot be encoded by a 3D combinatorial map.

6.1 Minimal 3D Combinatorial Maps

It will be shown in this section that the operations introduced in chapter 4 together with their 10 conditions will reduce a 3D combinatorial map to its minimal form if the map does not contain face self-loops. For this purpose the definition of a minimal 3D combinatorial map is introduced. Within this report, a 3D combinatorial map is considered to be minimal, if each connected 3D component is represented by exactly one vertex.

Definition 22 (Minimal 3D Combinatorial Map) *Given a 3D combinatorial map G. G is a minimal 3D combinatorial map if every connected 3D component of G is encoded by exactly one vertex of G.*

Theorem 6.1 *Given a combinatorial map $G = (D, \beta_1, \beta_2, \beta_3)$, such that $\nexists$ 2-cell $f \in G, f$ is a face self-loop. G can be reduced to an equivalent minimal 3D combinatorial map.*

This theorem uses the definition of a face self-loop (see definitions 20 and 24).

Proof: Given a combinatorial map $G = (D, \beta_1, \beta_2, \beta_3)$, such that $\nexists$ 2-cell $f \in G, f$ is a face self-loop.

To prove that G can be reduced to an equivalent minimal 3D combinatorial map, it will shown that a connected component $k \in G$ can be reduced to a single vertex using edge contractions. This result can be directly applied to every connected component of G.

To show that a connected component $k \in G$ can be reduced to single vertex, two cases are considered:

- Let there be only one vertex in G with the label of k. In this case k is already minimal.
- Let there be n vertices in G with the label of k. In this case there must be an edge connecting two vertices with the label of k, since k is a connected component: Let K be the set of all vertices in G with the label of k. $\exists$ 0-cell $v_1, v_2 \in K, v_1 \neq v_2; \exists$ 1-cell $e \in G$, such that v_1 and v_2 are incident to e.

  It will be shown, that the number of vertices with the label of k can be reduced to $n - 1$ by the contraction of e.

  It will be shown, the the 5 conditions for the contraction of e are fulfilled if G does not contain a face self-loop.

  Further it will be shown that the resulting map after the contraction $G' = G/D(e)$ will still not contain a face self-loop. Thus the procedure can be repeated until k consists of only vertex.

  $G' = G/D(e)$ is an equivalent map as the contraction of an edge will not change the connected component adjacency relations or inclusion relations.

The contraction of an edge eliminates the edge and merges the two incident vertices. Thus the contraction of e will merge v_1 and v_2, reducing the number of vertices with the label of k by one and the number of vertices with the label of k is $n - 1$.

The contraction of e is allowed, if the 5 conditions for a contraction operation are fulfilled. It can easily be seen, that conditions 1, 3, 4, and 5 are fulfilled:

- Condition 1 (Dual Degree): $\bar{\delta}(e) = 2$ is fulfilled as the dual-degree of an edge is always two.
- Condition 3 (Self-Loops): e cannot be a self-loop, since $v_1 \neq v_2$. Therefore this condition is fulfilled.
- Condition 4 (Common $(i-2)$-cells): This condition only applies for the contraction of a i-cell, $i \geq 2$. Thus the 1-cell e fulfills this condition.
- Condition 5 (Background vertex): This condition is fulfilled since e is incident to v_1 and v_2.

To fulfill the Condition 2 (Bounding and Incidence Relations) it must be shown, that there is no cell $c \in G, c \neq e, c \neq v_1, c \neq v_2$, such that $D(c) \subseteq D(e)$. For this condition the 4 cases are considered:

- Let c be a 0-cell. Since $c \neq v_1, c \neq v_2$ and v_1 and v_2 are incident to e, there cannot be another 0-cell also being incident to e. If c is not incident to e it follows that $D(c) \not\subseteq D(e)$.
- Let c be a 1-cell. Since $c \neq e$ and since a 1-cell cannot be incident to another 1-cell it follows that $D(c) \not\subseteq D(e)$.
- Let c be a 2-cell. If $D(c) \subseteq D(e)$, then e must be the only edge incident to c. Thus c is only bounded by e. Since e is not a self-loop, this can only be the case if c is a face self-loop. As G does not contain face self-loops, the condition is fulfilled.
- Let c be a 3-cell. If $D(c) \subseteq D(e)$, then e must be the only edge incident to c. Thus c is only bounded by a face that again is only bounded by e. Since e is not a self-loop, this can only be the case if c is bounded by a face self-loop. Since G does contain face self-loops, this cannot be the case and the condition is fulfilled.

It remains to be shown that the the combinatorial map $G' = G/D(e)$ created by the contraction of e will also not contain a face self-loops. G does not contain face self-loops. A face self-loop is a face that is incident to one edge twice. Since the contraction of an edge cannot change the number of

times that a face is incident to an edge, the contraction of e cannot create a face self-loop in G. $\square$

Note that only edge contractions are used to obtain a minimal 3D combinatorial map. The other operations introduced in this report can then be applied to the minimal 3D combinatorial map to eliminate redundant elements, thus reducing the number of i-cells present.

Theorem 6.1 shows that a combinatorial map can be reduced to a minimal combinatorial map if only edge contractions are used. It does not apply to a combinatorial map that is obtained by using other contraction or removal operations before all edges are contracted, as it cannot be guaranteed that face self-loops will not occur in such a combinatorial map. Still it might be advantageous for certain algorithms to select only a subset of edges for contraction in a first step to create an intermediate map and then using the other operations to reduce the number of i-cells in such this intermediate map. E.g. for building a pyramid. In the experimental results both variants of reducing a 3D combinatorial map will be shown.

According to theorem 6.1 a 3D combinatorial amp that does not contain face self-loops can be reduced to a minimal 3D combinatorial map. The size of the final 3D combinatorial maps obtained by this reduction process is known in advance as it depends only on the number of connected components in the volumetric data. Since each connected component is encoded by a exactly one vertex in the minimal 3D combinatorial map, the size of a minimal 3D combinatorial map G encoding n connected components can be specified by

$$n_v(G) = n + 1.$$

The number of vertices corresponds to the number of connected components n plus one vertex for the background region.

6.2 Minimal Configurations

In the previous section minimal 3D combinatorial map have been defined and it has been shown that such minimal maps can be obtained using edge contractions.

Additionally it is also of interest to examine combinatorial maps that are minimal in terms of i-cells to study their properties with a focus on the cells needed to encode a given configuration and on how to identify them. For

this purpose the a minimal configuration is defined as a combinatorial map encoding a given configuration with the minimal number of i-cells.

Definition 23 (Minimal 3D Configuration) *Given a combinatorial map G. G is a minimal configuration if the number of i-cells $c \in G$ is minimal and there is no combinatorial map G' encoding the same topological configuration as G.*

In this work a topological configuration is identified by the represented objects, the encoded adjacency and inclusion relations as well as the Euler characteristic of the bounding surfaces.

Within this report, minimal configurations are examined with respect to the contraction and removal operations introduced in chapter 4.

As will be shown in this section, there may be more than one minimal configuration for edges and volumes within a 3D combinatorial map. The reason for this is the strict bounding- and incidence-relation discussed in chapter 4. E.g. an edge must have a face and a volume it is incident to and must be bounded by vertices twice (In the case of an edge self-loop, the edge is bounded by the same vertex twice). This means that at least one cell of each dimension is needed for a valid combinatorial map. The number of additional cells and their dimension defines the different minimal configurations for edges, faces and volumes.

Proposition 6.1 *Given an nD combinatorial map $G = (D, \beta_1, \ldots, \beta_n)$. G contains at least one i-cell of each dimension $0 \leq i \leq n$.*

Proof: Given an nD combinatorial map $G = (D, \beta_1, \ldots, \beta_n)$ and a dart $d \in D$. According to proposition 2.2 d belongs to exactly one i-cell of each dimension. Thus G also contains at least one i-cell of each dimension. □

It follows from proposition 6.1 that a minimal 3D combinatorial map contains at least four i-cells (one i-cell of each dimension). For this reason the method to find minimal configurations for an edge, a face, and a volume used in this section, is to examine if the configurations can be encoded using only one i-cell of each dimension. If this is not the case, the minimal number of i-cells that must be added to the configuration is determined.

6.2.1 Edges

A combinatorial map must contain at least one i-cell of each dimension. To encode two different vertices connected by an edge, the minimal number of

i-cells is therefore five (2 vertices, 1 edge, 1 face, and 1 volume).

As will be shown, there is no such combinatorial map using five i-cells that can be obtained using the contraction and removal operations introduced in this report. Therefore combinatorial maps with six i-cells are considered minimal. Three such configurations with one additional i-cell are presented here.

The number of cells used for each of these configurations is shown in Table 6.1. All three configurations consist of six cells. Four darts are needed for the configurations "edge 1" and "edge 2" while six darts are necessary for "edge 3".

configuration	darts	vertices	edges	faces	volumes
edge 1	4	2	1	1	2
edge 2	4	2	2	1	1
edge 3	6	2	2	1	1

Table 6.1: Number of cells and darts for the three minimal representations of an edge.

According to definition 23, a combinatorial map is called minimal if the number of i-cells is minimal. Further, according to proposition 6.1, a combinatorial map must contain at least four i-cells (one i-cell for each dimension). Therefore a configuration that contains at least one edge connecting two vertices must consist of at least five i-cells: 2 vertices, 1 edge, 1 face, and 1 volume.

Proposition 6.2 *Given a 3D combinatorial map $G = (D, \beta_1, \beta_2, \beta_3)$ that encodes two different vertices and one edge connecting both vertices. G contains at least six i-cells if there is no dart $d \in D$ with $\beta_3(d) = d \vee \beta_2(d) = d$.*

Proof: Additionally to the two vertices and the one connecting edge, G must contain at least one face and one volume. To prove that G contains at least six i-cells, it is sufficient to show that no valid combinatorial map can be created with these five i-cells if there is no $d \in D$ with $\beta_3(d) = d \vee \beta_2(d) = d$.

The face in the configuration must be bounded by at least one edge. If more than one edge is used to bound the face, then this additional edge means that G contains at least six i-cells. To bound a face with only one edge, two possibilities exist:

1. The edge bounding the face is a self-loop. This is not possible for this configuration, since the only edge in the configuration connects two different vertices and is therefore no self-loop.

2. The face is bounded twice by the same edge. In this case the face is a face self-loop, with an inner and an outer volume (see Fig. 6.1). Note, that there must be an inner volume since $\beta_3(d) = d$ is not allowed by proposition 6.2. Thus an additional volume is introduced to the configuration and the total number of i-cells for this configuration is at least six.

□

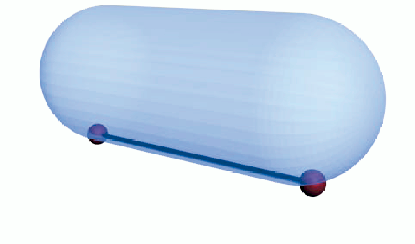
According to proposition 6.2, configurations with six i-cells are considered minimal. Three such configurations "edge 1", "edge 2", and "edge 3" are discussed here. The difference between these configurations is determined by the way the face is bounded.

For "edge 1" this face is bounded by the only edge twice. Thus forming an capsule like and empty face self-loop (Fig. 6.1a) with an inner and an outer volume. Therefore two volumes are present in this map. The face self-loop is indicated in the bounding relationship diagram by the 2 on the arrow connecting the face and edge (Fig. 6.1b).

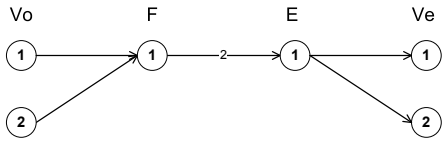
In both other configurations - "edge 2" and "edge 3" - an additional edge is introduced to bound the face. The difference between these two combinatorial maps is the kind of edge used to bound the face. "Edge 2" uses a parallel edge connecting the two vertices (Fig. 6.1c). The face is incident to the same volume on both sides and both edges connect to both vertices (Fig. 6.1d). The additional edge in the configuration "edge 3" is a self-loop, connecting to the same vertex on both endpoints. The face bounded by these two edges can be imagined as a conical surface (Fig. 6.1e). The bounding relationship of configuration "edge 3" is shown in Fig. 6.1f.

These three configurations cannot be simplified without either creating an invalid combinatorial map or changing the topological meaning of the configuration. The additional volume in "edge 1" is defined by a face self-loop which could be removed by a face removal. However, this would remove the only face from the configuration and is therefore not allowed by condition 7.

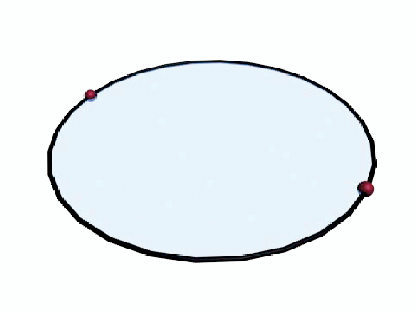
Using a face contraction to eliminate the parallel edge present in "edge 2" would also remove the only face and is therefore disallowed by condition 2. Contracting either of the edges would merge the two vertices thus changing



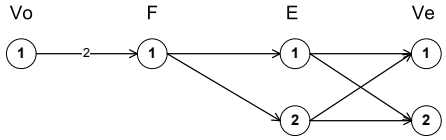
a) edge 1



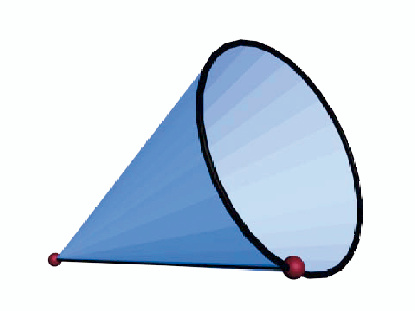
b) bounding relationship diagram for edge 1



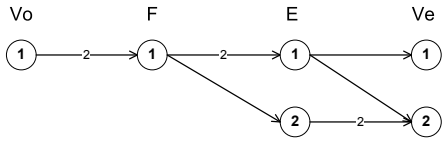
c) edge 2



d) bounding relationship diagram for edge 2



e) edge 3



f) bounding relationship diagram for edge 3

Figure 6.1: The 3 minimal representations that consist of at least one edge connecting two vertices (a,c,e) and their bounding relationship diagrams (b,d,e).

