Lecture Notes in Computational Vision and Biomechanics 2

Valentin E. Brimkov
Reneta P. Barneva *Editors*

Digital Geometry Algorithms

Theoretical Foundations and Applications to Computational Imaging

Springer

Lecture Notes in Computational Vision and Biomechanics

Editors

João Manuel R.S. Tavares
R.M. Natal Jorge

Address:
Faculdade de Engenharia
Universidade do Porto
Rua Dr. Roberto Frias, s/n
4200-465 Porto
Portugal
tavares@fe.up.pt, rnatal@fe.up.pt

Editorial Advisory Board

For further volumes:
http://www.springer.com/series/8910

Lecture Notes in Computational Vision and Biomechanics
Volume 2

The research related to the analysis of living structures (Biomechanics) has been a source of recent research in several distinct areas of science, for example, Mathematics, Mechanical Engineering, Physics, Informatics, Medicine and Sport. However, for its successful achievement, numerous research topics should be considered, such as image processing and analysis, geometric and numerical modelling, biomechanics, experimental analysis, mechanobiology and enhanced visualization, and their application to real cases must be developed and more investigation is needed. Additionally, enhanced hardware solutions and less invasive devices are demanded.

On the other hand, Image Analysis (Computational Vision) is used for the extraction of high level information from static images or dynamic image sequences. Examples of applications involving image analysis can be the study of motion of structures from image sequences, shape reconstruction from images and medical diagnosis. As a multidisciplinary area, Computational Vision considers techniques and methods from other disciplines, such as Artificial Intelligence, Signal Processing, Mathematics, Physics and Informatics. Despite the many research projects in this area, more robust and efficient methods of Computational Imaging are still demanded in many application domains in Medicine, and their validation in real scenarios is matter of urgency.

These two important and predominant branches of Science are increasingly considered to be strongly connected and related. Hence, the main goal of the LNCV&B book series consists of the provision of a comprehensive forum for discussion on the current state-of-the-art in these fields by emphasizing their connection. The book series covers (but is not limited to):

- Applications of Computational Vision and Biomechanics
- Biometrics and Biomedical Pattern Analysis
- Cellular Imaging and Cellular Mechanics
- Clinical Biomechanics
- Computational Bioimaging and Visualization
- Computational Biology in Biomedical Imaging
- Development of Biomechanical Devices
- Device and Technique Development for Biomedical Imaging
- Digital Geometry Algorithms for Computational Vision and Visualization
- Experimental Biomechanics
- Gait & Posture Mechanics
- Multiscale Analysis in Biomechanics
- Neuromuscular Biomechanics
- Numerical Methods for Living Tissues
- Numerical Simulation
- Software Development on Computational Vision and Biomechanics
- Grid and High Performance Computing for Computational Vision and Biomechanics
- Image-based Geometric Modeling and Mesh Generation
- Image Processing and Analysis
- Image Processing and Visualization in Biofluids
- Image Understanding
- Material Models
- Mechanobiology
- Medical Image Analysis
- Molecular Mechanics
- Multi-modal Image Systems
- Multiscale Biosensors in Biomedical Imaging
- Multiscale Devices and Biomems for Biomedical Imaging
- Musculoskeletal Biomechanics
- Sport Biomechanics
- Virtual Reality in Biomechanics
- Vision Systems

Valentin E. Brimkov • Reneta P. Barneva
Editors

Digital Geometry Algorithms

Theoretical Foundations and Applications to Computational Imaging

Springer

Editors
Valentin E. Brimkov
Department of Mathematics
Buffalo State College
State University of New York
Buffalo, NY
USA

Reneta P. Barneva
Department of Computer and
Information Sciences
State University of New York at Fredonia
Fredonia, NY
USA

ISSN 2212-9391 ISSN 2212-9413 (electronic)
Lecture Notes in Computational Vision and Biomechanics
ISBN 978-94-007-4173-7 ISBN 978-94-007-4174-4 (eBook)
DOI 10.1007/978-94-007-4174-4
Springer Dordrecht Heidelberg New York London

Library of Congress Control Number: 2012939722

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Digital geometry is a modern mathematical discipline studying the geometric properties of digital objects (usually modeled by sets of points with integer coordinates) and providing methods for solving various problems defined on such objects. Digital geometry is developed with the explicit goal to provide rigorous mathematical foundations and basic algorithms for applied disciplines such as computer graphics, medical imaging, pattern recognition, image analysis and processing, computer vision, image understanding, and biometrics. These are in turn applicable to important and societally sensitive areas like medicine, defense, and security.

Although digital geometry has its roots in several classical disciplines (such as graph theory, topology, number theory, and Euclidean and analytic geometry), it was established as an independent subject only in the last few decades. Several researchers have played a pioneering role in setting the foundations of digital geometry. Notable among these is the late Azriel Rosenfeld and his seminal works from the late 60's and early 70's of the last century. Some authors of chapters of the present book are also among the founders of the area or its prominent promoters. The last two decades feature an increasing number of active contributors throughout the world. A number of excellent monographs and hundreds of research papers have been devoted to the subject. One can legitimately say that at present digital geometry is an independent subject with its own history, vibrant international community, regular scientific meetings and events, and, most importantly, serious scientific achievements.

This contributed book contains thirteen chapters devoted to different (although interrelated) important problems of digital geometry, algorithms for their solution, and various applications. All authors are well-recognized researchers, as some of them are world leaders in the field. As a general framework, each chapter presents a research topic of considerable importance, provides a review of fundamental results and algorithms for the considered problems, presents new unpublished results, as well as a discussion on related applications, current developments and perspectives. By its structure and content, this publication does not appear to be an exhaustive source of information for all branches of digital geometry. Rather, the book is aimed at attracting readers' attention to central digital geometry tasks and related

applications, as diverse as creating image-based metrology, proposing new tools for processing multidimensional images, studying topological transformations for image processing, and developing algorithms for shape analysis.

An advantage of the chosen contributed book framework is that all chapters provide enough complete presentations written by leading experts on the considered specific matters. The chapters are self-contained and can be studied in succession dictated by the readers' interests and preferences.

We believe that this publication would be a useful source of information for researchers in digital geometry as well as for practitioners in related applied disciplines. It can also be used as a supplementary material or a text for graduate or upper level undergraduate courses.

We would like to thank all those who made this publication possible. We are indebted to João Manuel R.S. Tavares and Renato Manuel Natal Jorge, editors of the Springer's series "Lecture Notes in Computational Vision and Biomechanics," for inviting us to organize and edit a volume of the series. We are thankful to Springer's Office and particularly to Ms. Nathalie Jacobs, Senior Publishing Editor, and Dr. D. Merkle, Editorial Director, for reviewing our proposal and giving us the opportunity to publish this work with Springer, as well as for the pleasant cooperation throughout the editorial process. Lastly and most importantly, our thanks go to all authors who contributed excellent chapters to this book.

Fredonia and Buffalo, NY, USA

Valentin E. Brimkov
Reneta P. Barneva

Contents

Contributors

Xiang Bai Huazhong University of Science and Technology, Wuhan, China

Gilles Bertrand Laboratoire d'Informatique Gaspard-Monge, Équipe A3SI, Université Paris-Est, ESIEE Paris, Marne-la-Vallée, France

Bhargab B. Bhattacharya Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India

Partha Bhowmick Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India

Alfred M. Bruckstein Ollendorff Professor of Science, Computer Science Department, Technion, IIT, Haifa, Israel

David Coeurjolly LIRIS, UMR CNRS 5205, Université de Lyon, Villeurbanne, France

Lidija Čomić Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia

Michel Couprie Laboratoire d'Informatique Gaspard-Monge, Équipe A3SI, Université Paris-Est, ESIEE Paris, Marne-la-Vallée, France

Guillaume Damiand LIRIS, UMR5205, Université de Lyon, CNRS, Lyon, France

Leila De Floriani Department of Computer Science, University of Genoa, Genoa, Italy

Alexandre Dupas Unit 698, Inserm, Paris, France

Fabien Feschet IGCNC - EA 2782, Clermont Université, Université d'Auvergne, Clermont-Ferrand, France

Gabor T. Herman Computer Science Ph.D. Program, Graduate Center, City University of New York, New York, NY, USA

Atsushi Imiya Institute of Media and Information Technology, Chiba University, Chiba, Japan

Péter Kardos Institute of Informatics, University of Szeged, Szeged, Hungary

T. Yung Kong Computer Science Department, Queens College, City University of New York, Flushing, NY, USA

Jacques-Olivier Lachaud LAMA, UMR CNRS 5127, University of Savoie, Le Bourget du Lac, France

Longin Jan Latecki Temple University, Philadelphia, PA, USA

Quannan Li University of California, Los Angeles, CA, USA

Wenyu Liu Huazhong University of Science and Technology, Wuhan, China

Tianyang Ma Temple University, Philadelphia, PA, USA

Gábor Németh Institute of Informatics, University of Szeged, Szeged, Hungary

Lucas M. Oliveira Computer Science Ph.D. Program, Graduate Center, City University of New York, New York, NY, USA

Kálmán Palágyi Institute of Informatics, University of Szeged, Szeged, Hungary

Tristan Roussillon LIRIS, UMR CNRS 5205, Université de Lyon, CNRS, Villeurbanne, France

Kosuke Sato School of Science and Technology, Chiba University, Chiba, Japan; Information Technology Systems Dept. Intelligent Transport Systems Engineering Section, Mitsubishi Electric Corporation Kamakura Works, Kamakura, Kanagea, Japan

Antoine Vacavant Clermont Université, Université d'Auvergne, ISIT, CNRS, UMR6284, Clermont-Ferrand, France

Peter Veelaert Ghent University, Ghent, Belgium

Chapter 1
Digital Geometry in Image-Based Metrology

Alfred M. Bruckstein

Abstract Interesting issues in digital geometry arise due to the need to perform accurate automated measurements on objects that are "seen through the eyes" of modern imaging devices. These devices are typically regular arrays of light sensors and they yield matrices of quantized probings of the objects being looked at. In this setting, the natural questions that may be posed are: how can we locate and recognize instances from classes of possible objects, and how precisely can we measure various geometric properties of the objects of interest, how accurately can we locate them given the limitations imposed upon us by the geometry of the sensor lattices and the quantization and noise omnipresent in the sensing process. Another interesting area of investigation is the design of classes of objects that enable optimal exploitation of the imaging device capabilities, in the sense of yielding the most accurate measurements possible.

1.1 Introduction

Scanned character recognition systems are by now working quite well, several companies emerged based on the need to do image based inspection for quality control in the semiconductor industry and, in general, automated visual inspection systems are by now widely used in many areas of manufacturing. In these important applications one often needs to perform precise geometric measurements based on images of various types of planar objects or shapes. Images of these shapes are provided by sensors with limited capabilities. These sensors are spatially arranged in regular planar arrays providing matrices of quantized pixel-values that need to be processed by automated metrology systems to extract information on the location, identity, size and orientation, texture and color of the objects being looked at. The geometry, spatial resolution and sensitivity of the sensor array are crucial factors in the measurement performances that are possible. When sensor arrays are regular planar grids, we have to deal with a wealth of issues involving geometry on the integer grid,

A.M. Bruckstein (✉)
Ollendorff Professor of Science, Computer Science Department, Technion, IIT, 32000 Haifa, Israel
e-mail: freddy@cs.technion.ac.il

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_1,

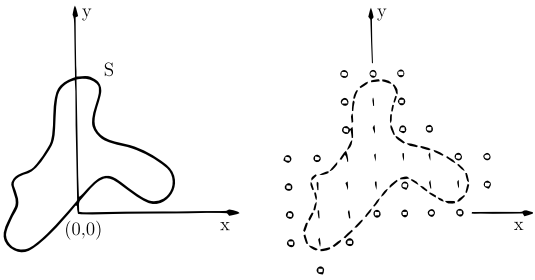


Fig. 1.1 Image digitization by point sampling on the unit grid $\mathbb{Z}^2$

hence digital geometry problems enter the picture in industrial metrology tasks in very fundamental ways.

1.2 The Digitization Model and the Metrology Tasks

We assume that planar shapes, the objects we are interested to locate, measure and recognize are binary (black on a white background) and live in the real plane, $\mathbb{R}^2$. Hence their full description can be given via an indicator function $\xi(x, y)$ which is 1 (black) if (x, y) is inside the shape and 0 (white) if (x, y) is in the background. The digitization process assumed will be point sampling on the integer grid, $\mathbb{Z}^2$, hence the result of digitization will be a discrete indicator function on the integer grid: a discrete binary image, or a zero/one matrix of picture elements, or pixels, see Fig. 1.1. The "generic problem" we deal with is: given the discretized shape, i.e.,

$$\xi_D(i, j) = \begin{cases} 1 & \text{if } \xi_D(i, j) = 1 \\ 0 & \text{if } \xi_D(i, j) = 0 \end{cases}$$

recover as much information as possible on the "pre-image", i.e., on the original binary shape that lives on the continuous real plane. The information on the pre-image shape that one needs might be its location and orientation, area, perimeter, etc. In order to solve the particular problem at hand we shall also need to exploit whatever prior information we may have on the continuous pre-images. This prior information sometimes defines the objects or shapes we digitize as members of parameterized sets of possible pre-images. For example, we might know that the shapes we are called upon to measure are circular with varying locations and sizes. In this case the parameter defining the particular object instance being analyzed from its digitization is a vector comprising three numbers: two coordinates pointing out the center of the disk and a positive number providing its radius. The digitized shape $\xi_D(i, j)$ then provides some information on the center and radius of the disk and we may ask how good an estimate can we get for these quantities given the data.

1.3 Self Similarity of Digital Lines

Digital lines result from point-sampling half-plane pre-images. More is known about the jagged boundaries obtained in this process topic than anyone can possibly know, but the basic facts are both simple and beautiful. Half-planes are not very interesting or practically useful objects, however they already pose the following metrology problem: given the digital image of a half-plane, locate it (i.e., its boundary line) as precisely as possible. Of course, we must ask ourselves whether and how our location estimation improves as we see more and more of the digitized boundary. We can think about the location estimation problem as a problem of determining the half-plane pre-images that satisfy all the constraints that the digitized image provides. Indeed every grid-point pixel that is 0 (white) will tell us that the half-plane does not cover that location while every black (1) pixel will indicate that the half-plane covers its position. It should come as no surprise that the boundary pixels, i.e., the locations where white pixels are neighboring black ones, carry all the information. The constraint that a certain location in the plane belongs, or does not belong to the half-plane that is being probed translates into a condition that the boundary line has a slope and intercept pair in a half-plane defined in the dual representation space (which is called in pattern recognition circles the Hough parameter plane). Therefore, as we collect progressively more data in the "image-plane" we have to intersect more and more half-planes in the Hough plane to get the so called "locale", or the uncertainty region in parameter space where the boundary line parameters lie, see [12, 18, 25]. Looking at the grid geometry and analyzing the lines that correspond to grid-points in the dual plane one quickly realizes that only the boundary points contribute to setting the limits of the locale of interest, and a careful analysis reveals that, due to the regularity of the sampling grid, the locales are always polygons of at most four sides, see [12, 25]. Hence as more and more consecutive boundary points are added to the pool of information on the digitized half plane, we have to perform half-plane intersections with at most four sided polygonal locales to update them. Clearly the locales generally strictly decrease in size as the number of points increases, and we can get exact estimates on the uncertainty behavior as the jagged boundary is progressively revealed. This idea, combining the geometry of locales for digital straight lines with the process of successively performing the half-plane intersections for each new data point while walking along the jagged digitized boundary, led to the simplest, and only recursive $O(\text{length})$ algorithm for detecting straight edge segments. A complete description of this algorithm is the subject of the next section of this paper.

The jagged edges that result from discretizing half-planes have a beautiful, self-similar structure, intimately related to the continued fraction representation of the real number that defines the slope of their boundary line. It is clear that at various sampling resolutions the boundary maintains its jaggedness in a fractal manner, but here we mean a different type of self-similarity, inherent in the jagged boundaries at any given resolution! A wealth of interesting and beautiful properties that were described over many years of research on digital straight lines follow from a very simple unifying principle: invariance of the linear separability property under re-encoding with respect to regular grids embedded into the integer lattice. Not only

does this principle help in re-discovering and proving in a very straightforward manner digital straight edge properties that were often arrived at and proved in sinuous ways, but it also points out all the self-similarity type properties that are possible, making nice connections to number-theoretic issues that arise in this context and the general linear group $GL(2, Z)$ that describes all integer lattice isomorphisms. Following [6], we next present the basic self-similarity results.