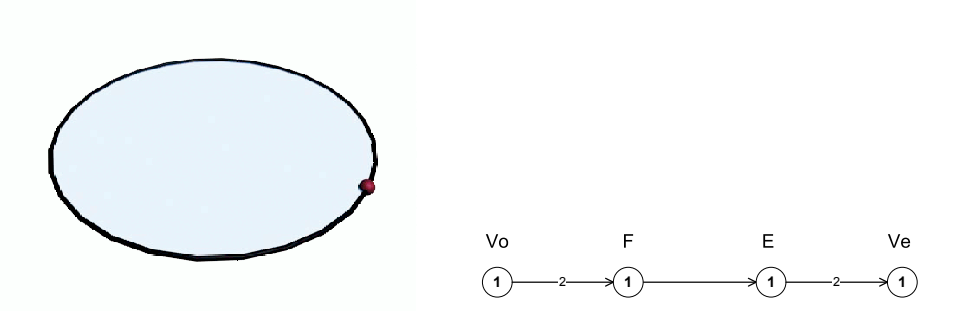
the configuration from encoding two adjacent voxels to a single voxel. The resulting configuration would be an edge self-loop that is bounded by the only remaining vertex twice.

Finally configuration "edge 3" is also not reducible. Removing the edge self-loop is disallowed by condition 1 as its degree is one. Contracting the second edge that connects the two vertices is again not allowed as it would merge the two vertices and thus the configuration would not contain two

vertices connected by an edge anymore.

6.2.2 Faces

There is exactly one minimal configuration containing at least one face. This configuration consists of one cell of each dimension and no additional cells are needed. It is therefore minimal according to proposition 6.1 The number of darts for this configuration is two as is shown in Table 6.2.



a) face 1 b) bounding relationship diagram for face 1

Figure 6.2: The minimal representation of a face (a) and its bounding relationship diagram (b).

configuration	darts	vertices	edges	faces	volumes
face 1	2	1	1	1	1

Table 6.2: Number of cells and darts for the minimal representation of a face.

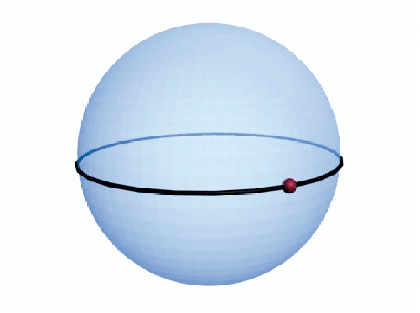
To encode a face using 3D combinatorial maps, only one cell of each dimension is needed. The face is bounded by an edge self-loop that connects to the same vertex on both sides. The face is incident to one volume that touches the face on both sides.

This configuration is not reducible as it already consists of only 1 cell of each dimension. Thus no i-cell can be removed without creating an invalid combinatorial map and it is therefore minimal according to definition 23 and proposition 6.1.

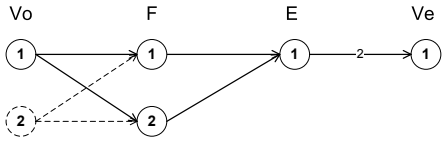
6.2.3 Volumes

In this section minimal configurations are examined that contain at least one volume that is completely bounded by at least one face. According to proposition 6.1 such a configuration must contain at least five i-cells: 1 vertex, 1 edge, 1 face, and 2 volumes (the inner and the outer volume). As will be shown, there is no combinatorial map obtained by the contraction and removal operations defined in this report that contains exactly five i-cells and encodes such a configuration.

Instead two combinatorial maps with six i-cells are presented. Both consist of four darts and six cells. (Table 6.3).

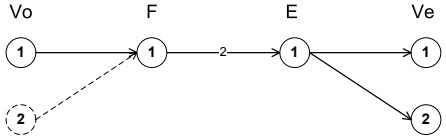


a) volume 1



b) bounding relationship diagram for volume 1

c) volume 2



d) bounding relationship diagram for volume 2

Figure 6.3: The minimal representations of a volume (a,c) and their bounding relationship diagrams (b,d). The outer volume is indicated in the bounding relationship diagrams by the dotted lines.

Proposition 6.3 *Given a 3D combinatorial map $G = (D, \beta_1, \beta_2, beta_3)$ that encodes 2 volumes and at least one face separating the two volumes. G contains at least 6 i-cells if there is no dart $d \in D$ with $\beta_3(d) = d \vee \beta_2(d) = d$.*

configuration	darts	vertices	edges	faces	volumes
volume 1	4	1	1	2	1 (2)
volume 2	4	2	1	1	1 (2)

Table 6.3: Number of cells and darts for the two minimal representation of a volume. The number of volumes present in these configurations is two if the outer volume is also counted.

Proof: The configuration contains at least two volumes and one face. Additionally one edge and one vertex are needed to satisfy proposition 6.1. Thus at least five i-cells are needed for this configuration. In order to prove that an additional i-cell is needed, it is sufficient to show that the bounding relations cannot be satisfied by these five cells if $\beta_3(d) = d$ and $\beta_2(d) = d$ are not allowed.

The face must be bounded by an edge. This edge must be a self-loop, because there is only one vertex in the configuration meaning that the edge must be bounded by this vertex twice. Since $\beta_2(d) = d$ is not allowed, a face bounded by an edge self-loop cannot separate two volumes (it can be imagined as one half of a globe incident to the same volume on both sides.).

Thus an additional face must be introduced to the configuration (the second half of the globe) to separate the two volumes and the configuration therefore contains six i-cells. □

Using proposition 6.3, configurations with six i-cells are considered minimal according to definition 23. Two such configurations "volume 1" and "volume 2" are presented here. Additionally, a combinatorial map without the restriction $\beta_2(d) \neq d$ that uses only five cells is discussed.

Similar to the minimal configurations of an edge, the difference between both combinatorial maps is defined by the kind of edge present. In the first configuration the volume is bounded by one vertex and one edge. The edge is a self-loop that connects to the only vertex twice, thus dividing the face bounding the volume into two parallel faces (Fig. 6.3a).

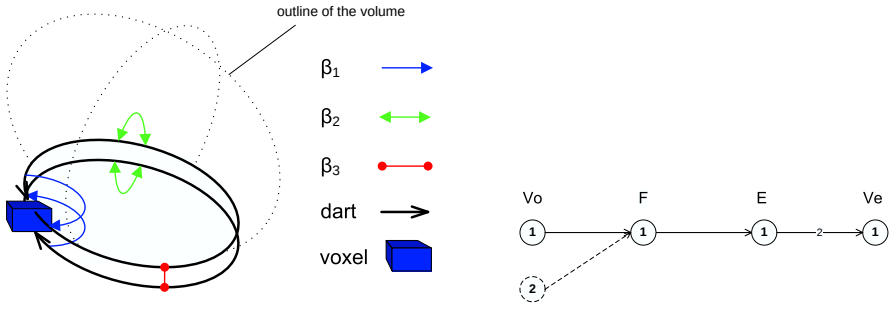
"Volume 2" is bounded by an edge that is not a self-loop, thus connecting two different vertices, adding an additional vertex to the map. This edge is bounding the only face twice, therefore the face is a face self-loop. Note that this configuration is identical to the "edge 1" configuration for an edge, and that it is also identical to its dual configuration.

Neither "volume 1" nor "volume 2" can be simplified. A reduction of the first configuration would require the elimination of the parallel faces using

a volume contraction. This operation would remove the inner volume, thus the configuration would not contain an inner volume separated by at least one face from the outer volume anymore.

The additional vertex in "volume 2" can only be removed by merging both vertices with an edge contraction. However, this operation would remove the only edge from the combinatorial map which is not allowed by condition 2.

Both configurations are minimal because they only use one i-cell of each dimension plus one additional i-cell for the second volume and one additional i-cell to satisfy the bounding relations (a face for "volume 1" and a vertex for "volume 2"). Therefore the total number of i-cells present in these configurations is six. Both configurations are minimal as there is no configuration consisting of only five i-cells that can be obtained by the operations used in this report (proposition 6.3).



a) combinatorial map b) bounding relationship diagram

Figure 6.4: The combinatorial map (a) and bounding relationship diagram (b) of a volume using 5 i-cells.

Fig. 6.4 shows a configuration that uses exactly one i-cell of each dimension plus the additional 3-cell to represent two volumes separated by a face. This is done by removing one of the faces from configuration "volume 1" without merging the inner and outer volume: the inner volume is then bounded by one face which is itself bounded by a single edge self-loop being bounded by one vertex (This configuration can be imagined like a balloon filled with air whose opening is closed tightly in order to prevent the air from escaping).

Such a configuration can be encoded by a combinatorial map. The β_2 permutation of the darts belonging to the bounding face are set to themselves:

$\beta_2(d) = d$. Note that this is still an involution and therefore valid. Fig. 6.4 shows an example of a combinatorial map encoding such a volume.

This configuration is more minimal than configuration "volume 1" and "volume 2" as it needs one i-cell less. But it cannot be obtained by the operations used in this report, as the removal of a face is defined by removing the face and merging the two incident volumes. Thus this configuration is not considered.

6.3 Topological Relations

Combinatorial maps are a suitable representation method for encoding the topological relations that define the structure of the processed data. In this section the way such relations are encoded in minimal configurations and how to identify them using pseudo elements is examined. Pseudo elements are non-empty self-loops.

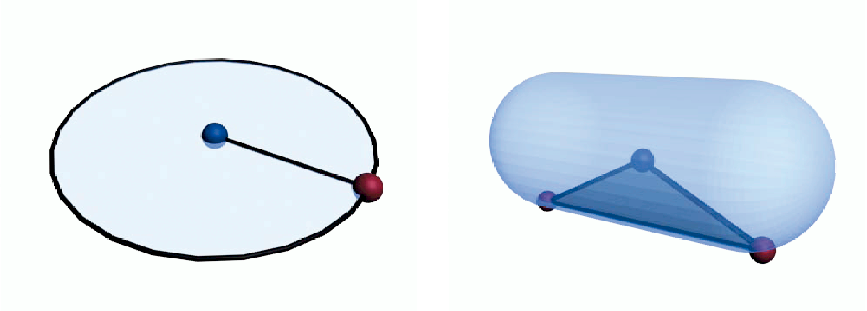
Definition 24 (Pseudo Element) *Given a combinatorial map G and an i-cell $c \in G$. c is a pseudo element, if it is an non-empty self-loop. c is a non-empty self-loop if there is an $(i+1)$-cell $c_i \in G, \bar{\delta}(c_i) > 1$ such that*

- *c_i is bounded by c;*
- *$\bar{\delta}(c) = 2$;*
- *c is bounded by exactly one $(i-1)$-cell $c_j \in G$.*

Given a combinatorial map G and an i-cell $c \in G$. c is a pseudo element if it is either a non-empty parallel

Based on this definitions, two kinds of pseudo elements exist in a 3D combinatorial map: non-empty edge self-loops and non-empty face self-loops. Fig 6.5 shows examples of these two types of pseudo elements.

In a more intuitive way pseudo elements can be seen as i-cells that define the topological structure encoded by a combinatorial map: a minimal configuration consists of 0-cells (vertices) encoding the objects, i-cells with $i \geq 1$ (edges, faces, and volumes) encoding the structure of the objects and additional i-cells with $i \geq 1$ needed to fulfill the boundary and incidence relations.



a) non-empty edge self-loop b) non-empty face self-loop

Figure 6.5: Examples of pseudo elements: non-empty edge self-loop (a) and non-empty face self-loop (b).

In 2D combinatorial maps, the two relations "adjacency" - two neighboring regions - and "contains" - a region or a set of regions that is completely enclosed by another region - can be identified. When working with 3D combinatorial maps, "surrounds" is introduced as a third topological relation.

Table 6.4 lists these relations and the pseudo elements used to identify them. They are directly related to the three basic configurations of an object in 3D: simplex, tunnel and hole, as discussed in [Illetschko et al., 2006a]. A simplex is a solid object that is adjacent to other objects. A tunnel represents a ring-like object that surrounds other objects and the hole represents an object enclosing other objects.

The representation of these topological relations in a minimal configuration as well as the methods to identify and discriminate them is discussed in the next sections.

6.3.1 Adjacency

Adjacency describes the fact that two objects are neighbors. A 3D object is adjacent to another object, if they have a common face that separates them.

In a minimal 3D combinatorial map each connected object is represented by one vertex. The adjacency of two objects is then expressed by an edge connecting two vertices. Thus the representation of adjacency in a minimal configuration is equivalent with the representation of the connecting edge. These configurations have been studied in section 6.2.

	object	relation	pseudo elements
	simplex	object is adjacent to other objects	“adjacency” is represented by edges connecting the vertices that represent the adjacent objects.
	tunnel	object surrounds other objects	“surrounds” is represented by a face bounded by an edge self-loop. The surrounding object is represented by the vertex on the boundary. The surrounded objects are enclosed by the face.
	hole	object encloses other objects	“encloses” is represented by a volume. The outer object it represented by the only vertices on the boundary. The inner objects are enclosed by the volume.

Table 6.4: Topological relations in a 3D combinatorial map.

It is sufficient to iterate through all edges of a given combinatorial map, in order to identify all adjacency relations encoded by this map. All edges connecting two different vertices indicate an adjacency relation.

To identify all objects being adjacent to a specific object, an algorithm similar to the calculation of the degree of a vertex can be used. The algorithm iterates through all darts belonging to the vertex that represents the object of interest. If an unmarked dart is encountered, the edge is investigated and all darts of this edge are marked. Marked darts are ignored in the subsequent iterations. For each edge found this way, the vertices connected by this edge are evaluated. If this edge connects two different vertices, an adjacency relation has been identified.

A C++ implementation of this algorithm is shown in listing 6.1. The function `adjacent_to` takes one vertex as its argument. It will then print all adjacency relations between the object represented by this vertex and other objects onto the screen.

6.3.2 Surrounds

This relation describes an object with a tunnel best imagined as a donut or a ring. As such it surrounds another object or a set of objects.

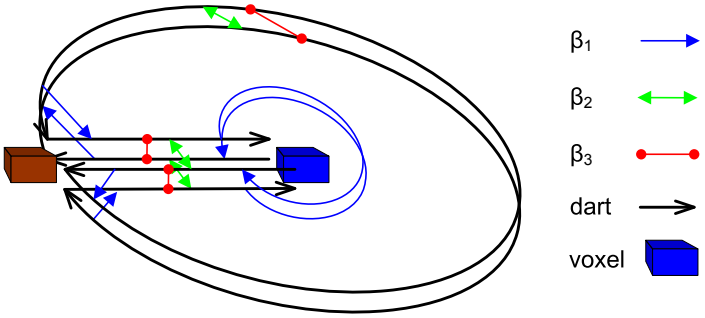
```
void adjacent_to(int vertex_1) {
  // get one dart of the vertex and initialize the flag
  // for all darts belonging to the vertex
  dart = get_dart_of_vertex(vertex_1);
  reset_marks_of_vertex(vertex_1);

  // iterate through all darts belonging to the vertex
  do {
    // examine edge if an unmarked dart is encountered
    if (dart_not_marked(dart)) {
      // get the the opposite vertex by following
      // the beta_3 permutation
      int vertex_2 = dart->beta_3();

      // if the vertices are not identical, then
      // an adjacency relation has been found
      if ( vertex_1 != vertex_2 ) {
        cout << ``adjacent to `` << vertex_2 << ``\n'';
      }

      // mark all darts belonging to the same edge
      tmp_dart = dart;
      do {
        mark_dart(tmp_dart);
        tmp_dart = beta_2(tmp_dart);
        tmp_dart = beta_3(tmp_dart);
      } while (tmp_dart != dart);
    }

    // get the next dart of this vertex
    dart = get_next_dart();
  } while (dart);
}
```

Listing 6.1: Identifying the adjacency relations of an object

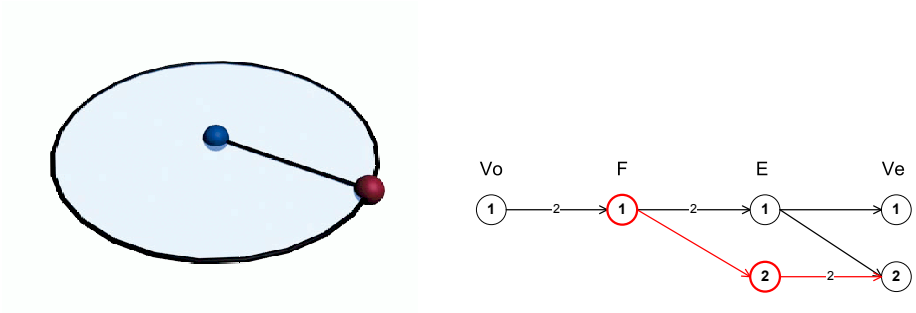
In a minimal configuration the ring-like outer object and the inner object are both represented by a single vertex. Both objects are adjacent, so these vertices are connected by an edge. The "surrounds" relation itself is represented by a face. This face is bounded by an edge self-loop and a bridge. The vertex incident to the edge self-loop and the bridge represents the outer object while the vertex of the inner object is incident only to the bridge. Thus this configuration can be imagined as a face with one vertex on the boundary and one vertex in the middle.

The face and the non-empty edge self-loop bounding this face are the

pseudo elements of this configuration.



a) combinatorial map



b) configuration c) bounding relationship diagram

Figure 6.6: The minimal representation of the "surrounds" relation. The pseudo elements are drawn in red in the bounding relationship diagram (c).

Fig. 6.6b shows an image of this configuration. This configuration cannot be reduced without changing the relation or creating an invalid combinatorial map. Removing the connecting edge is not allowed as it is a bridge. Thus its removal would disconnect the inner vertex and remove the adjacency between both objects. Eliminating the edge self-loop would remove the "surrounds" relation and is therefore not allowed.

Note that this configuration is equivalent with "edge 3", the third minimal representation of an edge. This can be verified by comparing the bounding relationship diagram of the surrounds relation (Fig. 6.6c) with the diagram of "edge 3" (Fig. 6.1f).

The six darts needed for the combinatorial map of this configuration as well as their permutations are shown in Fig. 6.6a.

An algorithm can be designed to detect "surrounds" relations within a

combinatorial map using the pseudo elements. This algorithm is shown in listing 6.2. The pseudo elements of this relation are the face containing the inner object and the non-empty edge self-loop bounding this face. The algorithm iterates through all edges of the map. In a minimal configuration edge self-loops are detected by comparing both endpoints. A self-loop is detected, if the two vertices are identical.

Since only non-empty self-loops indicate a "surrounds" relation an additional test is required. If one of the faces that is bounded by the self-loop contains vertices that do not belong to the self-loop, it is non-empty. This test can be further simplified since an empty edge self-loop consists of only one vertex. Therefore it is sufficient to test, whether more than one vertex is incident to one of the bounded faces.

```
// Create an Iterator that will process all edges
edge_iterator     edge_it = map->edges.begin();

// iterate through all edges
for(edge_it; edge_it!=map->edges.end(); edge_it++) {
  // get both vertices bounding the edge
  int vertex_1 = edge_it->cell(VERTEX);
  int vertex_2 = edge_it->beta_3()->cell(VERTEX);

  // if the vertices are identical, then this is an
  // edge self_loop
  if ( vertex_1 == vertex_2 ) {
    // if the self-loop bounds a non-empty face
    // then it is a surrounds-relation:
    // iterate through all faces incident to the edge
    face_iterator face_it = edge_it->faces()->begin();
    face_iterator last_face = edge_it->faces()->end();
    for (face_it; face_it!=last_face; face_it++) {

      // the self-loop is non-empty if the bounded
      // face has more then on vertex
      if ( face_it->number_of_vertices() > 1 ) {
        cout << ``non-empty self-loop at vertex ``
             << vertex_1 << ``\n'';
      }
    }
  }
}
```

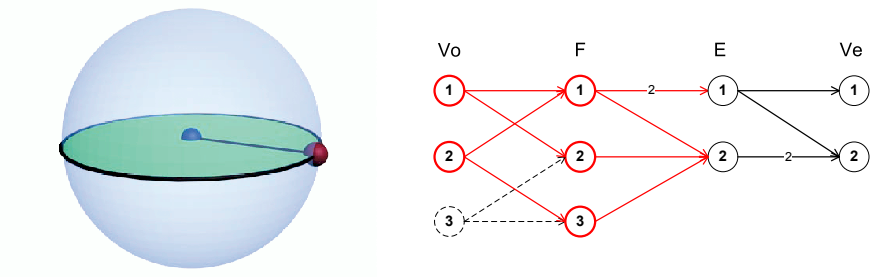
Listing 6.2: Algorithm for detecting "surrounds" relations in a combinatorial map

6.3.3 Contains

The third topological relation describes an object completely inside an outer object. This outer object can be imagined as a closed box containing another object.

There are two equivalent, minimal representations of the "contains" relation. They are related to the two configurations "volume 1" and "volume 2" presented as the minimal configurations of a volume. In both cases the inner object is represented by a single vertex. The "contains" relation is indicated by a volume that completely encloses this inner vertex. The volume itself is bounded by faces that themselves are bounded only by vertices belonging to the outer object. According to the minimal configurations of a volume, there are two different ways this volume can be represented.

The configuration "volume 1" uses a single vertex. In the first representation of the "contains" relation, the outer object is therefore also represented by a single vertex. This vertex is incident to an edge self-loop that divides the face bounding the volume into an "upper" and a "lower" half.



a) configuration b) bounding relationship diagram

Figure 6.7: The first minimal representation of the "contains" relation. The pseudo elements are drawn in red in the bounding relationship diagram (b). The volume surrounding the entire configuration is shown as well for the completeness of the diagram (dotted lines).

The adjacency of the inner and the outer object is represented by an edge connecting both vertices. The bounding relations are satisfied by an additional face added to this configuration. This face is spanned by the two edges - the self-loop bounding the faces of the volume, and the edge connecting the vertices. Note that this configuration is therefore a combination of the "surrounds" configuration and "volume 1". Fig. 6.7a shows this configuration.

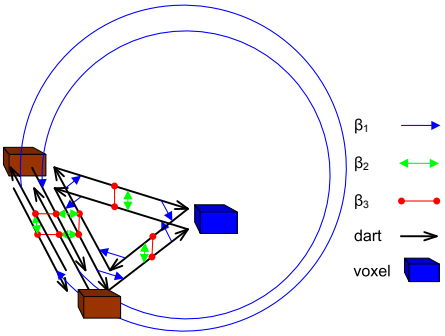
The additional face also divides the volume into two parts. Both volumes are bounded by an outer face defined only by the edge self-loop and the inner face that contains the connecting edge and the inner vertex. These two empty volumes are the pseudo elements of this configuration. Fig. 6.7b contains the bounding relationship diagram of this configuration. The two symmetrical volumes indicating the "contains" relation are drawn in red.

Combinatorial maps encode the orientation of faces around an edge. This can be used to build an algorithm to identify this "contains" relation in a minimal configuration. It utilizes the algorithm for detection of the surrounds relation. Once such a non-empty edge self-loop indicating the "surrounds" relation is found in the map, the previous and subsequent face incident to that edge are examined. If both are empty faces bounded only by the edge self-loop, it is a "contains" relation.

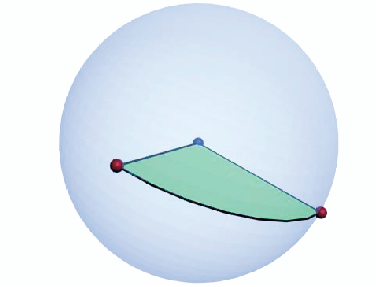
The second representation of the "contains" relation is related to "volume 2". In this case the outer object is represented by two vertices. They cannot be merged as the edge connecting them is also the only bounding edge of the face defining the volume. An edge contraction would therefore remove the bounding face as a side effect which is disallowed by condition 2.

The single vertex representing the inner object is therefore connected to both outer vertices by an edge to express the adjacency relation. Again an additional face must be added to satisfy the bounding relations of the combinatorial map. This face is spanned by the three edges connecting the three vertices (Fig. 6.8b). The combinatorial map of this configuration consists of 10 darts (Fig. 6.8a).

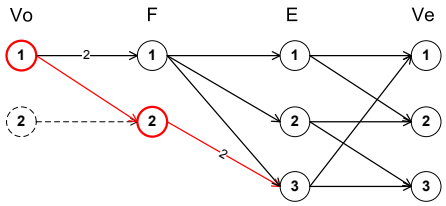
The pseudo elements of this configuration are the volume enclosing the inner vertex and its bounding face (Fig. 6.8c). This face is a non-empty face self-loop. Therefore the algorithm for detecting the "surrounds" relation can be reused to detect the "contains"-relation. The detection of non-empty edge self-loops is replaced by the detection of non-empty face self-loops.



a) combinatorial map



b) configuration



c) bounding relationship diagram

Figure 6.8: The second minimal representation of the "contains" relation. The pseudo elements are drawn in red in the bounding relationship diagram (c). The volume surrounding the entire configuration is shown as well for the completeness of the diagram (dotted lines).

6.4 Invalid Configurations

The last section of this chapter studies configurations that cannot be represented by the 3D combinatorial maps used in this report.

There are two constraints for configurations represented by a combinatorial map. According to conditions 2 and 7 the cells represented by the map must satisfy strict bounding- and incidence-relations. This restriction is true for any 3D combinatorial map regardless of the permutations used and it leads to the minimal configurations discussed in the previous sections of this chapter.

The second restriction is related to the permutations of the 3D combinatorial maps used in this report as proposed by [Braquelaire et al., 2003]. Condition 10 showed that the removal of a face can lead to an invalid

configuration due to the way vertices are defined in a combinatorial map $G = (D, \beta_1, \beta_2, \beta_3)$.