A digitized straight line is defined as the boundary of a linearly separable dichotomy of the set of points with integer coordinates, $\mathbb{Z}^2 = \{(i, j)|i, j \in \mathbb{Z}\}$, in the plane. The boundary points of the dichotomy induced by a line with slope m and intercept n, $y = mx + n$, are

$$L(m, n) = \big\{(i, h_i)|i \in \mathbb{Z}, h_i = \lfloor mi + n \rfloor\big\}.$$

Without loss of generality let us assume that $m > 0$, so that the sequence h_i is a nondecreasing sequence of integers. Associate to the set of boundary points $L(m, n)$ a string of two symbols, 0 and 1, coding the sequence of differences $h_{i+1} - h_i$, as follows

$$C(m, n) = \cdots C_{-2}C_{-1}C_0C_1C_2 \cdots = \prod_i C_j(m, n)$$

where

$$C_i(m, n) = \begin{cases} 0, & \text{if } h_{i+1} - h_i = 0, \\ 01^k, & \text{if } h_{i+1} - h_i = k, \end{cases}$$

and 1^k means 1 1 $\cdots$ 1 with k 1's. $C(m, n)$ is called the *chain-code* of the line $L(m, n)$. Note that the sequence $C(m, n)$ can be uniquely parsed into its components $C_i(m, n)$, since a separator must precede every 01 string and follow every run of 1's and each of the remaining 0's must be a single component, as well. Clearly, given some h_{i_0}-value and $C(m, n)$, the entire sequence h_i can be recovered. The graphical meaning of the chain-code associated to $L(m, n)$ is depicted in Fig. 1.2.

The set of points $L(m, n)$ uniquely determines the slope of the line m. Indeed, $L(m, n) \equiv L(m', n')$ implies $m = m'$, since otherwise the vertical distance between $y = mx + n$ and $y = m'x + n'$ would become unbounded at $x \to \pm\infty$, and their h_i sequences would differ starting at some large enough i. Furthermore, if m is irrational we have, by a classical result, that the vertical intercepts of $y = mx + n$ modulo 1 are dense in [0, 1]. For every $\varepsilon > 0$ there exist i_0 and j_0 such that

$$mi_0 + n - \lfloor mi_0 + n \rfloor < \varepsilon,$$

$$mj_0 + n - \lfloor mj_0 + n \rfloor > 1 - \varepsilon,$$

and changing n by ε would result in a change in $L(m, n)$. Therefore for irrational m, $L(m, n)$ uniquely determines both m and n. If m is rational there exist only a finite set of distinct vertical intercepts of $y = mx + n$ modulo 1, therefore n is determined only up to an interval and the length of the worst interval of uncertainly for n depends on the minimal p/q representation of m. This also proves that the chain-code $C(m, n)$ determines m uniquely, and if m is irrational, n is also determined by it modulo 1, since we clearly have $C(m, n) \equiv C(m, n + 1)$.

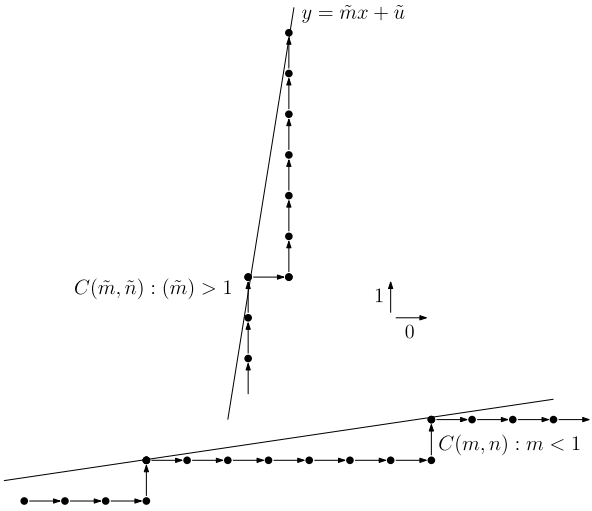


Fig. 1.2 Chain-codes of $L(m, n)$ for $m < 1$ and $\tilde{m} > 1$

From the definition of chain-codes $C(m, n)$ we obtain several immediate and basic properties a sequences of zeros and ones must have in order to be the chain-code of a straight edge. In the case of $m < 1$, the difference

$$h_{i+1} - h_i = \lfloor m(i+1) + n \rfloor - \lfloor mi + n \rfloor \tag{1.1}$$

can only be either 0 or 1. In this case the chain-code of a digitized line has runs of 0's separated by single 1's, and the 0's occur in runs with length determined by the number on integer coordinates that fall within the intervals determined on the x-axis by the points defined by

$$mx_i + n = i \in \mathbb{Z}, \quad \text{i.e.,} \quad x_i = \frac{i}{m} + \frac{n}{m}. \tag{1.2}$$

The intervals $[x_i, x_{i+1})$ have a constant length of $1/m$ and therefore the number of integer coordinates covered can be (see Fig. 1.3a) either $\rho_i = \lfloor 1/m \rfloor$ or $\rho_i = \lfloor 1/m \rfloor + 1$. Therefore, if $m < 1$, $C(m, n)$ is of the form

$$C(m, n) = \cdots 10^{\rho_1} 10^{\rho_2} 10^{\rho_3} 1 \cdots \tag{1.3}$$

where $\rho \in \{\lfloor 1/m \rfloor, \lfloor 1/m \rfloor + 1\}$. For the case $m > 1$, the difference $hi + 1 - h_i = \lfloor m(i+1) + n \rfloor - \lfloor mi + n \rfloor$ is always greater than 1, and therefore the chain-code $C(m, n)$ has runs of 1's separated by single 0's. Since $\lfloor m + mi + n \rfloor - \lfloor mi + n \rfloor$ equals the number of integer coordinates between the values $m(i+1)+n$ and $mi+n$ the run of 1's has length determined by the number of integral values in consecutive intervals of length m, see Fig. 1.3b. This shows that the run-lengths ρ_i in this case, will be either $\rho = \lfloor m \rfloor$ or $\lfloor m \rfloor + 1$. Therefore if $m > 1$, the chain-code $C(m, n)$ has the form

$$C(m, n) = \cdots 01^{\rho_1} 01^{\rho_2} 01^{\rho_3} 0 \cdots \tag{1.4}$$

with $\rho \in \{\lfloor m \rfloor, \lfloor m \rfloor + 1\}$.

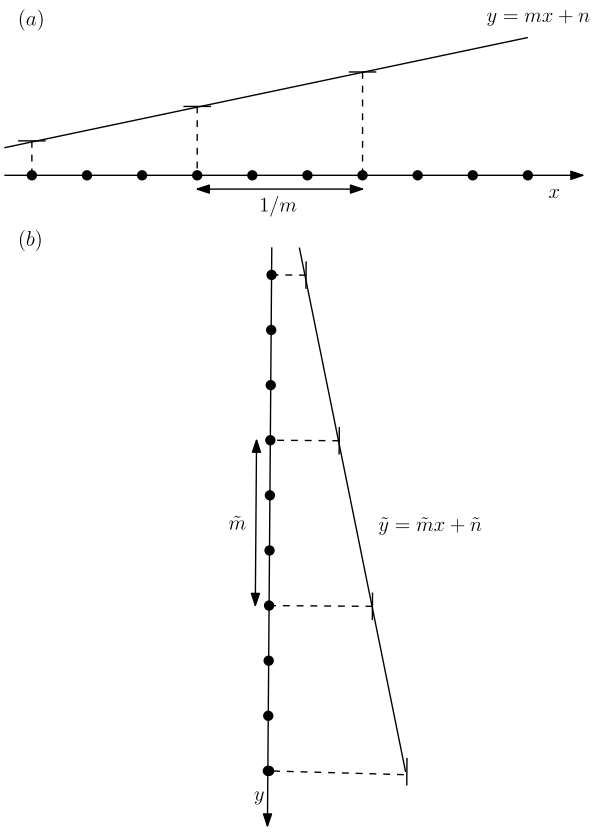


Fig. 1.3 Basic properties of chain-codes

The question that immediately arises is the following: is there any order or *pattern* in the appearance of the two values for the run length of the symbols 0 or 1, i.e., in the sequences $\{\rho_i\}$ that arise from "chain coding" digitized straight lines? The classical results on digital straight edges are focused on "uniformity properties" of the appearance of the separator symbol in the chain-codes sequences, see [14, 20]. We here briefly present a very high level and general uniformity result via self-similarity, as was first defined in [6].

Suppose we are given the chain-code of a digitized straight boundary $C(m, n)$. We know that $C(m, n)$ is a sequence composed of two symbols, 0 and 1, and that it looks either like (1.3) or (1.4), thus it has the general form

$$\cdots \Delta \square^{\rho_i} \Delta \square^{\rho_{i+1}} \Delta \square^{\rho_{i+2}} \Delta \cdots \tag{1.5}$$

where $\rho_i \in \{p, p+1\}$, $p \in \mathbb{Z}$, and Δ, $\square$ stand for either 0, 1 or 1, 0, respectively.

We can define several transformation rules on two symbol, or $\Delta/\square$, sequences of the type (1.5), transformations that yield new $\Delta/\square$ sequences.

RULE X. Interchange the symbols Δ and $\square$ (i.e., $\Delta \to \square$ and $\square \to \Delta$).

Application of **X** to a chain-code $C(m, n)$ yields a new sequence of symbols, with 0's replacing the 1's and 1's replacing the 0's of the original sequence.

RULE S. Replace every $\square\Delta$ sequence by Δ.

Application of the **S**-transformation to a chain-code of the form (1.5) yields a sequence of the same type with the run length ρ_i replaced by $\rho_i - 1$. Applying Rule **S**, p times yields the next transformation rule.

RULE S^p. Replace $\square^p\Delta$ by Δ and $\square^{p+1}\Delta$ by $\square\Delta$.

Notice that, in contrast to the transformation rules **X** and **S**, this rule depends on the $\{\rho_i\}$ sequence, i.e., it is adapted to the given pattern (1.5). Indeed we can apply the **S**-transformation successively at most p times where p is the minimal value of ρ_i's. After that, we need to do an **X** transformation in order to bring the sequence of symbols to the form (1.5).

RULE R. Replace $\square^p\Delta$ by Δ and $\square^{p+1}\Delta$ by $\square$.

We may view the action of **R** as a result of applying **S**^{p} first, then replacing $\square\Delta$ by $\square$. This rule is also adapted to the sequence on which it operates.

The next transformation rule is somewhat different, since it replaces symbols in a way that depends on the neighborhood or the "context".

RULE T. Replace $\square\Delta$ by $\square$ and the $\square$'s followed by a $\square$ by $\square\Delta$.

Application of rule **T** has the effect of putting a Δ between every consecutive pair of $\square$'s and removing all the Δ's appearing in the original sequence. For example the sequence

$$\cdots\square\Delta\square\square\square\Delta\square\square\Delta\square\square\Delta\square\cdots$$

will be mapped under **T**, to

$$\cdots\square\square\Delta\square\Delta\square\Delta\square\square\Delta\square\Delta\square\square\Delta\square\Delta\square\square\cdots$$

Up to this point the transformation rules were completely specified by rather simple local symbol replacement rules. The next two classes of transformation rules, require the setting of an initial position and a bilateral parsing for the generation of the transformed sequences.

V-RULES. Given the sequence of $\Delta\square$, choose a Δ symbol as an initial position, then to the right and to the left of the chosen Δ delete batches of $Q-1$ consecutive Δ's.

This transformation has the effect of joining together (from the starting position) Q consecutive $\square$-runs. The sequence

$$\Delta\square^{\rho_{i-Q}}\cdots\Delta\square^{\rho_{i-1}}\Delta_{\uparrow}\square^{\rho_i}\Delta\square^{\rho_{i-1}}\Delta\cdots\square^{\rho_{i+Q-1}}\Delta$$

will be mapped to

$$\cdots\Delta\square^{\rho_{i-Q}+\cdots+\rho_{i-1}}\Delta\square^{\rho_i+\rho_{i+1}+\cdots+\rho_{i+Q-1}}\Delta\cdots$$

if the Δ preceding $\square^{\rho_i}$ is chosen as the initial position. Therefore if a $\Delta/\square$-chain-code sequence of type (1.5) is specified by the $\square$-run length sequence $\{\rho_i\}_{i\in N}$ a **V**-transformation as defined above will produce a sequence of type (1.5) specified by $\{\rho_{i_0+nQ}+\rho_{i_0+nQ+1}+\cdots+\rho_{i_0+(n+1)Q-1}\}_{n\in N}$ for a given i_0 and a given integer $Q\geq 1$.

H-RULES. Given the sequence of $\Delta\Box$ symbols, choose a starting point between two consecutive symbols and parse the sequence to the right and left of the starting point, counting the number of $\Box$'s seen. After seeing P $\Box$'s, replace the subsequence by one $\Box$ followed by the number of Δ's encountered while accumulating the P $\Box$'s. In counting the Δ's encountered, apply the following rules: (1) when parsing to the right: if the P-th $\Box$ symbol is followed by a Δ count this Δ as well and start accumulating the next batch of P symbols after it and (2) when parsing to the left: if the P-th $\Box$ symbol is preceded by Δ do not count this Δ and start accumulating the next batch of P $\Box$'s immediately.
As an example, consider applying an **H**-transformation to the sequence below, with the indicated initial position,

$$\cdots\Box\Delta\Box\Box\Box\Delta\Box\Box\Box\Delta\uparrow\Box\Box\Box\Box\Delta\Box\Box\Box\Delta\Box\Box\Box\Delta\Box\Box\Box\Delta\Box\cdots$$

and parameter $P = 7$. We obtain the parsing

$$\cdots\uparrow\Box\Delta\Box\Box\Box\Delta\Box\Box\Box\Delta\uparrow\Box\Box\Box\Box\Delta\Box\Box\Box\Delta\uparrow\Box\Box\Box\Delta\Box\Box\Box\Delta\Box\uparrow\cdots$$

that yields the output

$$\cdots\uparrow\Box\Delta\Delta\Delta\uparrow\Box\Delta\Delta\uparrow\Box\Delta\Delta\uparrow\cdots$$

The same initial conditions with parameter $P = 3$ provide the parsing

$$\cdots\Box\Delta\uparrow\Box\Box\Box\Delta\uparrow\Box\Box\Box|\Delta\uparrow\Box\Box\Box\uparrow\Box\Delta\Box\Box\uparrow\Box\Delta\Box\Box\uparrow\Box\Delta\Box\Box\uparrow\Box\Delta\Box\cdots$$

and an output sequence

$$\cdots\uparrow\Box\Delta\uparrow\Box\Delta\uparrow\Box\uparrow\Box\Delta\uparrow\Box\Delta\uparrow\Box\Delta\uparrow\cdots$$

So far we have defined seven rules for transforming $\Delta/\Box$ sequences into new $\Delta/\Box$ sequences. The first five of them are uniquely specified in terms of local string replacement rules, the last two being classes of transformations that require the choice of an initial positions for parsing and are further specified by an arbitrarily chosen integer (Q or P). The main self similarity results are, [6]:

The Self-similarity Theorem

1. *Given a $\Delta\Box$ sequence of type* (1.5), *the new sequence produced by applying to it any of the transformations* **X**, **S**, $\mathbf{S}^p$, **R**, *or* **T**, *is the chain-code of a digitized straight line if and only if the original sequence was the chain-code of a digitized straight line*.
2. *If a $\Delta\Box$ sequence is the chain-code of a digitized straight line*, *then the sequences obtained from it by applying any transformation according to the* **H**-*rules*, *or* **V**-*rules*, *are also chain-codes of digitized straight lines*.

Note that, for the **X**-, **S**-, $\mathbf{S}^p$-, **R**-, or **T**-transformation rules we have stronger claims than for the classes of **H**- and **R**-rules. The reason for this will become obvious from the proof. The digital line properties stated above are *self-similarity* results since what we have is that a given chain-code pattern generates, under repeated applications of various transformation rules, new patterns in the same class: chain-codes of digitized straight lines.