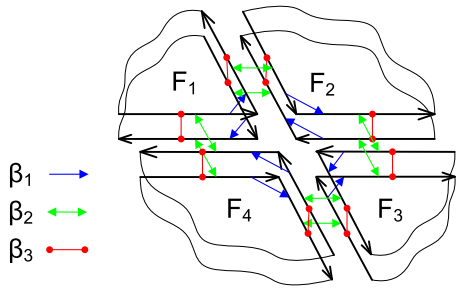
A vertex is defined by $V = (D_v, \beta_2\beta_1, \beta_3\beta_1)$. Therefore there must be a path from each dart $d_1 \in D_v$ to every other dart $d_2 \in D_v$ using the permutations $\beta_2\beta_1$ and $\beta_3\beta_1$.

Using $\beta_2\beta_1$ and $\beta_3\beta_1$ to obtain all darts belonging to a vertex means that all faces incident to a vertex must have at least one common edge with another face being incident to the same vertex. A face is incident to a vertex if one of its corners is incident to this vertex. The corners are defined by two subsequent edges around the boundary of the face. $\beta_3\beta_1$ is an involution whose cycle contains a dart for both edges of such a corner.

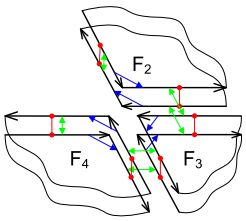
$\beta_2\beta_1$ will also return a dart of the second edge of such a corner, but from the next face around the edge that belongs to the same volume. β_2 and β_3 are identical if an edge is only incident to one face. This means that the results of $\beta_2\beta_1$ and $\beta_3\beta_1$ are also identical in this case. If both edges that define the corner of a face, are incident only to this face, then only the two darts of this corner can be obtained by both permutations and the face becomes isolated with respect to these permutations.

Thus every corner of a face that is incident to a vertex must have at least one edge in common with another corner of a face that is also incident to that vertex, or it must be the only corner of a face being incident to that vertex. Otherwise the combinatorial map would be invalid because not all darts of the vertex can be obtained by $\beta_2\beta_1$ and $\beta_3\beta_1$.

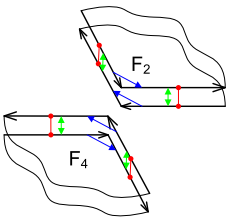
Fig. 6.9 gives an example of a configuration that violates this constraint. Fig. 6.9a shows part of a valid configuration consisting of a vertex that is incident to four faces. Each of the faces has a common edge with two of the other faces. Therefore each of the darts belonging to the vertex can be reached using the permutations $\beta_3\beta_1$ and $\beta_2\beta_1$. The same configuration is displayed in Fig. 6.9b after removing face F_1. F_2 and F_4 now have only one edge in common with another face of the vertex, but this configuration is still valid since the faces are connected via F_3. Fig. 6.9c shows the configuration after removing a second face F_3. There is no path using $\beta_3\beta_1$ and $\beta_2\beta_1$ from F_2 to F_4. So this configuration violates the definition of a vertex and is therefore invalid.



a) Vertex incident to 4 faces



b) After removal of F_1



c) Invalid configuration after removal of F_3

Figure 6.9: Invalid combinatorial map created by the removal of faces: In the initial configuration (a) a path to each dart exists using the permutations $\beta_3\beta_1$ and $\beta_2\beta_1$. Such a path still exists after the removal of face F_1 (b). After removing a second face - face F_3 - the remaining faces F_2 and F_4 are isolated, as no such path exists from darts of F_2 to darts of F_4 (c).

Chapter 7

Experimental Results

In this chapter the experimental results of this report are presented.

The first section examines the process of building a 3D combinatorial pyramid in the example of four representative configurations. This is done using the algorithm introduced in section 5.2. The purpose of these experiments is to examine pyramids built this way and to study the top level maps.

In the second section Minimal maps are investigated, demonstrating that a minimal 3D combinatorial map can be obtained for the configurations used in the first section by contracted all candidate edges in the first step. The differences between minimal maps and the maps obtained in the first set of experiments will be discussed.

The final section examines the performance of the reduction process for a 3D combinatorial pyramid by comparing the computational costs of building 3D combinatorial pyramids both in terms of bottom-up construction times and memory needed by pyramids of different sizes.

7.1 Construction of 3D Combinatorial Pyramids

7.1.1 Experimental Setup

In order to study 3D combinatorial pyramids, the C++ framework "COMA" was implemented for this report. This framework provides the functionality

for working with 2D and 3D combinatorial maps and pyramids. As such it implements all operations introduced in this report. [Illetschko et al., 2006b] describes this framework in detail.

It was decided to implement COMA even though a framework for 3D combinatorial maps already exists: "Moka - Modeleur de Cartes" [Damiand and Vidil, 2006]. Moka implements the 3D topological map which is a minimal combinatorial map associated to an image using the model and operations defined in [Damiand, 2001]. Differences between this work and the specific model and set of operations introduced in this report led to the decision for the implementation of COMA.

The main differences between these two works are the set of operations that are used to simplify a 3D combinatorial map and the the final combinatorial maps obtained by these operations. In the model described by [Damiand, 2001] voxels are represented by volumes instead of vertices. Thus the removal of a face is used to merge adjacent voxels and the removal of edges and vertices is used to simplify the maps. This corresponds to the contraction of edges used in this report to merge adjacent voxels and the use of face contractions and volume contractions to simplify the maps.

However, different methods are used to deal with i-cells that do not fulfill condition 3 (resp. condition 8) which disallows the contraction of self-loops (resp. the removal of bridges). While this report uses a second set of operations (edge removal and face removal) to deal with these cases, [Damiand, 2001] allows the removal of such i-cells. The possible disconnection of connected components as a result of the removal of such bridges is dealt with the use of an inclusion tree that is built in parallel. This additional data structure is used to retrieve the complete topological information from the minimal combinatorial maps obtained. On the other hand, using the operations described in this report creates a minimal combinatorial map that encodes all topological relations present in the data and thus pseudo elements can be used to identify specific topological relations.

Another difference is the use of the shifting operation by [Damiand, 2001]. This operation is used to deal with specific cases where a combinatorial map still contains redundant i-cells and is therefore not minimal, but a simplification is not allowed because the conditions are not fulfilled by any i-cell within the combinatorial map. The shifting operation then transforms the combinatorial map into a topologically equivalent combinatorial map where these i-cells can be eliminated. The use of such an operation has not been studied in this report, but will be the focus of further work based on the results presented here.

For the experiments presented in this chapter, the volumetric data is read as a stack of images that is converted into an initial map according to section 5.1. Based on this initial map a pyramid is built using the algorithm proposed in section 5.2. For the algorithm step a connected component algorithm is used that merges voxels of the same color.

The connected components algorithm is modified, as it would otherwise merge all neighboring voxels of the same color in one step. This would lead to very low pyramids that would only consist of few levels. Instead the modified algorithm selects at most one edge to be contracted for each vertex. All other edges of a vertex will be kept until the next level. All simplifaction steps will be applied before the next level is constructed. This means that it is not guaranteed that the top level map will contain a minimal combinatorial map, since not all candidate edges are contracted in the first step. Indeed it will be seen that one configuration is not reduced to a minimal combinatorial map. The construction of minimal combinatorial maps will be demonstrated in the next section with the second set of experiments.

Three methods are used to examine the top levels of each pyramid.

1. The receptive field of each vertex in the top level is visualized to validate the components they represent;
2. The whole map is visualized together with the pseudo elements indicating the topological relation within the data;
3. The bounding relationship diagram is generated for the top level to study the configuration.

The visualization of the combinatorial maps is done by using the "raybooster" software. This software was provided by the Computer Graphics Group of the Institute of Computer Graphics and Algorithms of the University of Technology Vienna. A detailed description of this software is provided by [Grimm et al., 2004b, Grimm et al., 2004a, Bruckner, 2004]. It reads voxel files generated by the COMA framework and creates a 3D view of the data.

Voxel files are generated for the receptive fields of each vertex in the top level of the pyramids as well as for the complete combinatorial map. The pseudo elements present in these visualizations are indicated by manually adding white faces and edges. For this purpose the COMA framework creates an output that indicates which vertices are part of a "surrounds" or a

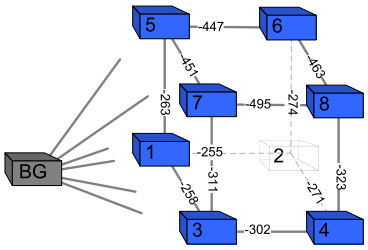
"contains" relation. Faces are then added for the "surrounds" relation and edges for the "contains" relation.

The bounding relationship diagrams are also generated by the COMA framework.

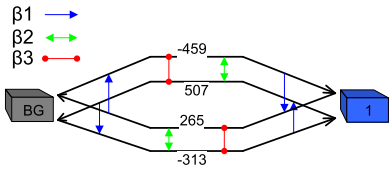
7.1.2 Simplex

	darts	vertices	edges	faces	volumes
level 0 (initial map)	120	9	20	18	7
level 1	32	5	8	5	2
level 2	6	3	3	1	1
level 3 (final map)	4	2	2	1	1

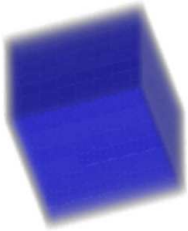
Table 7.1: Reduction of a simplex configuration. The table lists the number of darts and cells at each level of the pyramid.



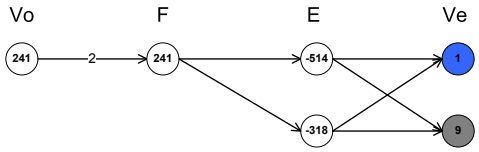
a) initial configuration



b) final combinatorial map



c) visualization of the top level



d) bounding relationship diagram of the top level

Figure 7.1: Reduction of a simplex configuration shown as the initial map representing the data (a), the final combinatorial map of the top level (b), the 3D visualization of the top level (c), and the bounding relationship diagram of the top level (d).

The first configuration represents a simplex. A single solid $2 \times 2 \times 2$ cube that is only adjacent to the background.

The initial map consists of a $2 \times 2 \times 2$ grid, where each vertex has the same color. This map can be seen in Fig. 7.1a. The background vertex is only symbolized to enhance the figure's clarity.

The initial map consists of 120 darts representing 9 vertices, 20 edges, 18 faces, and 7 volumes. This map is reduced in three levels to its minimal form of 4 darts, 2 vertices, 2 edges, 1 face, and 1 volume. The number of darts and cells at each level of the pyramid are listed in table 7.1.

The two vertices at the top level (Fig. 7.1b) represent the simplex object and the background. These two are adjacent to each other which is indicated by the two edges connecting both vertices. The final map is identical to the first minimal configuration of an edge - "edge 2" - and is therefore one of the minimal configurations to represent two adjacent objects. This can be verified by comparing the bounding relationship diagram of the top level (Fig. 7.1d) with the one for "edge 2" (Fig. 6.1d).

Note, that the top level map is also a minimal combinatorial map, as the only connected component in the configuration is reduced to a single vertex.

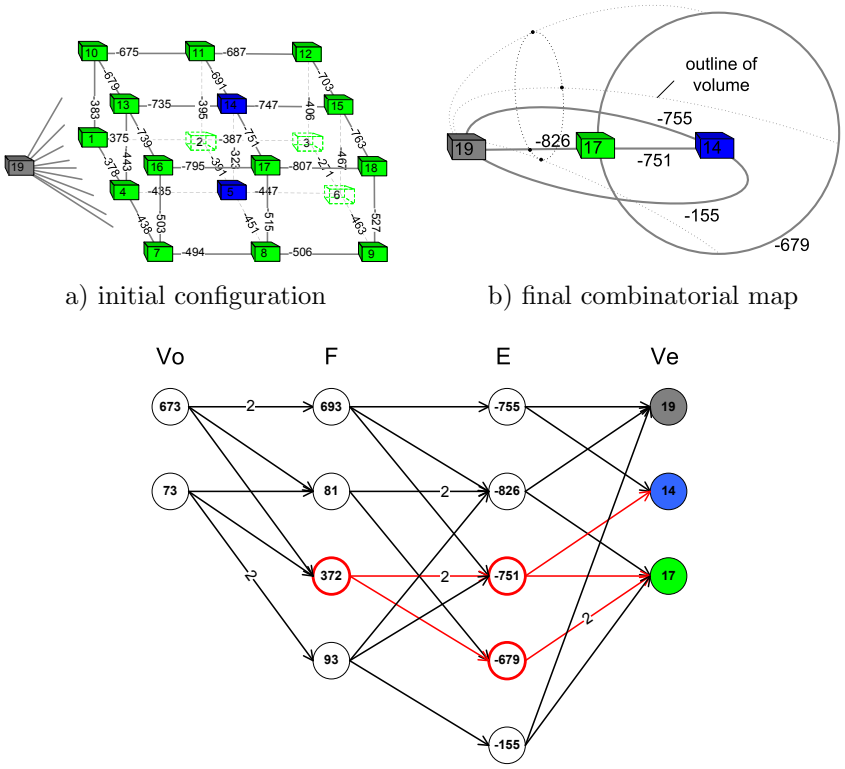
The visualization of the top level (Fig. 7.1c) shows the complete cube. So both, the data and the topological relations, are shown correctly in the final map of the pyramid.

7.1.3 Tunnel

	darts	vertices	edges	faces	volumes
level 0 (initial map)	352	19	51	52	20
level 1	224	10	30	36	16
level 2	112	6	16	18	8
level 3	56	4	9	9	4
level 4 (final map)	24	3	5	4	2

Table 7.2: Reduction of a tunnel configuration. The table lists the number of darts and cells in each level of the pyramid.

The second experiment shows the reduction of a $3 \times 3 \times 2$ ring-like object: the tunnel configuration. The ring surrounds an inner object. Thus the



a) initial configuration

b) final combinatorial map

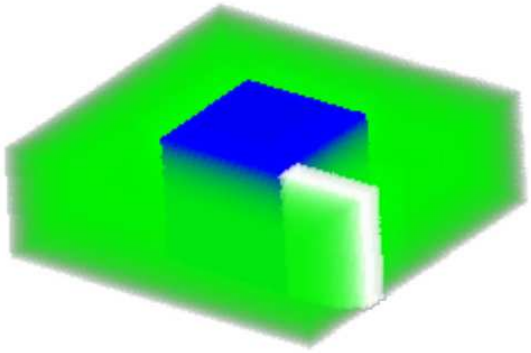
c) bounding relationship diagram of the top level

Figure 7.2: Reduction of a tunnel configuration shown as the initial map representing the data (a), the final map of the top level (b), and the bounding relationship diagram of the top level (c). The "surrounds" relation is pointed out in the diagram by the cells and arrows in red.

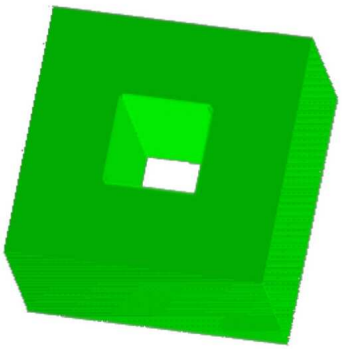
expected result is a combinatorial map encoding the adjacency of these two objects as well as the "surrounds" relation of the outer object.

The initial $3 \times 3 \times 2$ grid is shown in Fig. 7.2a. This map represents 19 vertices, 51 edges, 52 faces, and 20 volumes using 352 darts and is simplified in four levels. The final map consists of 24 darts encoding 3 vertices, 5 edges, 4 faces, and 2 volumes (Fig. 7.2b). Table 7.2 contains the number of darts and cells for each level of the pyramid.

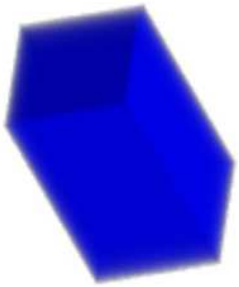
Three vertices are present in the combinatorial map at the top of the pyramid. They represent the ring (vertex 17), the inner object (vertex 14), and the background (vertex 19). The edges connecting these vertices show the adjacency of each object to the other two. The inner object is connected



a) visualization of the top level



b) receptive field of vertex 17



c) receptive field of vertex 14

Figure 7.3: Visualization of the tunnel configuration. (a) shows the visualization of the top level. The white face indicates the "surrounds" relation of the green ring. The receptive fields are visualized using the Raybooster software (b&c).

to the background by two edges. This represents the fact that it touches the background on both sides of the ring. The fifth edge (label -679) is a self-loop. This edge self-loop and the connecting edge -751 span a face (label 372) that connects the ring with the inner object. This is the pseudo element indicating the "surrounds" relation of the ring (Fig. 7.2c). The final map of the pyramid is also a minimal combinatorial map, as it encodes each connected component with one vertex.

Fig. 7.3a shows the visualization of the top level. A white face that cuts the green ring illustrates the pseudo element identifying the "surrounds"

relation of the green ring. The receptive fields of the two vertices in the final map are visualized in Fig. 7.3b&c. The receptive field of vertex 17 produces the ring while the receptive field of vertex 14 is the inner object of the configuration. This shows the correctness of the data represented by the vertices.

7.1.4 Hole

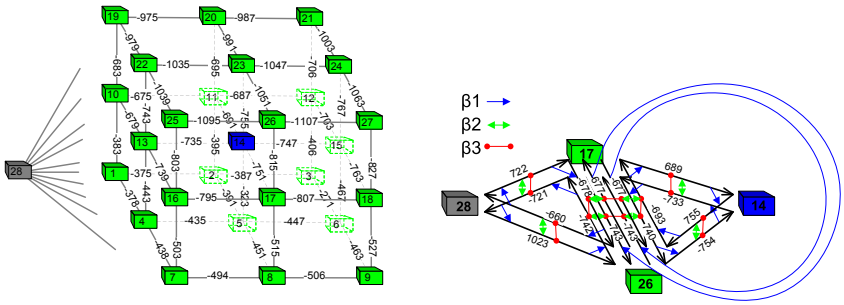
	darts	vertices	edges	faces	volumes
level 0 (initial map)	576	28	80	84	32
level 1	374	16	49	58	25
level 2	162	9	23	25	11
level 3	80	6	13	13	6
level 4	24	4	5	4	3
level 5 (final map)	16	4	5	3	2

Table 7.3: Reduction of a hole configuration. The table lists the number of darts and cells in each level of the pyramid.

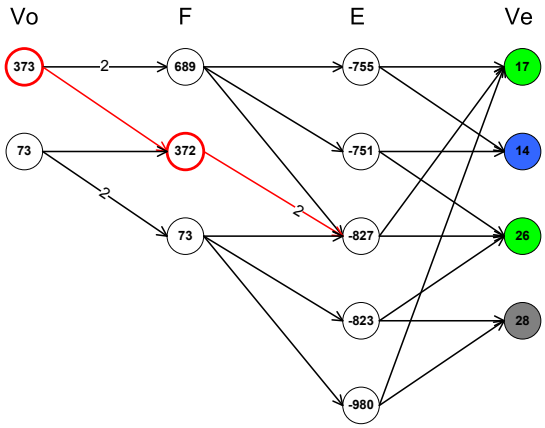
This experiment demonstrates the reduction of a configuration with a "contains" relation. The input data represents a $3 \times 3 \times 3$ cube that encloses an inner object.

The initial map for this configuration consists of 576 darts, 28 vertices, 80 edges, 84 faces, and 32 volumes (Fig. 7.4a). A pyramid six levels high is built to reduce this configuration to its top level combinatorial map shown in Fig. 7.4b. This map contains 16 darts, 4 vertices, 5 edges, 3 faces, and 2 volumes. Table 7.3 lists the number of darts and cells at each level of the pyramid.

The top level map encodes four vertices. One vertex represents the background (vertex 28), two vertices are representing the outer box (vertex 17 & 26) and one represents the inner object (vertex 14). The adjacency relations within the data are expressed by the five edges. The background (edges -823 & -980) and the inner object (edges -751 & -755) are adjacent to the two vertices representing the outer cube. An additional edge (label -827) shows the adjacency of the two halves of the outer box. This edge cannot be contracted because it is the only edge bounding face 372. Contracting the edge would violate condition 2 (Bounding and Incidence Relations) by removing the face as a side effect and thus the operation is disallowed.



a) initial configuration b) final combinatorial map

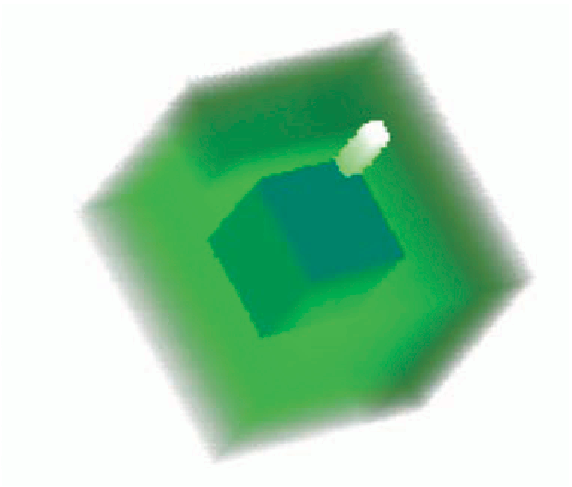


c) bounding relationship diagram of the top level

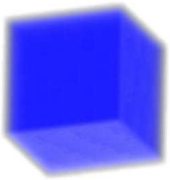
Figure 7.4: Reduction of a hole configuration shown as the initial map representing the data (a), the final combinatorial map of the top level (b), and the bounding relationship diagram of the top level (c). In the diagram the "contains" relation is pointed out by the cells and arrows in red.

One of the three faces (label 372) is the pseudo element of this configuration that indicates the "contains" relation. It is a face self-loop bounded only by the edge connecting the two vertices of the outer box. The face defines volume 373 that completely encloses the inner object. This configuration is topologically equivalent to the second minimal "contains" configuration as can be seen by comparing their bounding relationship diagrams (Fig. 7.4c & 6.8c).

Note, that the top level map of this pyramid is not a minimal combinatorial map, as the outer box is encoded by two vertices. This demonstrates,

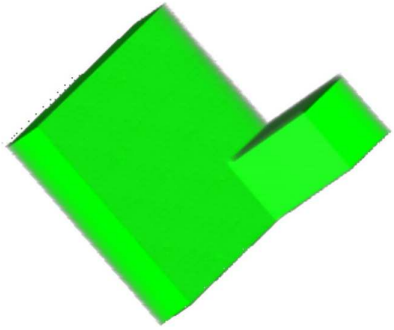


a) visualization of the top level



b) receptive field of vertex 14

c) receptive field of vertex 17



d) receptive field of vertex 26

Figure 7.5: Visualization of the hole configuration. (a) shows the visualization of the top level. The white edge indicates the "contains" relation of the outer box. The receptive fields are visualized using the Raybooster software (b,c&d).

that a minimal combinatorial map might not be obtained if other operations than the contraction of an edge are applied, before all connected components have been reduced to a single vertex.

The visualization of the top level is shown in Fig. 7.5a. The outer box is divided into two parts. This corresponds to the fact that it is represented by two vertices in the combinatorial map. The white edge indicates the "contains" relation of the outer box. The receptive fields of vertex 17 & 26 compose the $3 \times 3 \times 3$ cube with a hole in the middle (Fig. 7.5c&d). The

receptive field of vertex 14 shows the inner object (Fig. 7.5e).

7.1.5 Two Rings

	darts	vertices	edges	faces	volumes
level 0 (initial map)	800	37	109	116	44
level 1	506	19	64	79	34
level 2	262	10	31	41	20
level 3	156	6	18	25	13
level 4	74	4	10	12	6
level 5 (final map)	36	3	6	6	3

Table 7.4: Reduction of two rings. The table lists the number of darts and cells in each level of the pyramid.

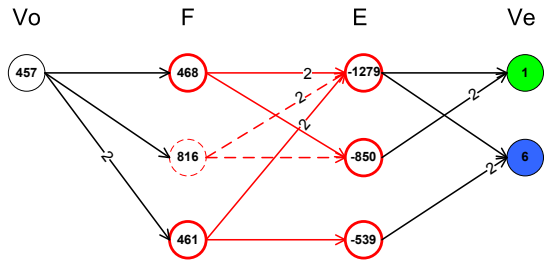
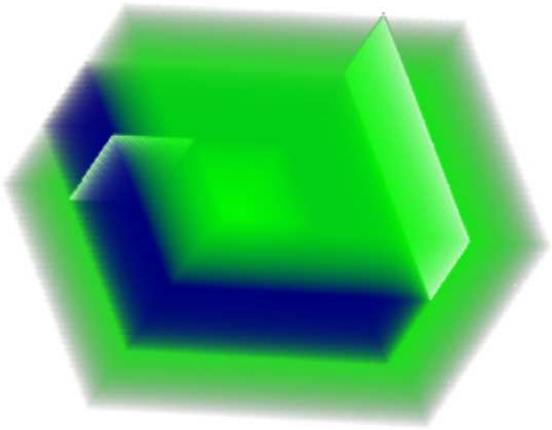


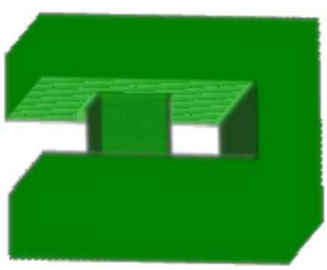
Figure 7.6: Bounding relationship diagram for the final map of two rings. The background vertex and all cells bounded by it are suppressed in the diagram. The pseudo elements are pointed out by the red cells and arrows. The diagram contains one redundant pseudo element that is indicated by the dotted lines.

The fourth experiment shows the reduction of two objects that are intertwined like two rings of a chain. In addition to being adjacent to each other, both rings also surround each other. The expected result of this experiment is therefore a map encoding the adjacency relation and both "surrounds" relations.

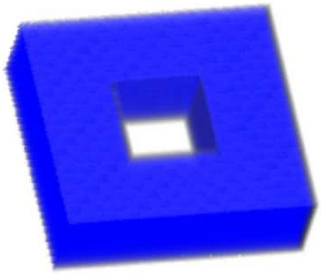
The initial map consists of a $4 \times 3 \times 3$ grid that contains both rings. The two objects have a different geometric shape, but are topologically equivalent to a ring. The grid-like base level of the pyramid contains 800 darts



a) visualization of the top level



b) receptive field of vertex 1



c) receptive field of vertex 6

Figure 7.7: Visualization of the final map of two rings. (a) shows the visualization of the top level. Both rings surround each other. This is indicated by two white faces, each cutting one ring. The receptive fields are visualized using the Raybooster software (b&c).

representing 37 vertices, 109 edges, 116 faces, and 44 volumes. It is reduced in five levels to a minimal combinatorial map of 36 darts encoding 3 vertices, 6 edges, 6 faces, and 3 volumes. The number of darts and cells at each level of the pyramid are listed in Table 7.4.