Proof We argue that the chain-code transformations defined above are simply re-encodings of digitized straight lines on regular lattices of points, embedded into the integer lattice $\mathbb{Z}^2$. This observation, combined with the fact that the embedded lattices are generated by affine coordinate transformations, readily yield the results claimed. Indeed, choose any two linearly independent basis vectors B_1 and B_2 with integer entries and a lattice point (i_0, j_0) for the origin Ω_0. Define a regular embedded lattice of points as follows

$$\mathbf{E}^2 = \{(i_0, j_0) + i\,B_1 + j\,B_2 | (i, j) \in \mathbb{Z}^2\}.$$

A given straight line $y = mx + n$ defines a dichotomy of the points of $\mathbb{Z}^2$, but also of the points of $\mathbf{E}^2 \subset \mathbb{Z}^2$! If B_1 and B_2 are basis vectors, there exists an affine transformation that maps lattice $\mathbf{E}^2$ the embedding back into $\mathbb{Z}^2$, i.e., the point $(i_0, j_0) + i\,B_1 + j\,B_2 \in \mathbf{E}^2$ into $(i, j) \in \mathbb{Z}^2$, and the *same* transformation maps the line $y = mx + n$ into some new line $Y = MX + N$, on the transformed plane. The points $(i_0, j_0) + i\,B_1 + j\,B_2$ from the original (x, y)-plane map into (i, j), hence the transformation from (X, Y) into (x, y) is

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} i_0 \\ j_0 \end{pmatrix} + \left[B_1^T\, B_2^T\right] \begin{pmatrix} X \\ Y \end{pmatrix}$$

and therefore the inverse mapping from (x, y) to (X, Y) is

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \left[B_1^T\, B_2^T\right]^{-1} \left[\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} i_0 \\ j_0 \end{pmatrix} \right] = \mathbf{M} \begin{pmatrix} i_0 \\ j_0 \end{pmatrix}. \tag{1.6}$$

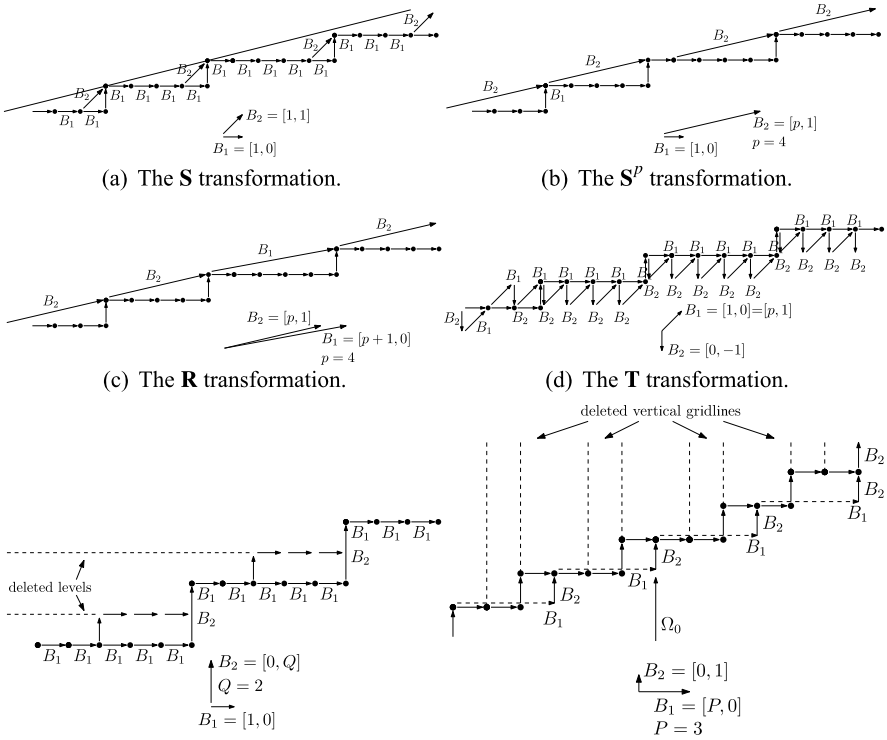
From these transformations the mapping of the line parameters (m, n) into (M, N) can also be readily obtained in terms of $B_1 B_2$ and Ω_0.

After performing the transformation (1.6) the line $Y = MX + N$ can be chain-coded with respect to the lattice $\mathbb{Z}^2$ (which is now the image of $\mathbf{E}^2$) and the resulting chain-code will somehow be related to the chain-code of $y = mx + n$ defined on the original grid $\mathbb{Z}^2$. The key observation, proving the results stated, is that the transformations introduced in the previous section represent straightforward re-encodings of digitized lines with respect to suitably chosen embedded lattices $\mathbf{E}^2$. The choices of basis vectors that lead to each of the sequence transformations we are concentrating on are shown in Fig. 1.4 and are analyzed in detail below:

1. The **X** transformation rule, the interchange of Δ and $\square$ symbols, is clearly accomplished by the coordinate-change mapping that takes (i, j) into (j, i). Here $B_1 = [0, 1]$ and $B_2 = [1, 0]$ and we have that $y = mx + n$ maps into $Y = (1/m)X - (n/m)$ under the transformation matrix $\mathbf{M}_X = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$.
2. The **S**-rule which reduces every integer of the $\{\rho_i\}$ sequence by 1 is induced by the mapping that considers a $\square$ step as a step in the $B_1 = [1, 0]$ direction, but a combined $\square\Delta$-step as the unit step in the $B_2 = [1, 1]$-direction (see Fig. 1.4a). Therefore, the **S**-transformation matrix is

$$\mathbf{M}_S = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

and $y = mx + n$ maps into $Y = Xm/(1 - m) + n/(1 - m)$.



(a) The **S** transformation. (b) The $\mathbf{S}^p$ transformation.

(c) The **R** transformation. (d) The **T** transformation.

(e) The **V** transformation. (f) The **H** transformation.

Fig. 1.4 The **S**, $\mathbf{S}^p$, **R**, **T**, **V** and **H** transformations

3. The adaptive $\mathbf{S}^p$-transformation rule which replaces $\Box^p \Delta$ by Δ, and $\Box^{p+1}\Delta$ by $\Box\Delta$, corresponds to choosing $B_1 = [1, 0]$ and $B_2 = [p, 1]$ (see Fig. 1.4b). The transformation matrix is

$$\mathbf{M}_{S^p} = \begin{bmatrix} 1 & p \\ 0 & 1 \end{bmatrix}^{-1} = \left(\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \cdots \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \right)^{-1} = \begin{bmatrix} 1 & -p \\ 0 & 1 \end{bmatrix}.$$

The line $y = mx + n$ is transformed into $Y = Xm/(1 - pm) + n/(1 - pm)$. Note that, if $m < 1$, $p = \lfloor 1/m \rfloor$ and we denote the fractional part of $1/m$ by $\alpha (= 1/m - \lfloor 1/m \rfloor)$, we have $m/(1 - mp) = 1/\alpha > 1$. This shows that one $\mathbf{S}^p$-transformation, that is adapted to the run-length of the $\Box$-symbols replaces the slope m with $(1/m - \lfloor 1/m \rfloor)^{-1}$. Therefore, repeated application of this adapted transformation followed by an X-transformation will produce a sequence of slopes recursively given by $m_k = 1/m_{k-1} - \lfloor 1/m_{k-1} \rfloor$, $m_0 = m$. Hence, the sequence of adapted "exponents" of the corresponding $\mathbf{S}^p$-transformations $p_k =$

$\lfloor 1/m_k \rfloor$, is the sequence of integers of the continued fraction representation of m_0,

$$m_0 = \cfrac{1}{p_0 + \cfrac{1}{p_1 + \cfrac{1}{p_2 + \cfrac{1}{\ddots}}}}$$

4. The transformation rule **R** maps $\square^p\Delta$ into Δ and $\square^{p+1}\Delta$ into $\square$. Therefore $B_1 = [p+1, 1]$ and $B_2 = [p, 1]$ (Fig. 1.4c). The transformation matrix is

$$\mathbf{M}_R = \begin{bmatrix} p+1 & p \\ 1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & -p \\ -1 & p+1 \end{bmatrix}$$

and an original line $y = mx + n$ is mapped into $Y = X[(p+1)m - 1]/(1 - pm) + m/(1 - pm)$. Note here that in terms of $\alpha = 1/m - \lfloor 1/m \rfloor$ the new slope is $(1-\alpha)/\alpha$, when $m < 1$.

5. The last of this class of transformations, Rule **T**, replaces $\square\Delta$ by $\square$'s, and $\square$'s followed by a $\square$ by $\square\Delta$'s. We may view this transformation as a sequence of two maps: the first one replacing $\square^{p+1}\Delta$ by $\square$, and $\square^p\Delta$ by Δ by the adapted rule **R**, the second replacing $\square$ by $\square\Delta\square\Delta\cdots\Delta\square$ with $(p+1)\square$'s, and Δ by $\square\Delta\square\Delta\cdots\Delta\square$ with $p\square$'s. This would imply that we first do an **R**-transformation via the matrix

$$\mathbf{M}_{RT} = \begin{bmatrix} p+1 & p \\ p & p-1 \end{bmatrix}.$$

Concatenating the two transformations we obtain

$$\mathbf{M}_T = \mathbf{M}_{RT}\mathbf{M}_R = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix},$$

which is not surprising. Indeed, $\square\Delta$ is mapped by $B_1 = [1\ 1]$ into one $\square$ step, but a $\square$ followed by another $\square$ will have to be mapped into a sequence of two steps, B_2B_1, the first one being $B_2 = [0, -1]$ (see Fig. 1.4d). We readily see from the $\mathbf{M}_T$ transformation that $y = mx + n$ maps into $Y = (1-m)X - n$. Therefore the slopes of the two lines add to 1. Indeed, "summing up" the two sequences in the sense of placing a Δ whenever there exists a Δ in either the original, or the **T**-transformed chain-code, we get the sequence $\cdots\square\Delta\square\Delta\square\Delta\square\cdots$, which represents the lines of the type $y = x + n$.

Up to this point, all the transformation matrices, whether adapted to the chain-code parameter p or not, were matrices with integer entries and had the property that $\det(\mathbf{M}) = \pm 1$. This implied that the matrices had inverses with integer entries and, as a consequence, the embedded lattice $\mathbf{E}^2$ was simply a "reorganization" or "relabeling" of the entire integer lattice $\mathbb{Z}^2$. In mathematical terms, unimodular lattice transformations are isomorphisms of the two dimensional lattice. The 2×2 integer matrices with determinant ± 1 (called unimodular matrices) form a well known group called $GL(2, \mathbb{Z})$ and this group is finitely gener-

ated by the matrices $\left[\begin{smallmatrix} 0 & 1 \\ +1 & 0 \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} -1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. For all such transformations (that are invertible within $GL(2,\mathbb{Z})$) the corresponding chain-code modification rules will yield chain-codes of linearly separable dichotomies, simply because the transformed line $Y = MX + N$ induces a linearly separable dichotomy of $\mathbf{E}^2$. Therefore the self-similarity results may be regarded as two different ways of stating that the points of the lattice $\mathbb{Z}^2$ are linearly separated by a given line $y = mx + n$. The first class of results presented above becomes obvious in this setting. Furthermore, from the fact that the group $GL(2,\mathbb{Z})$ is finitely generated, it follows that we have countably many sequence transformations, having the property that they yield chain-codes of straight lines if and only if the original chain-code is a digitized straight line, and they are expressible as products of sequences of basic transformations of the type **X**, **S**, and, say **T** (or one other transformation).

The situation is somewhat different for the remaining classes of transformation rules, the **V** and **H**-rules.

6. In the embedded lattice setting it is easy to see that a **V**-rule implies choosing some origin point Ω_0 and basis vectors of the form $B_1 = [1, 0]$, $B_2 = [0, Q]$ (see Fig. 1.4e). In this case, the set $\mathbf{E}^2$ is properly contained in $\mathbb{Z}^2$, i.e., $\mathbf{E}^2 \subset \mathbb{Z}^2$ and the mapping of Eq. (1.6) has fractional entries. Since **V**-rules imply a decimation of the horizontal grid lines, the fact that a chain-code of a digitized line provides a new digitized line, is obvious. However, due to the proper embedding of $\mathbf{E}^2$ into $\mathbb{Z}^2$ these results are not "if and only if" results any more. Indeed, we could start with a sequence like

$$\cdots \Delta\Box\Delta\Box^p\Delta\Box\Delta\Box^p\Delta\Box\Delta\cdots$$

and any **V**-transformation with $Q = 2$ will provide the transformed sequence

$$\Delta\Box^{p+1}\Delta\Box^{p+1}\Delta\Box^{p+1}\ldots$$

This sequence is obviously a digitized straight line while the original one is obviously not, for any $p > 2$. Hence, the proper embedding of $\mathbf{E}^2$ in $\mathbb{Z}^2$ implies that digital lines, but not only digital lines, map into digital lines. Note also that for a **V**-rule determined by an integer Q, the line $y = mx + n$ is mapped into a line with slope m/Q.

7. The **H**-rules defined imply choosing some origin point Ω_0 and decimating this time the vertical grid lines, by removing batches of P consecutive vertical lines. The basis vectors are in this case $B_1 = [P, 0]$ and $B_2 = [0, 1]$ (see Fig. 1.4f). In this case too $\mathbf{E}^2$ is properly contained in $\mathbb{Z}^2$ and again the mapping (1.6) has fractional entries, the determinant of $[B_1 B_2]$ being P. Clearly applying an **H**-rule to the chain-code of a digital straight line will yield the chain-code of a new line with slope mP, however this too is only an one-directional implication, not an "if and only if" result.

   We can clearly combine **V** and **H**-transformations to yield new and more complicated sequence mapping rules. For example, applying a **V** and an **H**-transformation with the same parameter, i.e., $P = Q$ is equivalent to re-encoding

the digitized straight line at a reduced resolution. Note that if the line passes through the origin, i.e., we have $y = mx$, and we apply a chain-code transformation rule that has the effect of reducing resolution with any $P = Q$, we must always obtain exactly the same chain-code since the new slope will be the same, $(m \cdot P)/Q = m$. This is a rather nice invariance property of chain-codes of lines passing through the origin and it is not entirely obvious in a nongeometric context. □

The fact that a digitized straight line has the above discussed series of invariance, or "self-similarity" properties, has many immediate consequences.

The result that an **R**-transformation on a sequence of symbols yields the chain-code of digitized straight line if and only if the original sequence was itself a straight line, constrains the run patterns of the symbol occurring in runs. We may have runs of equal-length runs but one of the run-length must always occur in isolation (otherwise the **R**-transformation would yield a sequence in which both symbols occur in runs longer than 1). Furthermore, this must also be the case at further levels of run-length encoding of the run-length sequences.

Consider the chain-code of a digitized straight line $C(m, n)$. Performing an **S**-transformation on it we get a new chain-code with the property that every symbol in the new sequence of symbols corresponds to, or "contains", exactly one □ symbol from the original chain-code. Therefore parsing the **S**-transformed code into subsequences of equal length is equivalent to performing an **H**-transformation on the original chain-code. This shows that in any two equal length subsequences of a straight line chain-code the number of Δ's (and consequently also □'s) may differ by at most 1. This property shows that self-similarity is in fact a description of uniformity in the distribution of the separator symbols (Δ) in the chain-code sequence. Indeed the slope of the line m sets the density of these symbols, and the digitization process ensures that this density will be achieved with a distortion as uniform as possible. This interpretation of digital straight lines, as well as their connections to Euclid's division algorithm (via continued fraction representations) and to a wealth of other areas as diverse as music [16], billiard trajectories [4], abstract sequence analysis [3], combinatorics on words [24], and quasicrystals [30, 32], make this area of research essentially inexhaustible.

From among many interesting consequences of the self-similarity results we have chosen to mention the above two properties because such results have been obtained before, using different proofs, in the context of testing whether a finite sequence of two symbols could be the chain-code of a digitized straight line segment, see, e.g., [20].

What we have in fact shown is that one can obtain chain-code transformation rules that characterize linearity, via the group $GL(2, \mathbb{Z})$ of unimodular lattice transformations. As a consequence we can readily "produce" countably many interesting and new self-similarity properties of linear chain-code patterns.

Using the properties of digital straight lines, we can not only solve the somewhat theoretical issue of locating a half-plane object of infinite extent but we can also address some very practical issues like measuring the perimeters of gen-

eral planar shapes from their versions digitized on regular grids of pixels. Indeed, analyzing the properties of digitized lines made possible the rational design of some very simple and accurate perimeter estimators, based on classifications of the boundary pixels into different classes according to the jaggedness of their neighborhoods. Building upon earlier work of Proffit and Rosen [31], Koplowitz and Bruckstein proposed a general methodology for the design of simple and accurate perimeter estimation algorithms that are based on minimizing the maximum error that would be incurred for infinitely long digitized straight edges over all orientations [21]. This methodology enables predictions of the expected performance for shapes having arbitrary, but bounded curvature boundaries.

1.4 Digital Straight Segments: Their Characterization and Recognition

The previous section focused on the properties of digitized half planes of infinite extent but we often discretize polygonal shapes that have finite length straight segments as boundaries. Such shapes which will yield finite sequences of chain-code symbols that will be called Digitally Straight Segments (DSS's). In general it is of interest to describe a general discretized boundary by partitioning it into a sequence of digital straight segments, effectively producing a "polygonal pre-image" of the boundary on which a variety of measurements can be performed. In order to describe a very efficient and easy to grasp algorithm for partitioning a chain-code sequence into discrete straight portions we need to formalize the Hough-domain, or dual-space "pre-image" concept. It is well known that a point in the plane defines a pencil of lines that pass through it, i.e., $(x_0, y_0) \in \mathbb{R}^2$ corresponds to the lines $y = mx + n$ that obey $y_0 = mx_0 + n$, and this is an equation defining a line in the Hough-space where the coordinates are (m, n). When a straight Black/White boundary is digitized by point sampling, the grid points that are on the border between the black region ($\xi_D(i, j) = 1$) and the white one ($\xi_D(k, l) = 0$) correspond to lines in the Hough-plane that delineate the (m, n) domain to which the straight line of the boundary belongs. Considering Fig. 1.5 we have that the lines corresponding to a vertical border of the discretized line, i.e., the pixels (i, j), $(i, j + 1)$ for which: $\{\xi_D(i, j) = 1, \xi_D(i, j + 1) = 0\}$ are the parallel lines in the (m, n)-plane defined by

$$\begin{cases} (i + j) = mi + n & \Rightarrow \quad n = -im + (j + 1) \\ j \qquad\quad = mi + n & \Rightarrow \quad n = -im + j \end{cases}$$

Similarly, the next vertical pair of border pixels for which $\{\xi_D(i + 1, j) = 1, \xi_D(i + 1, j + 1) = 0\}$ correspond to the (m, n)-plane lines given by the parallel lines:

$$\begin{cases} (j + 1) = m(i + 1) + n & \Rightarrow \quad n = -(i + 1)m + (j + 1) \\ j \qquad\quad = m(i + 1) + n & \Rightarrow \quad n = -(i + 1)m + j \end{cases}$$

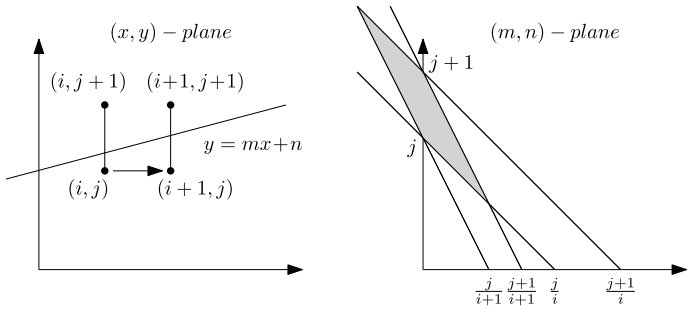


Fig. 1.5 Chain-code step and the corresponding Hough-plane geometry

Since clearly the pre-image border line intersects the segments $[(i, j), (i, j + 1)]$, and $[(i + 1, j), (i + 1, j + 1)]$ the (m, n) parameters of the pre-image line belong to the intersection of the bands defined by the two pairs of parallels corresponding to the border pixels. The "locale" for the pre-image has been therefore restricted to a parallelogram by the discrete data that corresponds to one step of the digitized boundary's chain-code. Since this process can be repeated for each chain-code symbol in the description of the discretized boundary, we have that each new chain-code symbol requires the intersection of the previously delineated "locale" for the "pre-image line" with a pair of parallel lines in the (m, n)-plane. We therefore have the following recursive algorithm for determining the "pre-image line locale", which is also, in fact, a process for determining digital straight segment portions of a chain-code:

Digital Straight Segment Detection Process

1. For each symbol of the 4-directional chain-code intersect the uncertainty region or locale in the (m, n)-plane with the corresponding band in the Hough(dual)-plane.
2. While the result is not empty there exists a linear-pre-image for the chain-coded portion of the boundary, hence the chain-code portion is a digital straight segment.

A careful analysis of how the intersections of chain-code bands look in the Hough plane reveals a miraculous fact: the locales are always regions defined by at most 4 boundary lines. This is a marvelous result due to Leo Dorst [12], which was given a simple proof by Douglas McIlroy in [25]. The result is indeed marvelous because it means that the recursive intersection process that the above described algorithm for detecting digital straight segment will only take $O(1)$-time, requiring the intersection of a four-sided polygon with two parallel lines. And the situation is even better: the points defining the locale polygon have rational coordinates hence the DSS detection process involves updating 8 integers for each chain-code symbol parsed, see [23]. Therefore we have a beautifully simple $O(1)$/(chain-code-step)

recursive digital straight segment detection process. Furthermore, starting the algorithm on an arbitrary chain-coded border we can very efficiently parse it into DSS-segments. Hence, given a shape digitized on $\mathbb{Z}^2$, we can determine a polygonal approximation for the shape by parsing the digitized boundary into DSS segments, and for each of these we have position and slope estimates readily provided by their Hough-plane [(m,n)-plane] "locales". In particular the recursive $O(1)$-per boundary pixel algorithm for detecting digital straightness described above, due to Lindenbaum and Bruckstein [23], enables parsing general, curved object boundaries into digitally straight segments in order to estimate the pre-image object's perimeter as a sum of the lengths of the line-segments so-detected. In terms of the methodology discussed in [21] this algorithm yields zero error for digital straight edges of infinite extent at all orientations, and hence should be the best perimeter estimator ever, if the criterion would be performance on straight boundaries.

1.5 Digital Disks, Convex and Star-Shaped Objects

From the realm of half-plane objects and digital straight lines we could move to either infinite extent regions that have more complex boundaries (say parabolas, hyperbolas or some periodic functions along a principal direction) or to the analysis of finite extent objects like polygons, disks and other interesting shapes. Some work has indeed been done on detecting polygonal preimages from their digitized versions, and, as we have seen, a good algorithm for parsing a jagged boundary into digital straight segments turns out to be a crucial ingredient in solving various issues regarding the metrology of such objects.

Suppose next that we have the prior information that the objects discretized are disks of various locations and sizes. Then the metrology question arising naturally is: how precisely can we determine the location of a disk and its radius. Considering the digitization by point sampling, as discussed above, given a digitized image of black and white pixels, we know that if a certain point in the plane is the center of a disk of unknown radius, this point will necessarily be closer to all black grid points than to any white grid point. Hence the locus of all possible points in the plane closer to all black points than to any white points is the locale of possible disk centers, and its size will quantify our uncertainty in locating the object in the preimage plane. It is interesting to note that this locale can be found without knowledge on the radius, which will still need to be estimated. It turns out that the locale as defined above is a well-known concept in computational geometry, and it is known that it is a convex region in the plane. Efrat and Gotsman have done a careful analysis of the problem and produced an $O(R \log R)$ algorithm to determine the locale, where R is the radius of the disk. We refer the interested reader to the paper [13] for details. Note again that the locale we are talking about is independent of the radius parameter. Had we prior knowledge on the exact radius, the location of the disk center could be determined by intersecting all disks of radius R around the black grid points with

all the complements of disks or radius R around the white (uncovered) grid points. The resulting intersection locale is generally not a convex shape, due to the precise knowledge of the radius.

For general convex shapes the question of determining the location, area and perimeter cannot be addressed in any generality. The digitized version of a convex shape is a set of black grid points on a background of white ones. As a union of square pixels the digitized shape will not be convex. Hence much work was done addressing the question whether there is a good definition of convexity for discrete objects [20, 37]. A variety of proposals were made and can be found in the literature. The metrology questions however, in all cases remain: determine with best precision the location (first order moments), orientation (second order moments) and other metric properties, like area (zeroth order moment) and perimeter of the shape. These questions, too have received some attention. It turns out that computing the moments of the black grid points yields good estimates for the corresponding continuous quantities, and more refined, boundary estimation procedures (say, based on polygonalization of the jagged boundary via an efficient digital straight segment detection, as discussed above) do indeed provide improved estimates but the improvement needs to be carefully weighed against the increased complexity involved.

Among the many procedures that propose polygonal approximations to preimages based on the discrete grid points that were covered by the shape, and also based on the ones that were not covered, one stands out in elegance and usefulness: the *minimum perimeter polygon* that is enclosing all black (covered) points and excludes all white (uncovered) ones. This minimum perimeter polygon turns out to be the relative convex hull of the connected graph of sampled black points with respect to the white ones. Here we assume that sampling is dense enough so that a connected preimage shape ensures that the black pixels form a 4-connected shape! The relative convex hull can be computed easily and may serve as a good approximation for preimages for all metrology purposes.

So far we talked about disks and convex objects. The next level of complexity in planar shapes are the so called star-shaped objects. These are defined as the shapes that have a "kernel region" inside them so that from any point in the kernel the entire boundary of the shape can be "seen", i.e., a line from the chosen point to any boundary point will lie entirely inside the shape. It is easy to see that this definition generalizes convexity in a rather natural way and that the kernels must be convex regions. Determining star-shapedness of a planar shape is not a too difficult task for polygons and for spline-gons, and the algorithms for doing this rely on locating and using the inflection points on the boundary, and intersecting the regions in the plane from where the convex boundary regions are seen, see [5]. As with the notion of convexity, determining digital star-shapedness posed a series of special problems that needed careful analysis. This was the topic of a paper by Shaked, Koplowitz and Bruckstein, and there it was shown that the relative convex hull, or minimal perimeter polygon of the grid points covered by the shape with respect to the ones that remained uncovered, provides a convenient computational way to define and algorithmically determine digital star-shapedness, see [33].

1.6 Shape Designs for Good Metrology

Up to this point we have discussed ways to analyze and measure planar shapes when seen through the looking glass of grid probing, or point-sampling discretization. The classes of shapes were assumed given in some, perhaps parameterized form, and we dealt with questions about recovering their various features and parameters, or about measuring their size and perimeter and determining their location with the highest precision possible.

When considering such issues, a further question that can be posed is the following: design planar shapes or collections of shapes that will interact with the discretization process in such a way that the quantities we need to measure will be very easily read out in the discretized images we get. Could we design an object in the plane (that can be a union of continuous binary shapes), so that digitization of this object translated to various locations, will yield black and white patterns on the (discretization) grid that clearly exhibit, say in a binary representation, the X and Y translation values up to a certain desired precision?

Interestingly, recently a pen-like device was invented and advertised, that has the following feature: it automatically computes with very high precision the location of its tip on any of the pages of a paper pad by looking at a faint pattern of dots that is printed on these sheets of paper. The pattern of these dots is so designed that the image obtained on any small region as seen by the pen near it's tip (with the help of a tiny light detector array) uniquely and easily locates the pen-tip's position on any of the pages of the pad, see [1].

This example shows that it is good practice to think about designing shapes to have such "self-advertising" properties and this approach could provide us surprisingly efficient and precise metrology devices. This problem was posed by Bruckstein, O'Gorman, and Orlitsky, at Bell Laboratories, already in 1989, with the aim of designing planar patterns that will serve as location marks, or fiducials on printed circuit boards. The need for location or registration fiducials in printing circuit boards and in processing VLSI devices is quite obvious. When layers of printing and processing are needed in the manufacturing operation, the precision in performing the desired processes in perfect registration with previously processed layers is indeed imperative. The work described in [7] proves that there exists an *information theoretic bound* that limits the location precision for any shape that has an spatial extent of say $A \times A$ in pixel-size. Such a shape, when digitized will provide for us about A^2 meaningful bits of information, via the pattern of black and white pixels in the digitized image. This number of bits can only effectively encode 2^{A^2-1} different locales, and hence the precision to which we can refine a region one pixel-square in size has a maximal area that must exceed $1/(2^{A^2-1})$. If we want balanced X and Y axis precision, we can only locate the pattern to a subpixel precision of $1/[2^{(A^2-1)/2}]$. This is the best precision possible assuming optimal exploitation of the real estate of an $A \times A$ area, assigned to the location mark. The important issue that was further settled in [7] is the existence of a fiducial pattern that indeed achieves this precision. The pattern is so cute that we exhibit it in Fig. 1.6.

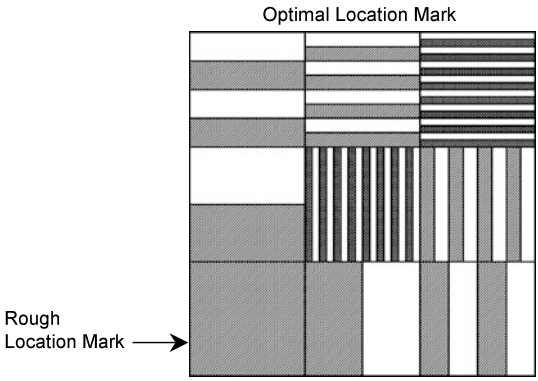
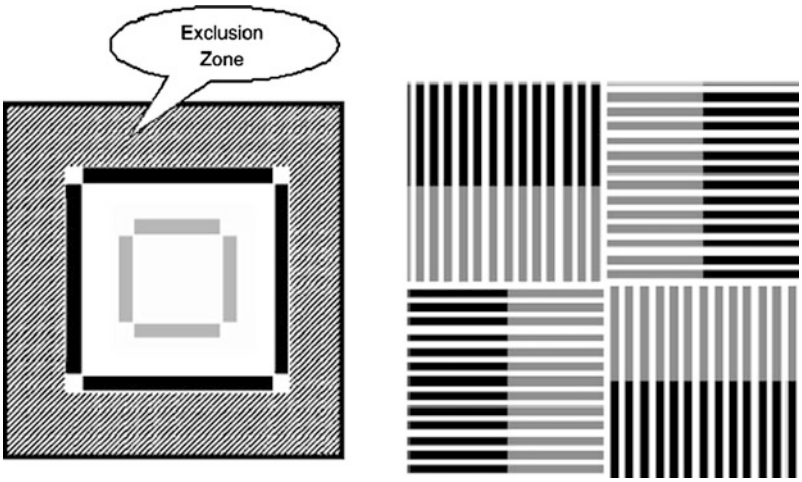


Fig. 1.6 An optimal 2D fiducial of area 3×3 pixels

Looking at this fiducial pattern it becomes obvious what it does. It is indeed a continuous 2D (analog) input that employs the point sampling discretization process to compute its X and Y displacement by providing a binary read-out of the subpixel location of the fiducial within the one pixel to which it can readily be located using the left lowest grid-point (the "rough location" mark) covered by the shape. This leftmost bit of information is also the reason we can only use $A^2 - 1$ bits for subpixel precision, i.e., for cutting the one pixel precision (provided by the "rough location" bit) into locale slices. This process turns the fiducial and the discretization process into a nice "analog computer" that yields the displacements in the X and Y direction easily, and achieves the highest precision in this task that is possible based on the available data. The analysis provided in [7] goes even further. The optimal fiducials turn out to require highly precise etchings on the VLSI or circuit board devices and hence might be difficult to realize in practice. Hence there is a need to analyze other types of fiducial shapes that achieve suboptimal exploitation of the area, however can provide good location accuracies. For rotational invariance, circularly symmetric shapes turn out to be necessary, and therefore bull-eye fiducials were also proposed in [7] and further analyzed in [13, 34]. The main message of the theoretical analysis provided in [7] was that a self location fiducial should have lots of edges that carry information on their location when seen through a digitization camera. Recently, the semiconductor industry used this insight in redesigning the standard registration fiducials. This was the result of a detailed study of novel, robust grating mark fiducials, which greatly increased precision and repeatability. The study, done by us in conjunction with a team of design engineers at KLA-Tencor, a leading manufacturer of vision based process inspection machines for semiconductor industry, proposed fiducial marks as shown in Fig. 1.7b to replace the traditional box in a box mark shown in Fig. 1.7a. The traditional fiducial was clearly not optimal in terms of exploiting the wafer area allocated to it. For a detailed description of the optimized overlay metrology marks that were adopted by industry and the theoretical analysis that led to their design, see [2].

The most interesting question that remains to be addressed here is the following: can we invent shapes that provide other metrological measures as easily as the above discussed example advertised its location?



(a) Box in Box fiducial. (b) The new standard fiducial.

Fig. 1.7 Overlay metrology fiducials (from [2])

1.7 The Importance of Being Gray

So far we have discussed the case of binary continuous images being point-sampled into matrices of zeros and ones, or Black and White pixels. However the real world is far richer in possibilities and complications.

First of all, point sampling is not a good model of the imaging process as performed by real life cameras. Those carry out, at each sensor level, a weighted integration of the incoming light from the continuous input pattern. This integration happens around each grid point, and the pixel influence region may be assumed circular. The integration yields, at each grid point, values that continuously vary from a lowest value for white (no object) input over the pixel influence region to highest value that corresponds to having the input object cover the entire area of integration. The result of this integration is then transformed into a discrete value encoded by several bits, via quantization. Therefore even for binary preimages, we get at each grid point a pixel value that is the quantization of a continuous variable proportional to the fraction of the pixel influence region that is covered by the input object.

Furthermore we may also consider the advantages of using non-binary, gray-scale of color pre-images. The combination or more realistic sampling and quantization processes with the use of gray levels in preimages open for us a great variety of further possibilities. As an example, Kiryati and Bruckstein have analyzed, following a question posed by Theo Pavlidis, the trade-off between spatial resolution and number of gray levels when the aim is to get as much information as possible on a class of binary pre-images that comprise polygonal shapes. The conclusion of this research was that "Gray Levels Can Improve the Performance of Binary Image Digitizers", see [19]. The paper introduces a measure of digitization-induced ambiguity in recovering the binary preimage, hence it is quite relevant to metrology under such

sampling conditions. It is then proved that, if the sampling grid is sufficiently dense (i.e., the sampling rate is high!) and if the pixels would provide us exact gray-levels rather than quantized values, error-free reconstruction of the binary pre-image is possible. This is not too surprising, however, when the total bit budget for the digitized image representation is limited (i.e., the sampling rate and the quantization depth are related, both being finite) the bit allocation problem that arises shows that the best resource allocation policy is to increase the gray level quantization accuracy as much as possible, once a sufficiently dense spatial sampling resolution has been reached. Therefore once we have a grid dense enough to ensure that all linear borders of the binary input image polygonal shapes can adequately be "seen" in the sampled image, all the remaining bit resources should go towards finer gray level quantization. The question, which prompted this research asked to explain why gray-level fax machines at low resolution yield nicer images than fax machines at higher resolution, even for binary document images. It was clear that some sort of anti-aliasing effect is in place, however [19] proved quantitatively that even in terms of a well-defined metrology error measure, the gray-levels help considerably more than increased spatial resolution.

Imagine next that we allow gray level input images too. In this case we shall certainly have, in conjunction with multilevel quantizations at each pixel much more information for location and various other measurements. A gradual boundary in the input image, or equivalently an area integration sensor providing a quantized multilevel pixel value at each grid-point, will transform the issue of locating a half plane into a problem of locating precisely several parallel digital straight edges, when they are simultaneously sampled. Such richness of detail will certainly dramatically reduce the size of the uncertainty locales, and enable us to design a wealth of improved location and orientation fiducials in the future.