Fig. 7.6 shows the bounding relationship diagram of the final map of this pyramid. In order to bring out the relations of the two rings more clearly, the background vertex (vertex 37) is not shown in this diagram as well as all other cells bounded by it.

The diagram shows that the final combinatorial map consists of two vertices. Each represents one of the two rings. The adjacency between the two

objects is encoded by one edge connecting them (edge -1279). Each vertex is also incident to an additional edge that is an edge self-loop (edges -850 & -539). Both self-loops are the boundary of a face (faces 461 & 468) together with the edge -1279 that connects both rings. These faces are the pseudo elements for the "surrounds" relation of each ring around the other one. Thus the topological relations are correctly encoded by this final map. Note that there is a third face in the diagram (label 816) that represents a redundant pseudo element for one of the "surrounds relations" as it is bounded by the same edges as face 468. It cannot be removed from the map as it does not fulfill the conditions of either a removal or a contraction: It cannot be contracted because it is has a dual-degree of 3 which violates condition 1. A face removal is also not allowed since the face is not an empty self-loop. The final map is a minimal combinatorial map, as it encodes each connected component with a single vertex.

Fig. 7.7 shows the visualizations of this experiment. Fig. 7.7a contains the top level with both rings. The two "surrounds" relations are indicated by the two white faces. Each cuts one of the two rings showing that each ring surrounds the other one. The receptive fields of both vertices are displayed in Fig. 7.7b&c. The visualization shows that both objects are correctly represented by the vertices in the the final map.

In this experiment the data and the topological relations have been correctly represented, but an additional face was found in the final map. This face is redundant but cannot be eliminated from the map by the operations used in this report, as it does not fulfill the conditions for a face contraction or a face removal.

7.2 Minimal 3D Combinatorial Maps

The second set of experiments demonstrates that a minimal combinatorial map can always be obtained for any combinatorial map that does not contain volumes bounded by only one face. This is done by contracting all edges connecting vertices that belong to the same connected component before any simplification step is applied.

While such minimal 3D combinatorial maps will encode each connected component using a single vertex, it is not ensured that the number i-cells encoded by such a map is minimal. To illustrate this, the configurations from the last section are used again for these experiments. Each configuration will

be reduced to a minimal 3D combinatorial map. The resulting map is then compared to the top level map obtained by the pyramids used in the first set of experiments.

7.2.1 Simplex

The simplex configuration contains exactly one connected component. a minimal 3D combinatorial map must therefore consist of two vertices: one vertex encoding the only connected component and one vertex encoding the background.

	darts	**vertices**	edges	faces	volumes	i-cells
initial map	120	**9**	20	18	7	54
top level of the pyramid	4	**2**	2	1	1	6
minimal combinatorial map	4	**2**	1	1	2	6

Table 7.5: Number of darts and cells for maps encoding the simplex configuration. The numbers are listed for the initial map, a map obtained by a pyramid using all 5 operations and a minimal combinatorial map obtained by contracting all candidate edges in the first step.

Table 7.5 lists the result of the reduction. As can be seen in the third row that lists the values for the minimal combinatorial map, the map was indeed reduced to 2 vertices and it is therefore minimal.

The second row contains the number of cells encoded by the top level map from the first experiment. As can be seen, both use exactly 6 cells in total, so they are equivalent in terms of cells encoded by the map. However, the type of cells is different. The pyramid produced a final map in which the two vertices are connected by two edges that bound a single face and volume. The minimal combinatorial map produced in this experiment encodes one edge connecting both vertices. This edge bounds an empty face self-loop that separated two volumes.

7.2.2 Tunnel

Again the tunnel configuration from the first set of experiments is used. A minimal combinatorial map is created using only edge contractions in the first step and the two final maps are compared. Table 7.6 shows the number

of darts and cells for the initial map, the final map of the tunnel configuration from the last section, and the minimal combinatorial map produced by applying all edge contractions first.

	darts	**vertices**	edges	faces	volumes	i-cells
initial map	352	**19**	51	52	20	142
top level of the pyramid	24	**3**	5	4	2	14
minimal combinatorial map	54	**3**	7	1	6	26

Table 7.6: Number of darts and cells for maps encoding the tunnel configuration. The numbers are listed for the initial map, a map obtained by a pyramid using all 5 operations and a minimal combinatorial map obtained by contracting all candidate edges in the first step.

As can bee seen, the tunnel configuration is reduced to minimal 3D combinatorial map in both cases, but the number of other cells differs. However, even by applying all edge contractions in the first step, the total number of cells is reduced from 142 in the initial map, to 26 in the minimal combinatorial map, which is a reduction factor of more than 5. The map obtained by the pyramid is also minimal and it contains the least number of cells, as it is reduced to 14 cells.

7.2.3 Hole

The hole configuration is of special interest for this experiment, as it was not reduced to a minimal 3D combinatorial map by the pyramid. Table 7.7 shows the number of darts and cells for all three maps.

	darts	**vertices**	edges	faces	volumes	i-cells
initial map	576	**28**	80	84	32	224
top level of the pyramid	16	**4**	5	3	2	14
minimal combinatorial map	124	**3**	15	24	12	54

Table 7.7: Number of darts and cells for maps encoding the hole configuration. The numbers are listed for the initial map, a map obtained by a pyramid using all 5 operations and a minimal combinatorial map obtained by contracting all candidate edges in the first step.

This table shows, that the hole configuration is indeed reduced to a minimal 3D combinatorial map by applying all edge contractions as the first step. It also reduced the number of cells from 224 in the initial map to 54. This is a reduction factor of more then 4.

The map produced by the pyramid is not minimal, as it consists of 4 vertices. This is one more vertiex than connected components (including the background vertex).However, it is smaller in terms of encoded cells of any dimension. Using the pyramid the inital map was reduced to 14 cells which is a reduction factor of 16.

7.2.4 2 Rings

The final configuration from the first set of experiments is the 2 rings configuration. Table 7.8 lists the number of darts and cells for the inital map, the map produced by the pyramid, and the minimal combinatorial map created by applying all edge contractions in the first step.

	darts	**vertices**	edges	faces	volumes	i-cells
initial map	800	**37**	109	116	44	306
top level of the pyramid	36	**3**	6	6	3	18
minimal combinatorial map	116	**3**	12	22	15	52

Table 7.8: Number of darts and cells for maps encoding the 2 rings configuration. The numbers are listed for the initial map, a map obtained by a pyramid using all 5 operations and a minimal combinatorial map obtained by contracting all candidate edges in the first step.

As is expected, the configuration is again reduced to a minimal 3D combinatorial map. The map obtained encodes 2 connected components and the background using exactly 3 vertices. As for the hole configuration and the tunnel configuration, the total number of cells is higher than for the map produced by the pyramid. For this configuration the pyramid created a minimal combinatorial map encoding 18 cells, while the map produced in this experiment encodes 52, which is a factor of about 3.

All 4 configurations from the first set of experiments were successfully reduced to a minimal 3D combinatorial map. This demonstrates that every 3D combinatorial map can be reduced to a minimal combinatorial map by using edge contractions to reduce all connected components to a single vertex before applying the simplification steps.

7.3 Performance

The last set of experiments examines the performance of 3D combinatorial pyramids by studying the resources used to build such pyramids as well as the time needed to construct them. For this purpose combinatorial pyramids for initial maps of different sizes that represent the same configuration are compared.

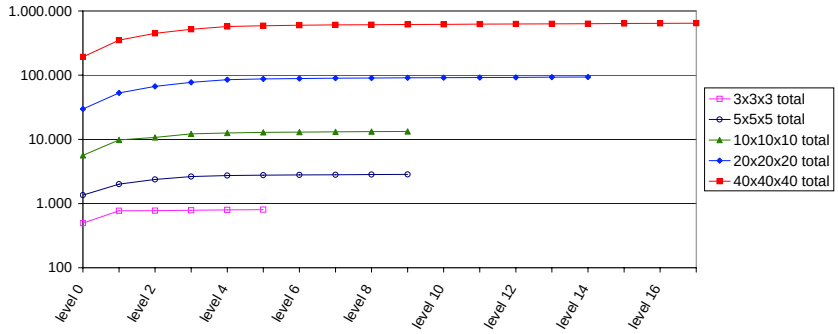
7.3.1 Size of Combinatorial Pyramids

level	$3 \times 3 \times 3$	$5 \times 5 \times 5$	$10 \times 10 \times 10$	$20 \times 20 \times 20$	$40 \times 40 \times 40$
0	496 (496)	1.356 (1.356)	5.640 (5.640)	29.720 (29.720)	192.640 (192.640)
1	772 (276)	2.004 (648)	9.820 (4.180)	52.944 (23.224)	351.660 (159.020)
2	776 (4)	2.372 (368)	10.728 (908)	66.980 (14.036)	449.256 (97.596)
3	784 (8)	2.628 (256)	12.156 (1.428)	77.356 (10.376)	518.000 (68.744)
4	796 (12)	2.728 (100)	12.584 (428)	85.148 (7.792)	574.056 (56.056)
5	**804 (8)**	2.768 (40)	12.864 (280)	87.448 (2.300)	589.832 (15.776)
6		2.788 (20)	13.004 (140)	88.616 (1.168)	599.724 (9.892)
7		2.808 (20)	13.108 (104)	89.596 (980)	606.512 (6.788)
8		2.828 (20)	13.200 (92)	90.340 (744)	611.632 (5.120)
9		**2.848 (20)**	**13.268 (68)**	90.868 (528)	615.908 (4.276)
10				91.396 (528)	619.696 (3.788)
11				91.920 (524)	623.408 (3.712)
12				92.444 (524)	627.040 (3.632)
13				92.964 (520)	630.628 (3.588)
14				**93.488 (524)**	634.192 (3.564)
15					637.760 (3.568)
16					641.328 (3.568)
17					**644.896 (3.568)**

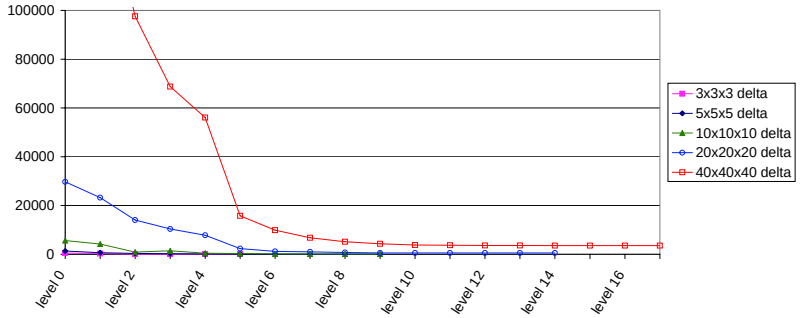
Table 7.9: Memory allocated by pyramids of different sizes. The memory is specified in kilobytes. Two numbers are given for each pyramid and level: the total amount of memory of the complete pyramid up to this level and the memory used for the combinatorial map of this level (in brackets).

Table 7.9 lists the allocated memory for five different pyramids. Each pyramid is initialized with a grid-like combinatorial map as the base level. These maps represent the same simplex configuration, but with different sizes ($3 \times 3 \times 3$, $5 \times 5 \times 5$, $10 \times 10 \times 10$, $20 \times 20 \times 20$ and $40 \times 40 \times 40$).

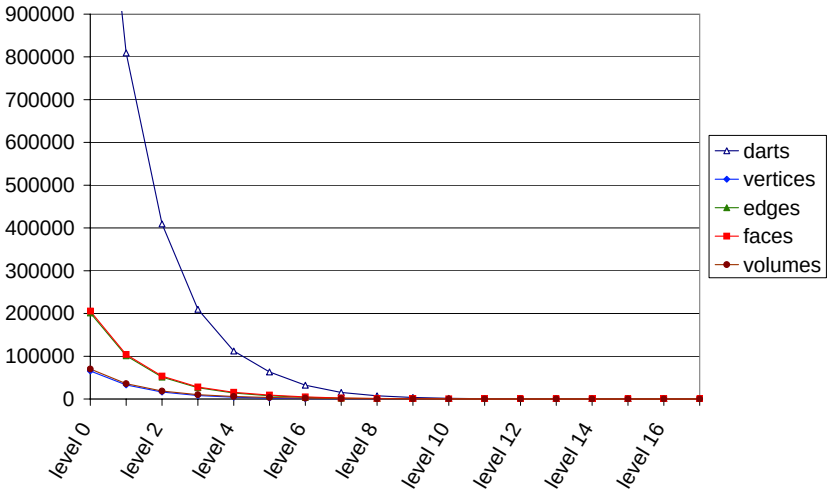
The pyramids are then built using the algorithm introduced in section 5.2 to reach the minimal combinatorial map for the simplex configuration. This



a) total memory allocation in kilobytes for pyramids of different sizes



a) memory allocation per level in kilobytes for pyramids of different sizes



b) number of darts and i-cells of a pyramid reducing a $40 \times 40 \times 40$ map

Figure 7.8: Size of combinatorial pyramids. (a,b) show the memory allocated (total and per level) by pyramids of different sizes. The total amount is shown using a logarithmic scale (a). (c) displays the number of darts and i-cells at each level of a pyramid that reduces a $40 \times 40 \times 40$ combinatorial map.

results in a pyramid 6 levels high for the smallest map and 18 levels are needed for the $40 \times 40 \times 40$ map.