The conclusion therefore is that gray levels matter, they are good for us! And the last word on these issues certainly has not been said yet. For some very nice recent work along these lines see [35].

1.8 Some Further Open Questions

As was mentioned in the previous sections, there are still many interesting and open digital geometry and metrology problems. Although digital straight edges did receive a lot of attention from digital geometry researchers we still expect to see complete theories pertaining to the sampling and quantization of linear non-step borders in gray level preimages. If a straight border with sigmoidal gray level profile is sampled by some type of area sampling (with pixels with circularly symmetric integration regions) the result will be a border-line with quantized gray levels that will look like a nicely anti-aliased line produced by a computer graphics algorithm. There are interesting digital line properties of the type we discussed in Sect. 1.3 embedded in the resulting image and these will surely be carefully studied sometime in the future.

Along these lines one could also study a class of location fiducials based on shapes with multiple parallel edges, or edges with an a priori known pattern. Such robust fiducials should enable "the design" of desired uncertainly locales for high precision registration, may even be insensitive to pixel size variations.

Another interesting question on self-location that may be subject to further research is the design of binary self-location patterns in the plane. This problem was partially addressed in the paper [15], the pattern proposed being a separable bit-pattern that is generated as the outer (binary) product of two one-dimensional de Bruijn sequences [28] that have the one-dimensional self-location property. Such a pattern can be shown to be robust to some read-out errors but clearly it has a bit too much redundancy built into it. The planar pattern used by the Anoto pen we mentioned before, [1], is an "analog" point pattern that is based on encoding location in geometric constellations of points near grid locations that carry the information on the absolute coordinates of the grid point. It seems that a binary array version of the problem has not been discussed before the work reported in [15].

The problem of length estimation of discretized boundaries was the subject of many papers, as seen in [21, 35] and the references therein. However even this topic was not yet completely exhausted. It is an interesting challenge to design perimeter estimators that will work in conjunction with corner detectors and curvature estimators, perhaps based on digital circle detectors [10], to yield more and more precise length measurements. The design here should not be aimed to get precise results on digital straight lines but rather on various types of continuous curves with breakpoints and corners, and the ranges of curvatures that are expected to appear in practice.

As we discussed in the previous section, subject of bit allocation tradeoff's between resolution and quantization has only been superficially touched upon so far [17, 19, 35]. Although the initial conclusions are that multilevel quantization provides quite a lot of information in binary preimage digitization, a similar study should be made for the case of gray level shape boundaries and gray level images of various types. In this context one might even ask what should be the design of the gray-scale profile of planar shape edges to enhance the edge location and length estimation performance.

We have not discussed in this paper important questions of shape comparison and recognition, of shape decompositions and isoperimetric inequities for digitized shapes. These topics all rise very interesting research questions that are recently beginning to be addressed, see, e.g., [8, 36]. Therefore we may expect the area of digital geometry to remain an active and exciting subject of research in the future.

1.9 Concluding Remarks

This paper surveys research that dealt with digital geometry and metrology issues. As is clear form the topics discussed above and the list of references below, metrology tasks require deep and interesting excursions into discrete geometry, motivating

the study of the pixelized world and importing from it important insights and results. More on the vast subject of discrete geometry can be found in several books [9, 11, 20, 22, 26, 27, 29].

Acknowledgement Many thanks to Ms. Yana Katz for preparing this paper for publication.

References

1. http://www.anoto.com/the_paper_3.aspx
2. Adel, M., Ghinovker, M., Golovanevsky, B., Izikson, P., Kassel, E., Yaffe, D., Bruckstein, A.M., Goldenberg, R., Rubner, Y., Rudzsky, M.: Optimized overlay metrology marks: theory and experiment. IEEE Trans. Semicond. Manuf. **17**(2), 166–179 (2004)
3. Allouche, J.P., Shallit, J.: Automatic Sequences. Cambridge University Press, Cambridge (2003)
4. Baryshnikov, Y.: Complexity of trajectories in rectangular billiards. Commun. Math. Phys. **174**(1), 43–56 (1995)
5. Bornstein, R., Bruckstein, A.M.: Finding the kernel of planar shapes. Pattern Recognit. **24**(11), 1019–1035 (1991)
6. Bruckstein, A.M.: Self-similarity properties of digitized straight lines. Contemp. Math. **119**, 1–20 (1991)
7. Bruckstein, A.M., O'Gorman, L., Orlitsky, A.: Design of shapes for precise image registration. IEEE Trans. Inf. Theory **44**(7), 3156–3162 (1998). AT&T Bell Laboratories Technical Memorandum, 1989
8. Bruckstein, A.M., Shaked, D.: Crazy-cuts: dissecting planar shapes into two identical parts. In: IMA Mathematics of Surfaces XIII Conference, The University of York, UK, September 7–9, 2009
9. Chassery, J.M., Montanvert, A.: Géométrie Discréte en Analyse d'Images. Hermes, Paris (1991)
10. Coeurjolly, D., Gérard, Y., Reveillès, J.P., Tougne, L.: An elementary algorithm for digital arc segmentation. Discrete Appl. Math. **139**, 31–50 (2004)
11. Davis, L.S. (ed.): Foundations on Image Understanding. Kluwer Academic, Dordrecht (2011). (The Azriel Rosenfeld Book)
12. Dorst, L.: Discrete straight line segments: parameters, primitives and properties. Ph.D. thesis, Technological University Delft (1986)
13. Efrat, A., Gotsman, C.: Subpixel image registration using circular fiducials. Int. J. Comput. Geom. Appl. **4**, 403–422 (1994)
14. Freeman, H.: On the encoding of arbitrary geometric configurations. IRE Trans. Electron. Comput. **EC-10**, 260–268 (1961)
15. Giryes, R., Shuldiner, D., Gordon, N., Holt, R.J., Bruckstein, A.M.: Simple and robust binary self-location patterns. IEEE Trans. Inf. Theory (2012). doi:10.1109/TIT.2012.2191699
16. Gómez-Martín, F., Taslakian, P., Toussaint, G.: Structural properties of euclidean rhythms. J. Math. Music **3**(1), 1–14 (2009)
17. Gustavson, S., Strand, R.: Anti-aliased euclidean distance transform. Pattern Recognit. Lett. **32**, 252–257 (2011)
18. Havelock, D.I.: The topology of locales and its effects on position uncertainty. IEEE Trans. Pattern Anal. Mach. Intell. **13**(4), 380–386 (1991)
19. Kiryati, N., Bruckstein, A.M.: Gray levels can improve the performance of binary image digitizers. CVGIP, Graph. Models Image Process. **53**(1), 31–39 (1991)
20. Klette, R., Rosenfeld, A.: Digital Geometry—Geometric Methods for Digital Picture Analysis. Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann, San Francisco (2004)

21. Koplowitz, J., Bruckstein, A.M.: Design of perimeter estimators for digitized planar shapes. IEEE Trans. Pattern Anal. Mach. Intell. **PAMI-11**(6), 611–622 (1989)
22. Latecki, L.J.: Discrete Representation of Spatial Objects in Computer Vision. Springer, Berlin (1998)
23. Lindenbaum, M., Bruckstein, A.M.: On recursive, $O(N)$ partitioning of a digitized curve into digital straight segments. IEEE Trans. Pattern Anal. Mach. Intell. **15**(9), 949–953 (1993)
24. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge University Press, Cambridge (2002)
25. McIlroy, M.: A note on discrete representation of lines. AT&T Bell Labs Tech. J. **64**(2), 481–490 (1984)
26. McIlroy, M.D.: Number theory in computer graphics. Proc. Symp. Appl. Math. **46**, 105–121 (1992)
27. Melter, R.A., Rosenfeld, A., Bhattacharya, P. (eds.): Vision Geometry. Contemporary Mathematics, vol. 119. AMS, Providence (1991)
28. Mitchell, C., Etzion, T., Paterson, K.G.: A method for constructing decodable de Bruijn sequences. IEEE Trans. Inf. Theory **42**(5), 1472–1478 (1996)
29. Pavlidis, T.: Algorithms for Graphics and Image Processing. Comput. Sci. Press, Rockville (1982)
30. Pleasants, P.A.B.: Quasicrystallography: some interesting new patterns. In: Elementary and Analytic Theory of Numbers, 2nd edn. Banach Center Publications, vol. 17, pp. 439–461. PWN, Warsaw (1985)
31. Proffit, D., Rosen, D.: Metrication errors and coding efficiency of chain coding schemes for the representation of lines and edges. Comput. Graph. Image Process. **10**, 318–332 (1979)
32. Senechal, M., Taylor, J.: Quasicrystals: the view from Les Houches. Math. Intell. **12**(2), 54–64 (1990)
33. Shaked, D., Koplowitz, J., Bruckstein, A.M.: Star-shapedness of digitized planar shapes. Contemp. Math. **119**, 137–158 (1991)
34. Shih, S.W., Yu, T.Y.: On designing an isotropic fiducial mark. IEEE Trans. Image Process. **12**(9), 1054–1066 (2003)
35. Sladoje, N., Lindblad, J.: High-precision boundary length estimation by utilizing gray-level information. IEEE Trans. Pattern Anal. Mach. Intell. **31**(2), 357–363 (2009)
36. Vainsencher, D., Bruckstein, A.: On isoperimetrically optimal polyforms. Theor. Comput. Sci. **406**, 146–159 (2008)
37. Voss, K.: Discrete Images, Objects, and Functions in Z^n. Springer, Berlin (1991)

Chapter 2
Provably Robust Simplification of Component Trees of Multidimensional Images

Gabor T. Herman, T. Yung Kong, and Lucas M. Oliveira

Abstract We are interested in translating n-dimensional arrays of real numbers (*images*) into simpler structures that nevertheless capture the topological/geometrical essence of the objects in the images. In the case $n = 3$ these structures may be used as descriptors of images in macromolecular databases. A *foreground component tree structure* (*FCTS*) contains all the information on the relationships between connected components when the image is thresholded at various levels. But unsimplified FCTSs are too sensitive to errors in the image to be good descriptors. This chapter presents a method of simplifying FCTSs which can be proved to be robust in the sense of producing essentially the same simplifications in the presence of small perturbations. We demonstrate the potential applicability of our methodology to macromolecular databases by showing that the simplified FCTSs can be used to distinguish between two slightly different versions of an adenovirus.

2.1 Introduction

High-level structural information about macromolecules is now being organized into databases. These include EM maps (three-dimensional grayscale image arrays obtained by reconstruction from electron microscopic data) of macromolecular structures. The large size of these image arrays, the arbitrary position and orientation of the macromolecule in the array, and the possibility of non-linear stretching of the

G.T. Herman (✉) · L.M. Oliveira
Computer Science Ph.D. Program, Graduate Center, City University of New York,
365 Fifth Avenue, New York, NY 10016, USA
e-mail: gaborherman@yahoo.com

L.M. Oliveira
e-mail: lmoliveira@gmail.com

T.Y. Kong
Computer Science Department, Queens College, City University of New York,
65-30 Kissena Boulevard, Flushing, NY 11367, USA
e-mail: ykong@cs.qc.cuny.edu

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_2,

range make standard methods of comparison between database entries infeasible. There is a need for simple robust descriptors that capture the topological/geometrical essence of the macromolecules in the images. We believe that appropriately simplified foreground component tree structures may be suitable for this purpose.

Foreground component trees are well known representations of grayscale images. Given a grayscale image $I : \mathcal{S} \to \mathbb{R}$ whose domain $\mathcal{S}$ is connected, the foreground component tree of I is a rooted tree whose nodes are the connected components of superlevel sets of I. These nodes have sometimes been called *maximum intensity extremal regions* [6]. A node $\mathbf{c}'$ is an ancestor in the tree of a node $\mathbf{c}$ if and only if $\mathbf{c}' \supseteq \mathbf{c}$. The tree can be efficiently constructed using an algorithm which processes the elements of $\mathcal{S}$ in decreasing order of their graylevels and uses Tarjan's union-find algorithm [11] to build the tree from the bottom up. For details, see [1, Alg. 4.1] or [7, Alg. 2]. The latter paper also describes applications of foreground component trees to image processing and gives a bibliography of some relevant literature.

Two related representations of images (contour trees and 0th persistence diagrams) will be described in Sect. 2.7 when we discuss research problems suggested by our work.

Unsimplified foreground component trees are too sensitive to errors in the image to be good descriptors. Accordingly, this chapter presents a new three-step method of simplifying these trees that is provably robust, in the sense that the method produces essentially the same simplified trees when the image is slightly perturbed. This property of our method is precisely stated in our main result, Theorem 1.

Methods of simplifying component trees to suppress features that are likely due to noise or artifacts have previously been considered (see, e.g., [7, 10]). But we are not aware of any previous work in which a tree simplification method has been proved to have a robustness property of the kind stated in Theorem 1.

We believe that the simplified trees produced by our method will be useful image descriptors for the identification and classification of macromolecules. As evidence of this we provide a sample biological application in which they are used to differentiate two versions of an adenovirus.

2.2 Foreground Component Tree Structures (FCTSs)

We use the term *adjacency relation* to mean an irreflexive symmetric binary relation (i.e., a set κ of ordered pairs such that if $(a, b) \in \kappa$ then $a \neq b$ and $(b, a) \in \kappa$). The members of the pairs that belong to any adjacency relation we are using will be called *spels*. (As in, e.g., [5], "spel" is an abbreviation of "spatial element", and we think of spels as generalizations of pixels and voxels.) We use the term *grayscale image* or, more briefly, the term *image*, to mean a real-valued function whose domain is a nonempty set of spels. If $I : \mathcal{S} \to \mathbb{R}$ is any image then for any $s \in \mathcal{S}$ we may refer to the real value $I(s)$ as the *graylevel* of s in I.

In the practical work described in Sect. 2.6, we use the "6-adjacency" relation [5, p. 16] on $\mathbb{Z}^3$ as our adjacency relation, and use grayscale images whose domain is the finite set $\{(x, y, z) \in \mathbb{Z}^3 \mid 0 \leq x \leq 274,\ 0 \leq y \leq 274,\ 0 \leq z \leq 274\}$.

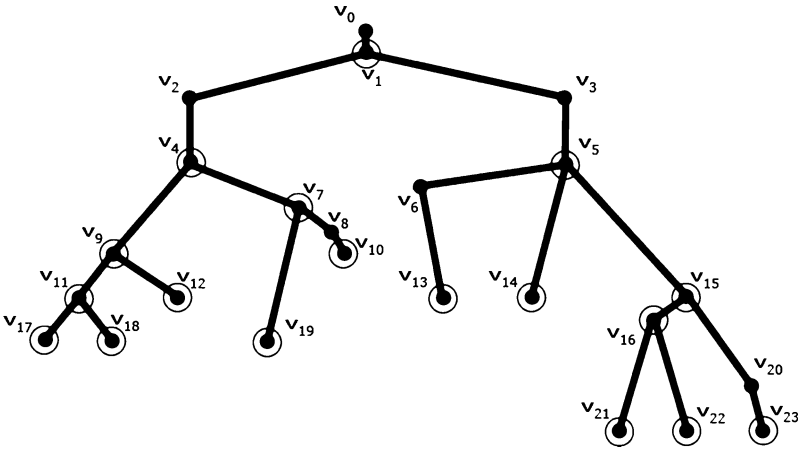


Fig. 2.1 A rooted tree in which the critical nodes have been circled

Let κ be an adjacency relation. We say that two disjoint sets of spels $\mathcal{S}_1$ and $\mathcal{S}_2$ are κ-*adjacent* if there exist $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$ such that $(s_1, s_2) \in \kappa$. We call a sequence $s_0, \ldots, s_l$ of $l+1$ spels a κ-*path* if $l = 0$ or if $l \geq 1$ and $(s_i, s_{i+1}) \in \kappa$ for $0 \leq i < l$. We say that a set $\mathcal{S}$ is κ-*connected* if for all $s, s' \in \mathcal{S}$ there exists a κ-path $s_0, \ldots, s_l$ such that $s_0 = s$, $s_l = s'$, and $s_i \in \mathcal{S}$ for $0 \leq i \leq l$.

Let $I : \mathcal{S} \to \mathbb{R}$ be any image, let $\tau \in \mathbb{R}$, and let $s \in \mathcal{S}$. Then $\mathcal{C}_\kappa(s, I, \tau)$ will denote the set of all $s' \in \mathcal{S}$ for which there exists a κ-path $s_0, \ldots, s_l$ such that $s_0 = s$, $s_l = s'$, and $I(s_i) \geq \tau$ for $0 \leq i \leq l$. Note that $\mathcal{C}_\kappa(s, I, \tau) = \emptyset$ if $\tau > I(s)$, and $s \in \mathcal{C}_\kappa(s, I, \tau)$ if $\tau \leq I(s)$. We write $\mathcal{C}_\kappa(s, I)$ to denote the set $\mathcal{C}_\kappa(s, I, I(s))$. Readily, if $t \in \mathcal{C}_\kappa(s, I)$, then $I(t) \geq I(s)$ and either $\mathcal{C}_\kappa(t, I) = \mathcal{C}_\kappa(s, I)$ or $\mathcal{C}_\kappa(t, I) \subsetneq \mathcal{C}_\kappa(s, I)$ according to whether $I(t) = I(s)$ or $I(t) > I(s)$.

We assume the reader is familiar with the concept of a rooted tree (as defined in, e.g., [3, Appendix B.5.2]). Let $\mathscr{T}$ be any rooted tree. We write $\mathbf{Nodes}(\mathscr{T})$ to denote the (finite) set of all nodes of $\mathscr{T}$, write $\mathbf{root}(\mathscr{T})$ to denote the root of $\mathscr{T}$, and write $\mathbf{Leaves}(\mathscr{T})$ to denote the set of all leaves of $\mathscr{T}$.

Recall that if $\mathbf{u} \in \mathbf{Nodes}(\mathscr{T})$ and $\mathbf{v}$ is a node of the subtree of $\mathscr{T}$ that is rooted at $\mathbf{u}$, then $\mathbf{u}$ is said to be an *ancestor* of $\mathbf{v}$ in $\mathscr{T}$, and $\mathbf{v}$ a *descendant* of $\mathbf{u}$ in $\mathscr{T}$. We write $\mathbf{u} \preceq_{\mathscr{T}} \mathbf{v}$ or $\mathbf{v} \succeq_{\mathscr{T}} \mathbf{u}$ to mean that $\mathbf{u}, \mathbf{v} \in \mathbf{Nodes}(\mathscr{T})$ and $\mathbf{u}$ is an ancestor of $\mathbf{v}$ in $\mathscr{T}$. We write $\mathbf{u} \prec_{\mathscr{T}} \mathbf{v}$ or $\mathbf{v} \succ_{\mathscr{T}} \mathbf{u}$ to mean that $\mathbf{u} \preceq_{\mathscr{T}} \mathbf{v}$ but $\mathbf{u} \neq \mathbf{v}$. If $\mathbf{u} \prec_{\mathscr{T}} \mathbf{v}$ then $\mathbf{u}$ is said to be a *proper* ancestor of $\mathbf{v}$ in $\mathscr{T}$, and $\mathbf{v}$ a *proper* descendant of $\mathbf{u}$ in $\mathscr{T}$.

For $\mathbf{v} \in \mathbf{Nodes}(\mathscr{T})$, we write $\mathbf{Children}_{\mathscr{T}}(\mathbf{v})$ to denote the set of all the children of $\mathbf{v}$ in $\mathscr{T}$, and if $\mathbf{v} \neq \mathbf{root}(\mathscr{T})$ then we write $\mathbf{parent}_{\mathscr{T}}(\mathbf{v})$ to denote the parent of $\mathbf{v}$ in $\mathscr{T}$. A node $\mathbf{v}$ of $\mathscr{T}$ is said to be *critical* if $|\mathbf{Children}_{\mathscr{T}}(\mathbf{v})| \neq 1$; thus $\mathbf{v}$ is a critical node if and only if either $\mathbf{v} \in \mathbf{Leaves}(\mathscr{T})$ or $|\mathbf{Children}_{\mathscr{T}}(\mathbf{v})| \geq 2$. In Fig. 2.1, the critical nodes are circled.

Let κ be any adjacency relation. Then a κ-*foreground component tree structure* or κ-*FCTS* is a pair $(\mathscr{T}, \ell)$ for which there exists a collection $\mathcal{C}$ of nonempty finite κ-connected sets of spels such that the following four conditions hold:

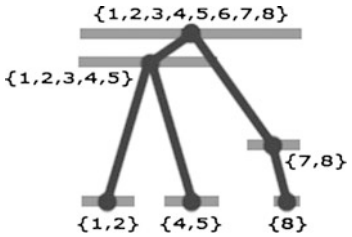


Fig. 2.2 The tree of the FCTS that is defined in Example 1

1. $\bigcup \mathcal{C} \in \mathcal{C}$
2. For all $\mathbf{u}, \mathbf{v} \in \mathcal{C}$, if $\mathbf{u} \not\supseteq \mathbf{v}$ and $\mathbf{v} \not\supseteq \mathbf{u}$ then the sets $\mathbf{u}$ and $\mathbf{v}$ are disjoint and are not κ-adjacent.
3. ℓ is a real-valued function on $\mathcal{C}$ such that, for all $\mathbf{u}, \mathbf{v} \in \mathcal{C}$, $\ell(\mathbf{u}) < \ell(\mathbf{v})$ whenever $\mathbf{u} \supsetneq \mathbf{v}$. (For each $\mathbf{v} \in \mathcal{C}$ we call $\ell(\mathbf{v})$ the *level* of $\mathbf{v}$.)
4. $\mathscr{T}$ is the rooted tree such that $\mathbf{Nodes}(\mathscr{T}) = \mathcal{C}$ and, for all $\mathbf{u}, \mathbf{v} \in \mathcal{C}$, $\mathbf{u} \prec_{\mathscr{T}} \mathbf{v}$ if and only if $\mathbf{u} \supsetneq \mathbf{v}$.

Condition 1 is equivalent to the condition that $\mathcal{C}$ have an element which is a superset of every element of $\mathcal{C}$. Moreover, since every element of $\mathcal{C}$ is required to be a nonempty finite κ-connected set, condition 1 implies that $\bigcup \mathcal{C}$ is a finite κ-connected set. Since $\bigcup \mathcal{C}$ is finite, $\mathcal{C}$ can only be a finite collection.

If $\mathcal{C}$ is *any* collection of nonempty finite κ-connected sets that satisfies conditions 1 and 2, and ℓ any function that satisfies condition 3, then there will exist a unique rooted tree $\mathscr{T}$ that satisfies condition 4 (so that $(\mathscr{T}, \ell)$ is a κ-FCTS); the root of this tree will be $\bigcup \mathcal{C}$.

Example 1 Let κ be the adjacency relation on the integers such that $(n_1, n_2) \in \kappa$ if and only if $|n_1 - n_2| = 1$. Let $\mathcal{C}$ be the following collection of six sets: $\{\{1, 2, 3, 4, 5, 6, 7, 8\}, \{1, 2, 3, 4, 5\}, \{1, 2\}, \{4, 5\}, \{7, 8\}, \{8\}\}$. Then it is readily confirmed that $\mathcal{C}$ satisfies conditions 1 and 2. Now let $\ell : \mathcal{C} \to \mathbb{R}$ be defined by $\ell(\{1, 2, 3, 4, 5, 6, 7, 8\}) = 12$, $\ell(\{1, 2, 3, 4, 5\}) = 13$, $\ell(\{7, 8\}) = 16$, and $\ell(\{1, 2\}) = \ell(\{4, 5\}) = \ell(\{8\}) = 18$. Then it is readily confirmed that ℓ satisfies condition 3. Thus there is a κ-FCTS $(\mathscr{T}, \ell)$ for which $\mathbf{Nodes}(\mathscr{T}) = \mathcal{C}$. The tree $\mathscr{T}$ of this κ-FCTS is shown in Fig. 2.2.

If $\mathfrak{F}$ is a κ-FCTS $(\mathscr{T}, \ell)$, then we may use $\mathfrak{F}$ to mean the rooted tree $\mathscr{T}$ in our terminology and notation. As examples of this, nodes and edges of $\mathscr{T}$ may be referred to as nodes and edges of $\mathfrak{F}$, the notations $\mathbf{Nodes}(\mathfrak{F})$, $\mathbf{root}(\mathfrak{F})$, and $\mathbf{Leaves}(\mathfrak{F})$ will have the same meanings as $\mathbf{Nodes}(\mathscr{T})$, $\mathbf{root}(\mathscr{T})$, and $\mathbf{Leaves}(\mathscr{T})$, and $\mathbf{parent}_{\mathfrak{F}}(\mathbf{v})$ will have the same meaning as $\mathbf{parent}_{\mathscr{T}}(\mathbf{v})$ for any $\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}) \setminus \mathbf{root}(\mathscr{T})$.

Let $\mathcal{S}$ be any nonempty finite κ-connected set of spels. Then we associate each image $I : \mathcal{S} \to \mathbb{R}$ with the κ-foreground component tree structure $\mathbf{FCTS}_{\kappa}(I)$ that is defined by $\mathbf{FCTS}_{\kappa}(I) = (\mathscr{T}_I, \ell_I)$, where:

(i) $\mathbf{Nodes}(\mathscr{T}_I) = \{\mathcal{C}_{\kappa}(s, I) \mid s \in \mathcal{S}\}$ and, for all $\mathbf{u}, \mathbf{v} \in \mathbf{Nodes}(\mathscr{T}_I)$, we have that $\mathbf{u} \preceq_{\mathscr{T}_I} \mathbf{v}$ if and only if $\mathbf{u} \supseteq \mathbf{v}$.

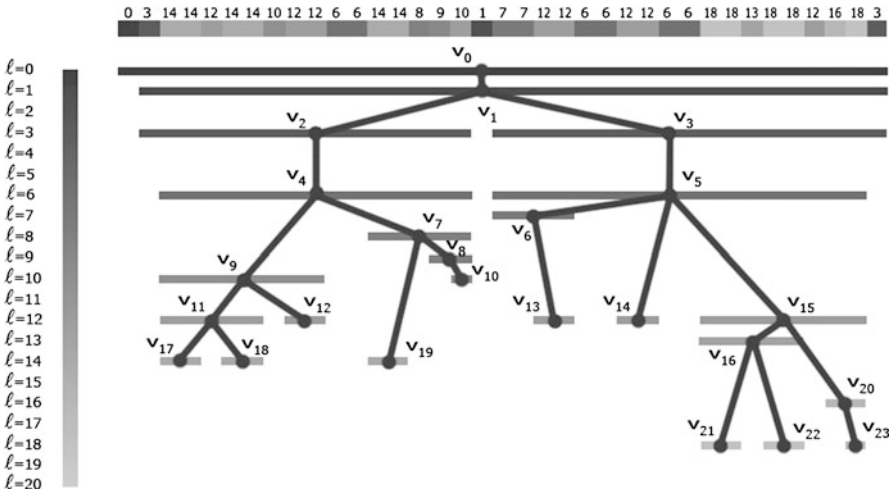


Fig. 2.3 A grayscale image whose domain is a row of 37 pixels is shown at the *top*. Writing I to denote this image, the *numbers* above the image show the graylevel $I(p)$ of each pixel p in the domain; for example, the graylevels of the first, second, third, and fourth pixels on the left are respectively 0, 3, 14, and 14. The κ-FCTS of the image (i.e., $\mathbf{FCTS}_\kappa(I)$) is shown below the image. Here κ is the adjacency relation such that $(p_1, p_2) \in \kappa$ just if p_1 and p_2 are pixels that share an edge. Writing $(\mathscr{T}, \ell)$ for this κ-FCTS, each node of the tree $\mathscr{T}$ is a κ-connected set of pixels whose elements are indicated in the figure by the *horizontal bar* which runs through that node. For example, the root node v_0 of $\mathscr{T}$ consists of all 37 pixels in the domain, the node v_1 consists of all pixels in the domain except the leftmost, and the leaf node v_{17} consists of just the third and the fourth pixels from the left. For each node v, the value of $\ell(\mathsf{v})$ can be read from the *vertical bar* on the left. For example, $\ell(\mathsf{v}_2) = \ell(\mathsf{v}_3) = 3$ and $\ell(\mathsf{v}_4) = \ell(\mathsf{v}_5) = 6$

(ii) For all $s \in \mathcal{S}$, we have that $\ell_I(\mathcal{C}_\kappa(s, I)) = I(s)$. ($\ell_I$ is well defined by this condition, because $I(s) = I(s')$ whenever $\mathcal{C}_\kappa(s, I) = \mathcal{C}_\kappa(s', I)$.)

It is readily confirmed that a κ-FCTS with these two properties exists, because $\mathcal{C} = \{\mathcal{C}_\kappa(s, I) \mid s \in \mathcal{S}\}$ satisfies conditions 1 and 2 in the definition of a κ-FCTS; the root of the tree of this FCTS is $\bigcup \mathcal{C} = \mathcal{S}$. It follows from (ii) that for each $\mathbf{v} \in \mathbf{Leaves}(\mathscr{T}_I)$ the level of $\mathbf{v}$ in $\mathbf{FCTS}_\kappa(I)$ is just the graylevel in I of each spel in $\mathbf{v}$, and that for each $\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}_I)$ the level of $\mathbf{v}$ is just the minimum of the graylevels of the spels in $\mathbf{v}$. We call $\mathbf{FCTS}_\kappa(I)$ the *κ-FCTS of* the image I. Figure 2.3 illustrates this concept.

Conversely, we associate each κ-FCTS $\mathfrak{F} = (\mathscr{T}, \ell)$ with the image $I_{\mathfrak{F}}$ that we now define. For each spel $s \in \mathbf{root}(\mathscr{T})$, conditions 2 and 4 in the definition of a κ-FCTS imply that, among the elements of $\mathbf{Nodes}(\mathscr{T})$ that contain s, there must be a smallest (i.e., a node that is a descendant in $\mathscr{T}$ of every node that contains s); that element will be denoted by $\mathbf{node}_{\mathscr{T}}(s)$. We define $I_{\mathfrak{F}} = I_{(\mathscr{T},\ell)}$ to be the image whose domain is $\mathbf{root}(\mathscr{T})$, and which satisfies $I_{\mathfrak{F}}(s) = \ell(\mathbf{node}_{\mathscr{T}}(s))$ for all $s \in \mathbf{root}(\mathscr{T})$. We also call $I_{\mathfrak{F}}$ the *image of* the κ-FCTS $\mathfrak{F}$.

Readily, $I_{\mathbf{FCTS}_\kappa(I)} = I$ for any image I whose domain is finite and κ-connected, and $\mathbf{FCTS}_\kappa(I_{\mathfrak{F}}) = \mathfrak{F}$ for every κ-FCTS $\mathfrak{F}$. Thus the maps $I \mapsto \mathbf{FCTS}_\kappa(I)$ and

$\mathfrak{F} \mapsto I_{\mathfrak{F}}$ are mutually inverse bijections between the set of all images with finite κ-connected domains and the set of all κ-FCTSs.

Consequently, a figure (such as Fig. 2.3) that shows an image I and its associated κ-FCTS $\mathbf{FCTS}_\kappa(I)$ can also be construed as showing the κ-FCTS $\mathfrak{F} = \mathbf{FCTS}_\kappa(I)$ and its associated image $I_{\mathfrak{F}} = I$.

2.3 The (λ, k)-Simplification of a κ-FCTS, Essential Isomorphism, and the Main Theorem

As mentioned earlier, the foreground component tree structure $\mathbf{FCTS}_\kappa(I)$ is too sensitive to errors in the image I to be a good descriptor. In this section we propose a method of simplifying $\mathbf{FCTS}_\kappa(I)$ that is provably robust, in the sense that the simplified κ-FCTS of I remains essentially the same when I is slightly perturbed. We begin by defining some further terminology and notation.

Let $\mathscr{T}$ be any rooted tree, and $\mathfrak{F} = (\mathscr{T}, \ell)$ a κ-FCTS. Then the set of all critical nodes of $\mathscr{T}$ will be denoted by $\mathbf{Crit}(\mathscr{T})$ or $\mathbf{Crit}(\mathfrak{F})$. The node in $\mathbf{Crit}(\mathscr{T})$ that is an ancestor in $\mathscr{T}$ of every node in $\mathbf{Crit}(\mathscr{T})$ will be called the *lowest critical node* or $\mathbf{LCN}$ of $\mathscr{T}$ or $\mathfrak{F}$, and denoted by $\mathbf{LCN}(\mathscr{T})$ or $\mathbf{LCN}(\mathfrak{F})$.

For any subset $\mathbf{V}$ of $\mathbf{Nodes}(\mathscr{T})$ that does not contain every ancestor of $\mathbf{LCN}(\mathscr{T})$, there is a κ-FCTS $(\mathscr{T}', \ell')$ such that $\mathbf{Nodes}(\mathscr{T}') = \mathbf{Nodes}(\mathscr{T}) \setminus \mathbf{V}$ and ℓ' is the restriction of ℓ to $\mathbf{Nodes}(\mathscr{T}')$. This κ-FCTS will be denoted by $\mathfrak{F} - \mathbf{V}$.

We write $\mathfrak{F}' \sqsubseteq \mathfrak{F}$ to mean that $\mathfrak{F}' = \mathfrak{F} - \mathbf{V}$ for some $\mathbf{V} \subseteq \mathbf{Nodes}(\mathscr{T}) \setminus \{\mathbf{root}(\mathscr{T})\}$. Thus $\mathfrak{F}' \sqsubseteq \mathfrak{F}$ implies that $\mathbf{root}(\mathfrak{F}') = \mathbf{root}(\mathfrak{F})$ and that $\mathbf{Nodes}(\mathfrak{F}') \subseteq \mathbf{Nodes}(\mathfrak{F})$.