Note, that it is not easily possible to calculate the memory needed by COMA for a map of a given size, even though it is known that COMA uses 80 bytes for each dart object. The reason is that COMA uses additional memory for internal data structures (e.g. arrays of i-cells, the voxels represented by the vertices, the connecting walks for non-surviving darts, etc.). Therefore it is not sufficient to calculate the number of darts using the formula from section 5.1 and multiplying it by 80 bytes.

Instead the allocated memory is measured after each level is completed. Table 7.9 gives the amount of memory for each level and pyramid by specifying two numbers: the total amount used by the pyramid up to this level and the amount of memory used for the new level alone. Charts displaying the memory allocation of all pyramids are shown in Fig. 7.8a&b.

The $3 \times 3 \times 3$ pyramid uses 496 kilobytes for the initial map and the complete pyramid is stored in 804 kilobytes. Thus the complete pyramid is built using less then a megabyte. On the other hand, nearly 200 megabytes (192.640 kilobytes) are needed for the base level of the largest configuration. The complete pyramid that reduces this $40 \times 40 \times 40$ map is stored in 645 megabytes (644.896 kilobytes).

For each of the pyramids the first levels are the most memory intensive, but after the construction of these first levels, the combinatorial maps are significantly reduced. For the smallest map two levels are needed to reduce it to a combinatorial map that has only 4 kilobytes, which resembles a reduction factor of 100. The same factor is reached after five levels for the largest pyramid. In this case the initial map of about 200 megabytes is reduced to a map that uses only 15,7 megabytes.

It can also be seen that the amount of memory needed for the internal data structures of COMA becomes less dominant for larger maps: The $3 \times 3 \times 3$ map uses 496 kilobytes and consists of 576 darts. The memory usage per dart is therefore $496kb/576 \approx 0,86kb$ which is ten times the size of the dart object itself (80 bytes). On the other hand the $40 \times 40 \times 40$ map uses about 200 megabytes for 1.569.672 darts. Thus the memory usage per dart is $192.640kb/1.569.672 \approx 0,12kb$ which is only a factor of $\sim 1,5$.

The reduction process is examined in more detail for the $40 \times 40 \times 40$ combinatorial map. Table 7.10 and Fig. 7.8c show the number of darts and cells of each level. The numbers for the initial map depend on the size of the grid. Thus ~1,6 million darts are needed to represent all cells of level 0.

level	darts	vertices	edges	faces	volumes
0	1.608.672	65.601	201.204	205.725	70.122
1	809.312	32.801	101.264	104.225	35.762
2	409.632	16.401	51.294	53.475	18.582
3	209.792	8.201	26.309	28.100	9.992
4	112.096	4.101	13.875	15.530	5.756
5	63.248	2.051	7.658	9.245	3.638
6	32.176	1.026	3.934	4.856	1.948
7	15.408	514	1.938	2.388	964
8	7.440	258	948	1.170	480
9	3.560	130	455	565	240
10	1.568	66	204	254	116
11	728	34	94	122	62
12	320	18	43	55	30
13	146	10	20	26	16
14	48	6	9	9	6
15	26	4	5	5	4
16	6	3	3	1	1
17	4	2	2	1	1

Table 7.10: Number of darts and cells for a $40 \times 40 \times 40$ combinatorial map. A pyramid of 17 levels is built to reduce this map to its minimal form.

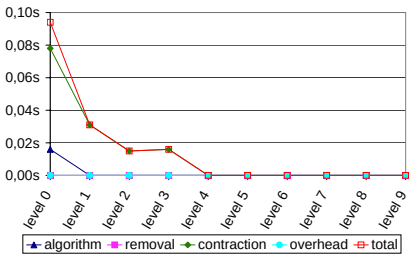
The size of the final level depends on the number of represented objects and their topological relations. For a simplex configuration this is a map representing two objects (the solid object and the background) and their adjacency. This combinatorial map corresponds to one of the minimal configurations of an edge presented in chapter 6.

In this case the top level map is identical to the configuration "edge 2". It consists of only four darts encoding 2 vertices, 2 edges, 1 face, and 1 volume. So the reduction factor for this pyramid is $1.608.672/4 \approx 400.000$.

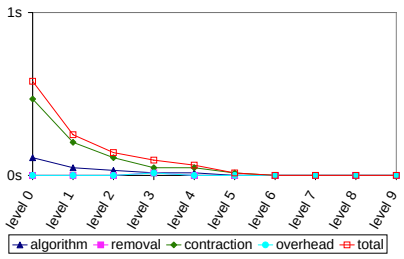
7.3.2 Bottom-up Construction Times

The bottom-up construction times of 3D combinatorial pyramids are measured in a similar way as the memory allocation. Pyramids are built for initial maps of different sizes. Since each of these maps represents the same simplex configuration, the build times should depend only on the size of the first level.

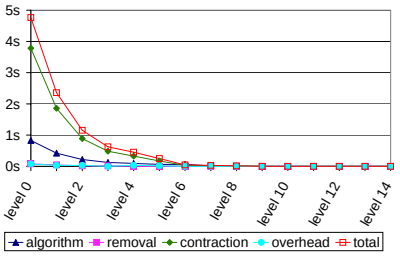
Table 7.11 lists the build times for four different pyramids. The sizes of the initial maps are $5 \times 5 \times 5$, $10 \times 10 \times 10$, $20 \times 20 \times 20$ and $40 \times 40 \times 40$. For



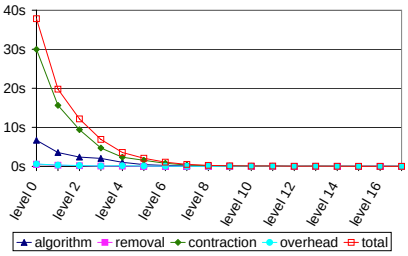
a) $5 \times 5 \times 5$



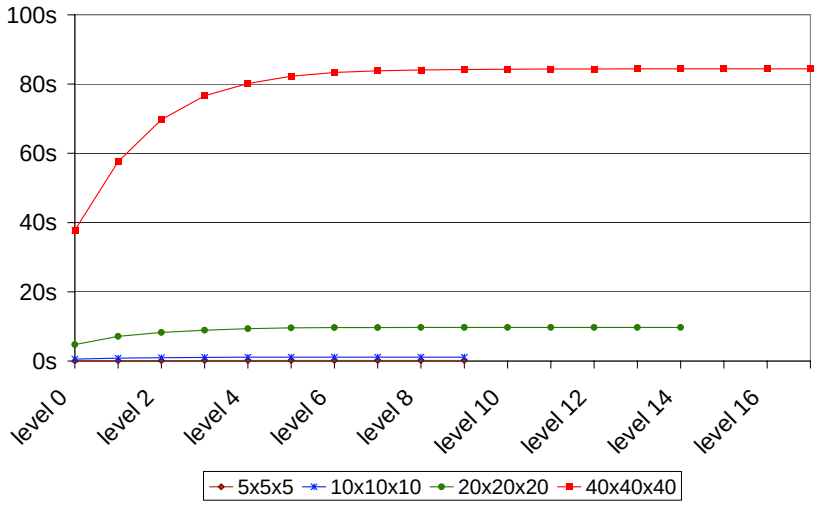
b) $10 \times 10 \times 10$



c) $20 \times 20 \times 20$



d) $40 \times 40 \times 40$



e) total times per level for pyramids of different size

Figure 7.9: Detailed bottom-up construction times for combinatorial pyramids with an initial map of different sizes (a-d). The total time (given in seconds) to construct a new level is specified as well as the times for each step of the algorithm. (e) gives a comparison of the accumulated build times for these pyramids.

each level and pyramid two measured times are presented: the total build time accumulated up to this level and the time needed to create the new level alone.

level	$5 \times 5 \times 5$	$10 \times 10 \times 10$	$20 \times 20 \times 20$	$40 \times 40 \times 40$
0	0,094s (0,094s)	0,578s (0,578s)	4,766s (4,766s)	37,812s (37,812s)
1	0,125s (0,031s)	0,828s (0,250s)	7,125s (2,359s)	57,578s (19,766s)
2	0,140s (0,015s)	0,968s (0,140s)	8,281s (1,156s)	69,734s (12,156s)
3	0,156s (0,016s)	1,062s (0,094s)	8,906s (0,625s)	76,625s (6,891s)
4	0,156s (0,000s)	1,125s (0,063s)	9,359s (0,453s)	80,187s (3,562s)
5	0,156s (0,000s)	1,140s (0,015s)	9,609s (0,250s)	82,265s (2,078s)
6	0,156s (0,000s)	1,140s (0,000s)	9,656s (0,047s)	83,343s (1,078s)
7	0,156s (0,000s)	1,140s (0,000s)	9,687s (0,031s)	83,843s (0,500s)
8	0,156s (0,000s)	1,140s (0,000s)	9,702s (0,015s)	84,093s (0,250s)
9	**0,156s (0,000s)**	**1,140s (0,000s)**	9,702s (0,000s)	84,218s (0,125s)
10			9,702s (0,000s)	84,296s (0,078s)
11			9,702s (0,000s)	84,359s (0,063s)
12			9,702s (0,000s)	84,374s (0,015s)
13			9,702s (0,000s)	84,405s (0,031s)
14			**9,702s (0,000s)**	84,421s (0,016s)
15				84,421s (0,000s)
16				84,421s (0,000s)
17				**84,421s (0,000s)**

Table 7.11: Bottom-up times of combinatorial pyramids of different sizes. For each pyramid and level two times are specified: The accumulated total build time up to this level and the time for the level alone (in brackets).

These times show that the construction of the first levels takes up most of the total built time. For all four pyramids the first level alone is responsible for ∼50% of the total time. For the $5 \times 5 \times 5$ pyramid ∼90% of the entire reduction process is spent at the first two levels. The same percentage is reached after four levels for the $40 \times 40 \times 40$ pyramid. This corresponds to the sizes of the pyramids as shown in the previous section.

Fig. 7.9a-d present diagrams of detailed times for each pyramid. These diagrams contain the times for each level of the pyramid given as the total time and the different steps of the algorithm used to build each level.

It can be seen that the ratio of times used by each of the steps remains the same throughout all levels of the pyramids. Simplifying each level with contraction operations accounts for ∼75% of the total time. The algorithm step uses ∼20% and the simplification with removal operations as well as the overhead (deletion of temporary maps and allocation of memory) is neglectable.

Fig. 7.9e compares the accumulated total bottom-up construction times of all four pyramids. Again it can be seen, that most of the total processing time is reached after the first few levels have been built. Thus the processing times seem to be mainly dependent on the size of the maps.

In order to reduce the time necessary to build a combinatorial pyramid, an important step is to deal with the construction times of the first levels. One possible optimization utilizes the fact that the base level of a pyramid is always grid-like. Therefore it is not necessary to completely build these initial maps each time. Grids for maps of a size that is often used can be saved in a pre-built form instead. This way only the actual values of the processed data must be associated with the vertices within the grid. The time needed to construct the base level is then reduced to the time necessary to read the pre-defined grid from a file and linking $n_1 \times n_2 \times n_3$ values to vertices.

As the construction of the first level accounts for $\sim 50\%$ of the total time, such an optimization of the base level can significantly reduce the bottom-up contruction time of the complete combinatorial pyramid.

Another aspect of efficient algorithms in terms of processing time is the possibility to use parallel computation. As [Brun and Kropatsch, 1999b] have shown, parallel algorithms can be designed for 2D combinatorial maps using contraction and removal kernels. Although this report does not provide a formal prove, it is expected that these results also apply to the extensions for 3D presented here.

Studying the optimization of the base level construction as well as the use of parallel algorithms will be the subject of further works based on this report.

Chapter 8

Conclusion and Outlook

Within this report the properties of minimal 3D combinatorial maps are studied with a strong focus on the process of creating minimal maps, as well as methods for identifying and distinguishing the topological relations present within volumetric data. For this purpose the definition of combinatorial maps as defined by [Brun and Kropatsch, 1999a, Damiand, 2001] is used with the extensions to 3D by [Braquelaire et al., 2003].

8.1 Minimal 3D Combinatorial Maps

For these combinatorial maps, operations are defined to produce a minimal 3D combinatorial map that preserves adjacency and inclusion relations. This is done by merging neighboring voxels that belong to the same component using edge contractions. Additionally the elimination of redundant cells is studied.

For this task, six specific operations that are based on the contraction and removal operations described by [Brun and Kropatsch, 1999a, Grasset-Simon et al., 2005] are defined in chapter 4. Edge contraction is used to merge vertices that represent voxels of the same component. The resulting maps contain redundant elements in the form of parallel edges and faces as well as empty edge- and face self-loops. Parallel cells are removed by the use of face and volume contractions while empty self-loops are eliminated with edge and face removals.

10 formal conditions are formulated in chapter 4 that specify if these operations may be applied. Using these conditions ensures that the resulting

combinatorial map will be consistent, valid, and that the information about adjacencies and inclusions is preserved. Using the Euler Characteristic It is shown that, if the conditions are fulfilled, the operations used to simplify the combinatorial maps do not change the genus of the bounding surfaces.