We write $\mathscr{T}^{\mathbf{crit}}$ to denote the rooted tree whose set of nodes is $\mathbf{Crit}(\mathscr{T}) \cup \{\mathbf{root}(\mathscr{T})\}$ in which a node $\mathbf{u}$ is an ancestor of a node $\mathbf{v}$ if and only if $\mathbf{u}$ is an ancestor of $\mathbf{v}$ in $\mathscr{T}$. Thus $\mathbf{root}(\mathscr{T}^{\mathbf{crit}}) = \mathbf{root}(\mathscr{T})$, $\mathbf{LCN}(\mathscr{T}^{\mathbf{crit}}) = \mathbf{LCN}(\mathscr{T})$, and $\mathbf{Crit}(\mathscr{T}^{\mathbf{crit}}) = \mathbf{Crit}(\mathscr{T})$. If $\mathbf{LCN}(\mathscr{T}) \neq \mathbf{root}(\mathscr{T})$ then $\mathbf{LCN}(\mathscr{T}^{\mathbf{crit}}) = \mathbf{LCN}(\mathscr{T})$ is the unique child of $\mathbf{root}(\mathscr{T}^{\mathbf{crit}}) = \mathbf{root}(\mathscr{T})$ in $\mathscr{T}^{\mathbf{crit}}$. The κ-FCTS $(\mathscr{T}^{\mathbf{crit}}, \ell^{\mathbf{crit}})$, where $\ell^{\mathbf{crit}}$ is the restriction of ℓ to $\mathbf{Nodes}(\mathscr{T}^{\mathbf{crit}})$, will be denoted by $\mathfrak{F}^{\mathbf{crit}}$. Note that $\mathfrak{F}^{\mathbf{crit}} \sqsubseteq \mathfrak{F}$. This concept is illustrated in Figs. 2.4 and 2.6.

Using this terminology, our method of simplifying $\mathbf{FCTS}_\kappa(I)$ can be stated as follows:

Let $\mathfrak{F}_0 = (\mathscr{T}_0, \ell_0)$ be any κ-FCTS. Then, for every positive real value λ and every nonnegative integer $k < |\mathbf{root}(\mathscr{T}_0)|$, we define the (λ, k)-*simplification* of $\mathfrak{F}_0$ to be the κ-FCTS $\mathfrak{F}_3$ that can be obtained from $\mathfrak{F}_0$ in three steps, as follows:

Step 1: Prune $\mathfrak{F}_0$ by removing nodes of size $\leq k$, to produce $\mathfrak{F}_1 \sqsubseteq \mathfrak{F}_0$.
Step 2: Prune $\mathfrak{F}_1$ by removing branches of length $\leq \lambda$, to produce $\mathfrak{F}_2 \sqsubseteq \mathfrak{F}_1$.
Step 3: Eliminate internal edges of length $\leq \lambda$ from $\mathfrak{F}_2^{\mathbf{crit}}$, to produce the final κ-FCTS $\mathfrak{F}_3 \sqsubseteq \mathfrak{F}_2^{\mathbf{crit}}$.

With the possible exception of the root, every non-leaf node of the final κ-FCTS $\mathfrak{F}_3$ is a critical node both of $\mathfrak{F}_3$ and of the original κ-FCTS $\mathfrak{F}_0$.

Step 1 is one of the filtering methods proposed in Sect. VI of [7]. It is defined as follows: The result of pruning the κ-FCTS $\mathfrak{F}_0 = (\mathscr{T}_0, \ell_0)$ by removing nodes of

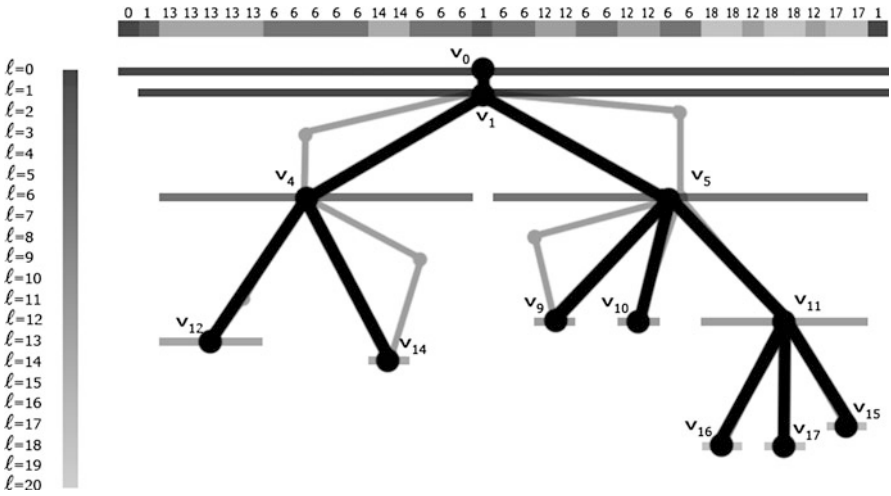


Fig. 2.4 The *thick black edges* are the edges of the FCTS $\mathfrak{F}^{\mathbf{crit}}$, where $\mathfrak{F}$ is the FCTS that is shown in Fig. 2.6 below. Nodes and edges of $\mathfrak{F}$ are shown in *gray*, but may be hidden by nodes and edges of $\mathfrak{F}^{\mathbf{crit}}$—for example, the edges of $\mathfrak{F}$ that join v_4 to v_8 and v_8 to v_{12} in Fig. 2.6 are not visible in this figure because they are hidden by the edge of $\mathfrak{F}^{\mathbf{crit}}$ that joins v_4 to v_{12}. (The image $I_{\mathfrak{F}^{\mathbf{crit}}}$ of $\mathfrak{F}^{\mathbf{crit}}$ is shown at the *top*)

size $\leq k$ is just the κ-FCTS

$$\mathfrak{F}_0 - \left\{\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}_0) \;\middle|\; |\mathbf{v}| \leq k\right\}$$

where, as usual, $|\mathbf{v}|$ denotes the cardinality of the set $\mathbf{v}$—i.e., the number of spels in $\mathbf{v}$. Note that the result is just $\mathfrak{F}_0$ itself if $k = 0$. Figure 2.5 shows an FCTS that has been obtained by pruning the FCTS of Fig. 2.3 in this way.

Precise definitions of steps 2 and 3 of (λ, k)-simplification will be given in Sects. 2.4 and 2.5 below.

While our simplification method is somewhat similar to the method of [10], it has the robustness properties that are stated in Theorem 1 and Corollary 2 below (which the method of [10] does not have). We now introduce terminology and notation that will be used to state these two results.

We say that two κ-FCTSs $\mathfrak{F}_\mathrm{a} = (\mathscr{T}_\mathrm{a}, \ell_\mathrm{a})$ and $\mathfrak{F}_\mathrm{b} = (\mathscr{T}_\mathrm{b}, \ell_\mathrm{b})$ are *essentially isomorphic* if the subtree of $\mathscr{T}_\mathrm{a}^{\mathbf{crit}}$ that is rooted at $\mathbf{LCN}(\mathscr{T}_\mathrm{a})$ is isomorphic to the subtree of $\mathscr{T}_\mathrm{b}^{\mathbf{crit}}$ that is rooted at $\mathbf{LCN}(\mathscr{T}_\mathrm{b})$. Thus $\mathfrak{F}_\mathrm{a}$ and $\mathfrak{F}_\mathrm{b}$ are essentially isomorphic if and only if there exists a mapping $\theta : \mathbf{Crit}(\mathscr{T}_\mathrm{a}) \to \mathbf{Crit}(\mathscr{T}_\mathrm{b})$ such that $\theta[\mathbf{Crit}(\mathscr{T}_\mathrm{a})] = \mathbf{Crit}(\mathscr{T}_\mathrm{b})$ and, for all $\mathbf{v}, \mathbf{v}' \in \mathbf{Crit}(\mathscr{T}_\mathrm{a})$, $\mathbf{v} \preceq_{\mathscr{T}_\mathrm{a}} \mathbf{v}'$ if and only if $\theta(\mathbf{v}) \preceq_{\mathscr{T}_\mathrm{b}} \theta(\mathbf{v}')$. (The latter property implies that θ is 1-to-1.) Any such θ will be called an *essential isomorphism* of $\mathfrak{F}_\mathrm{a}$ to $\mathfrak{F}_\mathrm{b}$.

Note that if the rooted trees $\mathscr{T}_\mathrm{a}^{\mathbf{crit}}$ and $\mathscr{T}_\mathrm{b}^{\mathbf{crit}}$ are isomorphic, then $\mathfrak{F}_\mathrm{a} = (\mathscr{T}_\mathrm{a}, \ell_\mathrm{a})$ and $\mathfrak{F}_\mathrm{b} = (\mathscr{T}_\mathrm{b}, \ell_\mathrm{b})$ are certainly essentially isomorphic. The converse is almost but not quite true. The only way in which $\mathfrak{F}_\mathrm{a} = (\mathscr{T}_\mathrm{a}, \ell_\mathrm{a})$ and $\mathfrak{F}_\mathrm{b} = (\mathscr{T}_\mathrm{b}, \ell_\mathrm{b})$ could be essentially isomorphic without $\mathscr{T}_\mathrm{a}^{\mathbf{crit}}$ and $\mathscr{T}_\mathrm{b}^{\mathbf{crit}}$ being isomorphic is if the root is

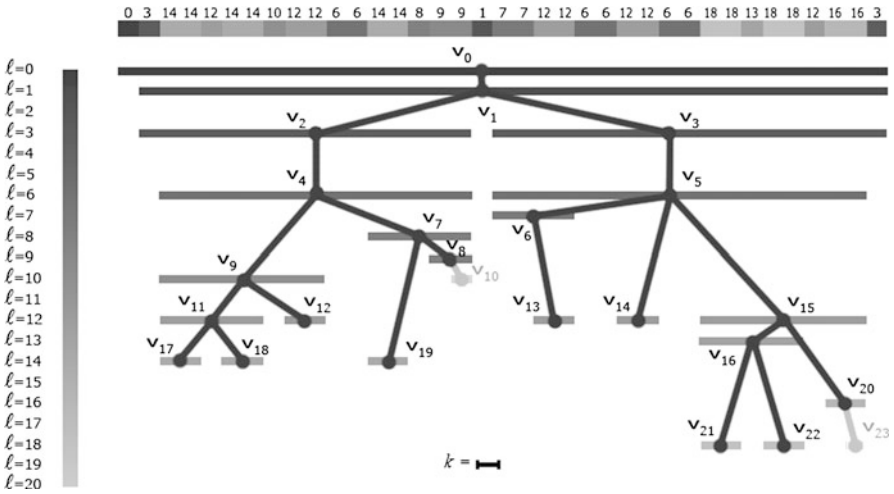


Fig. 2.5 The effect of pruning the FCTS of Fig. 2.3 by removing nodes of size $\leq k$ is shown, in the case $k = 1$; the *black edges* are the edges of the resulting FCTS. Just two nodes (v_{10} and v_{23}) are removed from the tree of Fig. 2.3, as these are the only nodes of that tree that consist of no more than k pixels (i.e., no more than 1 pixel, since $k = 1$). The image of the resulting FCTS is shown at the *top*: Note that the graylevel of the second pixel from the right has changed from 18 in Fig. 2.3 to 16 here; this reflects the removal of v_{23} from the tree. Similarly, the graylevel of the 17th pixel from the left has changed from 10 to 9; this reflects the removal of v_{10}. The graylevels of the other 35 pixels are the same as in Fig. 2.3

the same as the **LCN** in one of the trees but not in the other, and when we remove the root from the latter tree (so its **LCN** becomes its root) it becomes isomorphic to the former tree—e.g., if $\mathscr{T}_a^{\mathbf{crit}}$ has the structure $\wedge$ but $\mathscr{T}_b^{\mathbf{crit}}$ has the structure $\curlywedge$.

For any $\delta \geq 0$, if an essential isomorphism θ of $\mathfrak{F}_a$ to $\mathfrak{F}_b$ satisfies the condition $|\ell_b(\theta(\mathbf{x})) - \ell_a(\mathbf{x})| \leq \delta$ for all $\mathbf{x} \in \mathbf{Crit}(\mathfrak{F}_a)$, then we say that θ is *level-preserving to within* δ. Evidently, the inverse of any essential isomorphism of $\mathfrak{F}_a$ to $\mathfrak{F}_b$ that is level-preserving to within δ will be an essential isomorphism of $\mathfrak{F}_b$ to $\mathfrak{F}_a$ that is level-preserving to within δ.

If an essential isomorphism θ of $\mathfrak{F}_a$ to $\mathfrak{F}_b$ is level-preserving to within 0 (i.e., if $\ell_b(\theta(\mathbf{x})) = \ell_a(\mathbf{x})$ for all $\mathbf{x} \in \mathbf{Crit}(\mathfrak{F}_a)$), then we say that θ is *level-preserving*.

Example 2 The FCTS shown in Fig. 2.6 is essentially isomorphic to the FCTS shown by the thick black edges in Fig. 2.8. Indeed, if $(\mathscr{T}, \ell)$ is the FCTS shown in Fig. 2.6, and $(\mathscr{T}_*, \ell_*)$ is the FCTS shown by the thick black edges in Fig. 2.8, then $(\mathscr{T}, \ell)^{\mathbf{crit}}$ is the FCTS shown in Fig. 2.4, and $(\mathscr{T}_*, \ell_*)^{\mathbf{crit}} = (\mathscr{T}_*, \ell_*)$. It is evident from a quick glance at Figs. 2.4 and 2.8 that $\mathscr{T}^{\mathbf{crit}}$ is isomorphic to $\mathscr{T}_*^{\mathbf{crit}} = \mathscr{T}_*$, so that $(\mathscr{T}, \ell)$ is essentially isomorphic to $(\mathscr{T}_*, \ell_*)$, as we claimed. It is readily confirmed that the mapping $\theta : \mathbf{Crit}(\mathscr{T}) \to \mathbf{Crit}(\mathscr{T}_*)$ which respectively maps

$v_1, v_4, v_5, v_9, v_{10}, v_{11}, v_{12}, v_{14}, v_{15}, v_{16}, v_{17}$ in Fig. 2.6 (or Fig. 2.4)

to $v_1, v_4, v_5, v_{13}, v_{14}, v_{15}, v_{17}, v_{19}, v_{20}, v_{21}, v_{22}$ in Fig. 2.8

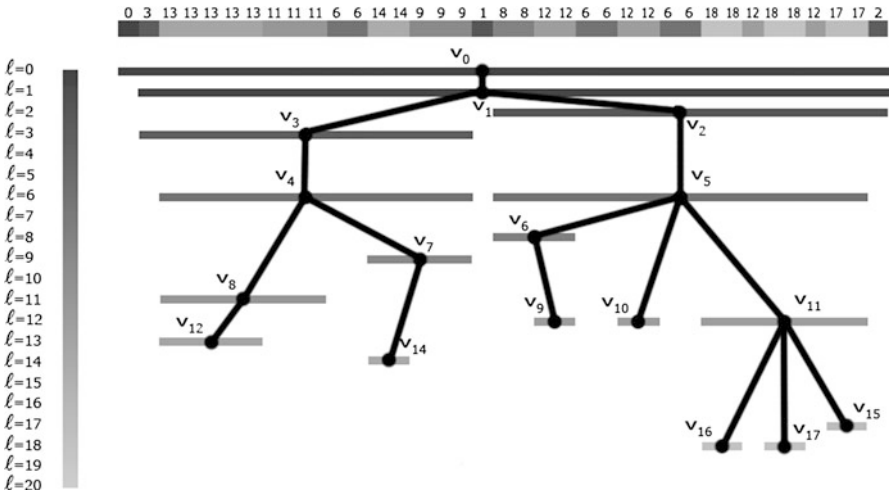


Fig. 2.6 If I is the image at the *top* (and κ is the same adjacency relation as in Figs. 2.3 and 2.5), then $\Lambda_\kappa(I) = 5$ and $K_\kappa(I) = 2$. $\Lambda_\kappa(I) = 5$ because, writing $(\mathscr{T}, \ell)$ for the κ-FCTS of I (which is shown in this figure), $\mathscr{T}$ has critical nodes v_i and v_j such that $\mathsf{v}_i \succ_{\mathscr{T}} \mathsf{v}_j$ and $\ell(\mathsf{v}_i) - \ell(\mathsf{v}_j) = 5$ (e.g., $(\mathsf{v}_i, \mathsf{v}_j) = (\mathsf{v}_4, \mathsf{v}_1)$), but $\mathscr{T}$ has no critical nodes v_i and v_j such that $\mathsf{v}_i \succ_{\mathscr{T}} \mathsf{v}_j$ and $\ell(\mathsf{v}_i) - \ell(\mathsf{v}_j) < 5$. $K_\kappa(I) = 2$ because $\mathscr{T}$ has a node (e.g., v_9) that consists of just 2 pixels, but no node of $\mathscr{T}$ consists of fewer than 2 pixels

is an essential isomorphism of $(\mathscr{T}, \ell)$ to $(\mathscr{T}_*, \ell_*)$. The essential isomorphism θ is not level-preserving, since $|\ell_*(\theta(\mathbf{x})) - \ell(\mathbf{x})| = 1$ when $\mathbf{x} = \mathsf{v}_{12}$ and when $\mathbf{x} = \mathsf{v}_{15}$; indeed, $\ell(\mathsf{v}_{12}) = 13$ but $\ell_*(\theta(\mathsf{v}_{12})) = 14$, and $\ell(\mathsf{v}_{15}) = 17$ but $\ell_*(\theta(\mathsf{v}_{15})) = 16$. But it is readily confirmed that $\ell_*(\theta(\mathbf{x})) = \ell(\mathbf{x})$ for all $\mathbf{x} \in \mathbf{Crit}(\mathscr{T}) \setminus \{\mathsf{v}_{12}, \mathsf{v}_{15}\}$, and so θ is level-preserving to within 1.