Based on the defined operations and their conditions an algorithm for building 3D combinatorial pyramids is constructed in chapter 5. Pyramids built with this algorithm contain a reduced 3D combinatorial map with respect to the 10 conditions as their top level.

Finally it is shown in chapter 6 that any 3D combinatorial map provided that does not contain face self-loops can be reduced to a minimal 3D combinatorial map. A minimal 3D combiantorial map is a combinatorial map in which each connected component is encoded by a single vertex.

Chapter 7 demonstrates the creation of minimal 3D combinatorial maps for four representative configurations (simplex, tunnel, hole, and two rings). In each case the initial combinatorial map has been successfully reduced to a minimal map.

The first conclusion is therefore that 3D combinatorial maps can be reduced to their minimal form where each connected component is represented by a single vertex while preserving information about adjacency relations and inclusions.

8.2 Minimal Configurations and Pseudo Elements

In the second part of this report minimal representations within minimal configurations are examined. A configuration is considered minimal if the number of cells contained in the configuration is minimal. It is shown in chapter 6 that more than one minimal configuration exists to represent edges and volumes. Based on these results the three topological relations adjacency, surrounds and contains are discussed.

Although these relations have been found to have multiple minimal representations as well, each can be unambiguously identified by their pseudo elements. Algorithms for this detection of topological relations are presented in chapter 6.

The reduced maps of four configurations examined in the experimental results contain the correct pseudo elements for the encoded topological

relations of the input data. This verifies that the maps were correctly reduced without destroying the topological structure and shows the usefulness of pseudo elements for the identification process.

Thus the second conclusion of this report is the existence of multiple minimal representations for i-cells and topological relations and the possibility to interpret the topological structure of the data based on pseudo elements.

8.3 Minimal Combinatorial Maps for analyzing 3D Data

Chapter 5 defines methods for calculating upper bounds specifying the size of initial 3D combinatorial maps. These numbers show that the initial grid-like maps are very large when compared to the original data. This results in a high demand on memory and other resources when building the base level of a pyramid as is confirmed by the experimental results.

At the same time it is shown that the size of a minimal maps depend only on the number of represented connected components. Minimal maps will contain exactly one vertex for each represented connected component and one additional vertex representing the background. This is confirmed by the last set of experiments. Initial maps of different sizes that represent the same configuration were all reduced to a final map of only 4 darts containing one vertex for each connected component.

The third conclusion is therefore the potential of working with minimal 3D combinatorial maps shown by these results. Due to their reduced size they are optimized in terms of memory and time needed to process them while at the same time providing all information about the structure of the data.

8.4 Outlook

The process of building a pyramid to minimize a 3D combinatorial map is also studied by the second set of experiments. In each case the creation of the first levels of the pyramid was mainly responsible for the total computation time.

The C++ library used for the experiments of this report is not capable of processing 3D data in real time (when compared to typical definitions of real time like 25 frames per second and a resolution of 640 by 480). However, depending on the demands of the application it is sufficient for processing data off line. Still, one of the future fields of research will be the optimization of the simplification process with a focus on the first levels of the pyramid.

Another focus of further studies will lie on the process of building the pyramids, motivated by the existence of multiple minimal representations for i-cells and topological relations within minimal combinatorial maps. This suggests that the order of applying the operations used to reduce the maps will produce different, but topologically equivalent, minimal maps. Studying different strategies for applying the operations in a certain order with the goal of finding an optimal strategy will therefore be of interest.

This report has a particular focus on defining operations and conditions to minimize 3D combinatorial maps while maintaining the topological relations. These operations and the resulting maps have been studied in the examples of small representative configurations. Experimenting with larger amounts of data like movies or the output from 3D scanners will be the next step toward using 3D combinatorial maps in the field of image analysis.

This report shows that minimizing 3D combinatorial maps is possible and the potential of this concept. Therefore the research of algorithms based on 3D combinatorial maps is of further interest. As a first step, successful 2D segmentation algorithms (e.g. [Haxhimusa et al., 2005b]) can be extended to work on volumetric data. Finally, 3D combinatorial pyramids can provide the basis for new algorithms as proposed by [Ion et al., 2005] as they allow for the advantages of an irregular pyramid while giving the possibility to access information about the topological structure of volumetric data.

Chapter 9

Acknowledgments

First and foremost I would like to thank Prof. W. G. Kropatsch for his contributions and guidance as the supervisor of the diploma thesis.

I also want to thank DI. Adrian Ion, Dr. Yll Haxhimusa, and Dr. Samuel Peltier for their contribution and help to this work as well as the Computer Graphics Group of the Institute of Computer Graphics and Algorithms of the University of Technology Vienna, especially Stefan Bruckner, for providing the software "raybooster" that was used to visualize the experimental results.

The support of the K plus Competence Center ADVANCED COMPUTER VISION is greatfully acknowledged.

This work was partially supported by the Austrian Science Fund under grants S9103-N04 and P18716-N13.

Bibliography

[Bertrand et al., 2000] Bertrand, Y., Damiand, G., and Fiorio, C. (2000). Topological Encoding of 3D Segmented Images. In Borgefors, G., Nyström, I., and S. di Baja, G., editors, *International Conference on Discrete Geometry for Computer Imagery*, volume 1953 of *Lecture Notes in Computer Science*, pages 311–324. Springer-Verlag, Germany.

[Borůvka, 1926] Borůvka, O. (1926). O jistém problému minimálnim. *Práce Mor. Přírodvěd. Spol. v Brně (Acta Societ. Scienc. Natur. Moravicae)*, 3(3):37–58.

[Braquelaire et al., 2003] Braquelaire, A., Damiand, G., Domenger, J.-P., and Vidil, F. (2003). Comparison and convergence of two topological models for 3d image segmentation. In *Workshop on Graph-Based Representations in Pattern Recognition*, number 2726 in Lecture Notes in Computer Science, pages 59–70, York, England.

[Bruckner, 2004] Bruckner, S. (2004). Efficient volume visualization of large medical datasets. Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria.

[Brun and Kropatsch, 2003] Brun, L. and Kropatsch, W. (2003). Receptive fields within the combinatorial pyramid framework. *Graph. Models*, 65(1-3):23–42.

[Brun and Kropatsch, 2005] Brun, L. and Kropatsch, W. (2005). Inside and outside within combinatorial pyramids. In Brun, L. and Vento, M., editors, *5th IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition*, Lecture Notes in Computer Science, Poitiers (France). IAPR-TC15, Springer, Berlin Heidelberg, New York.

[Brun and Kropatsch, 1999a] Brun, L. and Kropatsch, W. G. (1999a). Dual Contraction of Combinatorial Maps. Technical Report PRIP-TR-54, In-

stitute f. Computer Aided Automation 183/2, Pattern Recognition and Image Processing Group, TU Wien, Austria. Also available through http://www.prip.tuwien.ac.at/ftp/pub/publications/trs/tr54.ps.gz.

[Brun and Kropatsch, 1999b] Brun, L. and Kropatsch, W. G. (1999b). Pyramids with Combinatorial Maps. Technical Report PRIP-TR-57, Institute f. Computer Aided Automation 183/2, Pattern Recognition and Image Processing Group, TU Wien, Austria. Also available through http://www.prip.tuwien.ac.at/ftp/pub/publications/trs/tr57.ps.gz.

[Brun and Kropatsch, 2000] Brun, L. and Kropatsch, W. G. (2000). Irregular Pyramids with Combinatorial Maps. In Ferri, F. J., Iñesta, J. M., Amin, A., and Pudil, P., editors, *Advances in Pattern Recognition, Joint IAPR International Workshops on SSPR'2000 and SPR'2000*, volume 1876 of *Lecture Notes in Computer Science*, pages 256–265, Alicante, Spain. Springer, Berlin Heidelberg, New York.

[Damiand, 2001] Damiand, G. (2001). *Définition et étude d'un modèle topologique minimal de représentation d'images 2d et 3d.* Thèse de doctorat, Université Montpellier II.

[Damiand and Lienhardt, 2003] Damiand, G. and Lienhardt, P. (2003). Removal and contraction for n-dimensional generalized maps. In *DGCI*, pages 408–419.

[Damiand and Resch, 2002] Damiand, G. and Resch, P. (2002). Topological Map Based Algorithms for 3D Image Segmentation. In Lachaud, I. J. O., Braquelaire, A., and Vialard, A., editors, *DGCI 2002 Lecture Notes in Computer Science, 2301*, pages 220–231, France. Springer Verlag.

[Damiand and Vidil, 2006] Damiand, G. and Vidil, F. (2006). Moka - modeleur de cartes. http://www.sic.sp2mi.univ-poitiers.fr/moka/.

[Edmonds, 1960] Edmonds, J. (1960). A combinatorial representation of polyhedral surfaces. In *Notices of the American Mathematical Society*, volume 7, page 646.

[Felzenszwalb and Huttenlocher, 1998] Felzenszwalb, P. F. and Huttenlocher, D. P. (June 1998). Image Segmentation Using Local Variation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 98–104.

[Grasset-Simon et al., 2005] Grasset-Simon, C., Damiand, G., and Lienhardt, P. (2005). Pyramids of n-dimensional generalized map. In *Workshop*

on Graph-Based Representations in Pattern Recognition, number 3434 in Lecture Notes in Computer Science, pages 142–152, Poitiers, France.

[Grimm et al., 2004a] Grimm, S., Bruckner, S., Kanitsar, A., and Gröller, M. E. (2004a). Memory efficient acceleration structures and techniques for cpu-based volume raycasting of large data. In *Proceedings of the IEEE Symposium on Volume Visualization and Graphics 2004*, pages 1–8.

[Grimm et al., 2004b] Grimm, S., Bruckner, S., Kanitsar, A., and Gröller, M. E. (2004b). A refined data addressing and processing scheme to accelerate volume raycasting. *Computers & Graphics*, 28(5):719–729.

[Haxhimusa et al., 2005a] Haxhimusa, Y., Ion, A., Kropatsch, W. G., and Brun, L. (2005a). Hierarchical Image Partitioning using Combinatorial Maps. In Chetverikov, D., Czuni, L., and Vincze, M., editors, *Joint Hungarian-Austian Conference on Proceedings on Image Processing and Pattern Recognition, HACIPPR 2005 - OAGM 2005/KPAF 2005*, pages 179–186, Veszprm, Hungary. OCG.

[Haxhimusa et al., 2005b] Haxhimusa, Y., Ion, A., Kropatsch, W. G., and Illetschko, T. (2005b). Evaluating minimum spanning tree based segmentation algorithms. In Gagalowicz, A. and Philips, W., editors, *Proceedings of the 11th International Conference on Computer Analysis of Images and Patterns*, volume 3891 of *Lecture Notes in Computer Science*, pages 579–586, France. Springer.

[Illetschko, 2006] Illetschko, T. (2006). Minimal Combinatorial Maps for analyzing 3D Data. Master's thesis, Pattern Recognition and Image Processing Group, Institute of Computer Aided Automation, Vienna University of Technology.

[Illetschko et al., 2006a] Illetschko, T., Ion, A., Haxhimusa, Y., and Kropatsch, W. G. (2006a). Collapsing 3d combinatorial maps. In F. Lenzen, O. S. and Vincze, M., editors, *Proceedings of 30th Austrian Association for Pattern Recognition Workshop*, pages 85–93. Oesterreichische Computer Gesellschaft.

[Illetschko et al., 2006b] Illetschko, T., Ion, A., Haxhimusa, Y., and Kropatsch, W. G. (2006b). Effective programming of combinatorial maps using coma - a C++ framework for combinatorial maps. Technical Report PRIP-TR-106, Institute f. Computer Aided Automation 183/2, Pattern Recognition and Image Processing Group, TU Wien, Austria.

[Ion et al., 2005] Ion, A., Haxhimusa, Y., and Kropatsch, W. G. (2005). A Graph-Based Concept for Spatiotemporal Information in Cognitive Vision. In Brun, L. and Vento, M., editors, *5th IAPR-TC15 Workshop on Graph-based Representation in Pattern Recognition*, volume 3434 of *Lecture Notes in Computer Science*, pages 223–232, Poitiers, France. Springer, Berlin Heidelberg, New York.

[Klette and Rosenfeld, 2004] Klette, R. and Rosenfeld, A. (2004). *Digital Geometry*. Morgan Kaufmann.

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

[Kropatsch, 1995] Kropatsch, W. G. (1995). Equivalent Contraction Kernels and The Domain of Dual Irregular Pyramids. Technical Report PRIP-TR-42, Institute f. Automation 183/2, Dept. for Pattern Recognition and Image Processing, TU Wien, Austria. Also available through http://www.prip.tuwien.ac.at/ftp/pub/publications/trs/tr42.ps.gz.

[Lienhardt, 1989] Lienhardt, P. (1989). Subdivisions of n-dimensional spaces and n-dimensional generalized maps. In Mehlhorn, K., editor, *Proceedings of the 5th Annual Symposium on Computational Geometry (SCG '89)*, pages 228–236, Saarbrücken, FRG. ACM Press.

[Lienhardt, 1991] Lienhardt, P. (1991). Topological Models for Boundary Representation: A Comparison with n-dimensional Generalized Maps. *Computer-Aided Design*, 23(1):59–82.