Let $I : \mathcal{S} \to \mathbb{R}$ be an image whose domain $\mathcal{S}$ is finite and κ-connected, and let $(\mathscr{T}, \ell) = \mathbf{FCTS}_\kappa(I)$. Then we define:

$$K_\kappa(I) = \min_{s \in \mathcal{S}} \left| \mathcal{C}_\kappa(s, I) \right| = \min_{\mathbf{v} \in \mathbf{Leaves}(\mathscr{T})} |\mathbf{v}|$$

$$\Lambda_\kappa(I) = \min \left\{ \ell(\mathbf{u}) - \ell(\mathbf{v}) \;\middle|\; \mathbf{u}, \mathbf{v} \in \mathbf{Crit}(\mathscr{T}) \text{ and } \mathbf{u} \succ_{\mathscr{T}} \mathbf{v} \right\}$$

These concepts are illustrated in Fig. 2.6.

If $I : \mathcal{S} \to \mathbb{R}$ and $I' : \mathcal{S} \to \mathbb{R}$ are two images that have the same domain $\mathcal{S}$, then the value $\max_{s \in \mathcal{S}} |I'(s) - I(s)|$ will be denoted by $\|I' - I\|_\infty$.

Using this notation, we now state our principal robustness result regarding (λ, k)-simplification (a result which we will generalize in Corollary 2):

Theorem 1 (Main Theorem) *Let κ be any adjacency relation, $I : \mathcal{S} \to \mathbb{R}$ any image whose domain $\mathcal{S}$ is finite and κ-connected, k any integer such that $0 \le k < K_\kappa(I)$, and λ any value such that $0 < \lambda < \Lambda_\kappa(I)/2$. Let $I' : \mathcal{S} \to \mathbb{R}$ be an image such that $\|I' - I\|_\infty \le \lambda/2$. Then there is an essential isomorphism of the (λ, k)-simplification of $\mathbf{FCTS}_\kappa(I')$ to $\mathbf{FCTS}_\kappa(I)$ that is level-preserving to within $\lambda/2$.*

A proof of this theorem is given in Appendix B. In the theorem, and in Corollary 2 below, we may think of the image $I : \mathcal{S} \to \mathbb{R}$ as an ideal or perfect image of some object (such as a macromolecule) at a certain level of detail/resolution, and think of the image I' as an imperfect noisy approximation to the ideal image I (such as an EM map of the same object). We may suppose that the ideal image I is not available to us (and we do not know the exact structure of $\mathbf{FCTS}_\kappa(I)$), but the imperfect image I' is available and we can therefore construct $\mathbf{FCTS}_\kappa(I')$. Theorem 1 and Corollary 2 assure us that, if I' is "sufficiently similar" to I, then there will be values of λ and k for which the (λ, k)-simplification of $\mathbf{FCTS}_\kappa(I')$ is essentially isomorphic to $\mathbf{FCTS}_\kappa(I)$.

For this purpose it follows from Theorem 1 that the imperfect noisy approximation I' will be "sufficiently similar" to the ideal image I if there is no spel in $\mathcal{S}$ at which the value of I' differs from the value of I by $\Lambda_\kappa(I)/4$ or more. Additionally, it will follow from Corollary 2 (as we shall explain in Example 4) that I' might be sufficiently similar to I even if this condition is violated at a small number of spels whose values in I and I' may differ by arbitrarily large amounts.

Example 3 To illustrate Theorem 1, let I be the image that is shown in Fig. 2.6, and let I' be the image that is shown in Fig. 2.3. Then $\|I' - I\|_\infty = 1$, because there exists a pixel p (e.g., any of the three rightmost pixels in the domain) for which $|I'(p) - I(p)| = 1$, but there is no pixel p for which $|I'(p) - I(p)| > 1$. Now let $\lambda = 2$ and $k = 1$. As we observe in the caption of Fig. 2.6, $\Lambda_\kappa(I) = 5$ and $K_\kappa(I) = 2$, so the conditions $\lambda < \Lambda_\kappa(I)/2$, $k < K_\kappa(I)$, and $\|I' - I\|_\infty \leq \lambda/2$ that appear in Theorem 1 are satisfied. Thus the theorem says that there is an essential isomorphism of the (λ, k)-simplification of $\mathbf{FCTS}_\kappa(I')$ to $\mathbf{FCTS}_\kappa(I)$ that is level-preserving to within $\lambda/2 = 1$. In fact the inverse of the mapping θ defined in Example 2 above is just such an essential isomorphism! That is because (as we will see in Sect. 2.5) the FCTS shown by the thick black edges in Fig. 2.8 is exactly the (λ, k)-simplification of $\mathbf{FCTS}_\kappa(I')$.

From Theorem 1, it is easy to deduce Corollary 2 below. Theorem 1 is essentially the case of Corollary 2 in which $k^* = 0$ and $I^* = I$.

As mentioned above, one can think of I in Theorem 1 and Corollary 2 as a perfect or ideal image, and think of I' as an imperfect approximation to I. Theorem 1 is applicable only if the graylevel of *every* spel in I' is close to (specifically, within less than $\Lambda_\kappa(I)/4$ of) that spel's graylevel in I. Corollary 2 is more general; as we will see in Example 4 below, it may be applicable even if there are exceptional spels at which I''s graylevel is much lower or higher than I's graylevel.

Corollary 2 *Let $I : \mathcal{S} \to \mathbb{R}$ and $I' : \mathcal{S} \to \mathbb{R}$ be images on the same finite κ-connected domain $\mathcal{S}$. For any nonnegative integer $k < |\mathcal{S}|$, let I'_k denote the image of the κ-FCTS that results from pruning $\mathbf{FCTS}_\kappa(I')$ by removing nodes of size $\leq k$. Suppose there is an image $I^* : \mathcal{S} \to \mathbb{R}$ such that there exists a level-preserving essential isomorphism of $\mathbf{FCTS}_\kappa(I^*)$ to $\mathbf{FCTS}_\kappa(I)$, and there exists a nonnegative integer $k^* < K_\kappa(I^*)$ for which the image I' satisfies $\|I'_{k^*} - I^*\|_\infty < \Lambda_\kappa(I)/4$. Then,*

for any positive λ *and integer* k *such that* $2\|I'_{k^*} - I^*\|_\infty \leq \lambda < \Lambda_\kappa(I)/2$ *and* $k^* \leq k < K_\kappa(I^*)$, *there is an essential isomorphism of the* (λ, k)*-simplification of* $\mathbf{FCTS}_\kappa(I')$ *to* $\mathbf{FCTS}_\kappa(I)$ *that is level-preserving to within* $\lambda/2$.

Proof of Corollary 2, assuming Theorem 1 Let k be an integer such that $k^* \leq k < K_\kappa(I^*)$, and λ a positive value such that $2\|I'_{k^*} - I^*\|_\infty \leq \lambda < \Lambda_\kappa(I)/2$.

Now $\mathbf{FCTS}_\kappa(I'_{k^*})$ is the result of applying step 1 of (λ, k^*)-simplification to $\mathbf{FCTS}_\kappa(I')$. It follows that the (λ, k)-simplification of $\mathbf{FCTS}_\kappa(I')$ is the same as the (λ, k)-simplification of $\mathbf{FCTS}_\kappa(I'_{k^*})$ (since applying simplification step 1 twice in succession with parameter k^* and then k has the same effect as applying step 1 just once with the parameter $\max(k^*, k) = k$). To prove the corollary, we need to show that there is an essential isomorphism of this κ-FCTS (i.e., the (λ, k)-simplification of $\mathbf{FCTS}_\kappa(I'_{k^*})$) to $\mathbf{FCTS}_\kappa(I)$ that is level-preserving to within $\lambda/2$.

We have that $\Lambda_\kappa(I) = \Lambda_\kappa(I^*)$, since there is a level-preserving essential isomorphism of $\mathbf{FCTS}_\kappa(I^*)$ to $\mathbf{FCTS}_\kappa(I)$. Thus we have that $\lambda < \Lambda_\kappa(I^*)/2$. Moreover, $\|I'_{k^*} - I^*\|_\infty \leq \lambda/2$ and $k < K_\kappa(I^*)$. So, on applying Theorem 1 to I^* and I'_{k^*}, we see that there is an essential isomorphism of the (λ, k)-simplification of $\mathbf{FCTS}_\kappa(I'_{k^*})$ to $\mathbf{FCTS}_\kappa(I^*)$ that is level-preserving to within $\lambda/2$. Composing this essential isomorphism with the level-preserving essential isomorphism of $\mathbf{FCTS}_\kappa(I^*)$ to $\mathbf{FCTS}_\kappa(I)$ gives an essential isomorphism of the (λ, k)-simplification of $\mathbf{FCTS}_\kappa(I'_{k^*})$ to $\mathbf{FCTS}_\kappa(I)$ that is level-preserving to within $\lambda/2$, as required. □

The following example shows how the condition that I' must satisfy in Corollary 2 is much less restrictive than the condition $\|I' - I\|_\infty < \Lambda_\kappa(I)/4$ that I' needs to satisfy for Theorem 1 to be applicable.

Example 4 Let $\mathcal{S}$ be a 3D rectangular array of voxels, and let κ be the 6-adjacency relation on $\mathcal{S}$. Let $I : \mathcal{S} \to \mathbb{R}$ be an image such that, for each threshold $\tau \leq \max_{s \in \mathcal{S}} I(s)$, the members of $\{\mathcal{C}_\kappa(s, I, \tau) \mid I(s) \geq \tau\}$ have fairly compact shapes and are not very small, and no two of the sets are very close together. (Here "have fairly compact shapes" and "are not very small" imply that: (i) removing a very few randomly chosen voxels from a set $\mathcal{C}_\kappa(s, I, \tau)$ is unlikely to split it into two or more pieces, and unlikely to completely eliminate that set. The "no two of the sets are very close" condition implies that: (ii) adding a very few randomly chosen voxels to a set $\mathcal{C}_\kappa(s, I, \tau)$ is unlikely to connect that set to a different set $\mathcal{C}_\kappa(s', I, \tau)$.) Now let I' be an image on $\mathcal{S}$ that is obtained from I by changing the graylevels of a very small number of randomly chosen voxels by arbitrarily large positive and/or negative amounts. Then $\|I' - I\|_\infty < \Lambda_\kappa(I)/4$ will *not* hold unless every graylevel change is smaller in absolute value than $\Lambda_\kappa(I)/4$. But, regardless of the sizes of the graylevel changes, when k^* is the cardinality of the largest 6-connected subset of the set $\{s \in \mathcal{S} \mid I'(s) > I(s)\}$ it is likely (because of (i) and (ii)) that there will be a level-preserving essential isomorphism of $\mathbf{FCTS}_\kappa(I'_{k^*})$ to $\mathbf{FCTS}_\kappa(I)$, in which case the image I' will satisfy the condition of Corollary 2 with $I^* = I'_{k^*}$.

2.4 Pruning by Removing Branches of Length $\leq \lambda$

Step 2 of (λ, k)-simplification is to prune the FCTS that is the result of step 1 by removing branches of length $\leq \lambda$. We now give a mathematical specification of the output of step 2 (properties P1–P4 below), present a result (Proposition 3) that gives us an easily visualized characterization of the output, and then describe (in Sect. 2.4.3) how step 2 can be efficiently implemented.

2.4.1 Specification of Simplification Step 2

Let $\mathscr{T}$ be any rooted tree and let $\mathbf{x} \in \mathbf{Nodes}(\mathscr{T})$. Then we write $\mathbf{x}\Downarrow_{\mathscr{T}}$ to denote the set of all ancestors of $\mathbf{x}$ in $\mathscr{T}$, write $\mathbf{x}\downarrow_{\mathscr{T}}$ to denote the set $\mathbf{x}\Downarrow_{\mathscr{T}} \setminus \{\mathbf{x}\}$ (i.e., the set of all proper ancestors of $\mathbf{x}$ in $\mathscr{T}$), write $\mathbf{x}\Uparrow_{\mathscr{T}}$ to denote the set of all descendants of $\mathbf{x}$ in $\mathscr{T}$, and write $\mathbf{x}\uparrow_{\mathscr{T}}$ to denote the set $\mathbf{x}\Uparrow_{\mathscr{T}} \setminus \{\mathbf{x}\}$ (i.e., the set of all proper descendants of $\mathbf{x}$ in $\mathscr{T}$).

Now let $\emptyset \neq \mathbf{S} \subseteq \mathbf{Nodes}(\mathscr{T})$. Then we write $\bigwedge_{\mathscr{T}} \mathbf{S}$ to denote the *closest common ancestor* of $\mathbf{S}$, by which we mean the node $\mathbf{v}$ of $\mathscr{T}$ such that $\mathbf{v}\Downarrow_{\mathscr{T}} = \bigcap_{\mathbf{u}\in\mathbf{S}} \mathbf{u}\Downarrow_{\mathscr{T}}$, or, equivalently, the element of $\bigcap_{\mathbf{u}\in\mathbf{S}} \mathbf{u}\Downarrow_{\mathscr{T}}$ that is a descendant in $\mathscr{T}$ of every element of that set.

For any κ-FCTS $\mathfrak{F}_{\text{in}} = (\mathscr{T}_{\text{in}}, \ell_{\text{in}})$, we call a sequence $\mathsf{leaf}[1], \ldots, \mathsf{leaf}[\mathsf{n}]$ an ℓ_{in}-*increasing enumeration* of $\mathbf{Leaves}(\mathfrak{F}_{\text{in}})$ if no two of $\mathsf{leaf}[1], \ldots, \mathsf{leaf}[\mathsf{n}]$ are the same, $\{\mathsf{leaf}[1], \ldots, \mathsf{leaf}[\mathsf{n}]\} = \mathbf{Leaves}(\mathfrak{F}_{\text{in}})$ (so that $\mathsf{n} = |\mathbf{Leaves}(\mathfrak{F}_{\text{in}})|$), and $\ell_{\text{in}}(\mathsf{leaf}[1]) \leq \cdots \leq \ell_{\text{in}}(\mathsf{leaf}[\mathsf{n}])$. Pruning a κ-FCTS $\mathfrak{F}_{\text{in}}$ by removing branches of length $\leq \lambda$ is done using such an enumeration of $\mathbf{Leaves}(\mathfrak{F}_{\text{in}})$.

For any $\lambda > 0$, any κ-FCTS $\mathfrak{F}_{\text{in}} = (\mathscr{T}_{\text{in}}, \ell_{\text{in}})$, and any ℓ_{in}-increasing enumeration $\mathsf{leaf}[1], \ldots, \mathsf{leaf}[\mathsf{n}]$ of $\mathbf{Leaves}(\mathfrak{F}_{\text{in}})$, we define the result of pruning $\mathfrak{F}_{\text{in}}$ by removing branches of length $\leq \lambda$ using the leaf enumeration $\mathsf{leaf}[1], \ldots, \mathsf{leaf}[\mathsf{n}]$ to be the κ-FCTS $\mathfrak{F}_{\text{out}}$ that has the following four properties:

P1: $\mathfrak{F}_{\text{out}} \sqsubseteq \mathfrak{F}_{\text{in}}$
P2: $\mathsf{leaf}[\mathsf{n}] \in \mathbf{Leaves}(\mathfrak{F}_{\text{out}})$
P3: For $1 \leq i < \mathsf{n}$, $\mathsf{leaf}[i] \in \mathbf{Leaves}(\mathfrak{F}_{\text{out}})$ if and only if there does *not* exist any $j \in \{i+1, \ldots, \mathsf{n}\}$ for which $\ell_{\text{in}}(\mathsf{leaf}[i]) - \ell_{\text{in}}(\bigwedge_{\mathscr{T}_{\text{in}}}\{\mathsf{leaf}[j], \mathsf{leaf}[i]\}) \leq \lambda$.
P4: $\mathbf{Nodes}(\mathfrak{F}_{\text{out}}) = \bigcup\{\mathsf{leaf}[i]\Downarrow_{\mathscr{T}_{\text{in}}} \mid 1 \leq i \leq \mathsf{n} \text{ and } \mathsf{leaf}[i] \in \mathbf{Leaves}(\mathfrak{F}_{\text{out}})\}$

Given any κ-FCTS $\mathfrak{F}_{\text{in}} = (\mathscr{T}_{\text{in}}, \ell_{\text{in}})$, any $\lambda > 0$, and any ℓ_{in}-increasing enumeration $\mathsf{leaf}[1], \ldots, \mathsf{leaf}[\mathsf{n}]$ of $\mathbf{Leaves}(\mathfrak{F}_{\text{in}})$, it is evident that P1–P4 uniquely determine $\mathfrak{F}_{\text{out}}$. Moreover, even though the result $\mathfrak{F}_{\text{out}}$ of pruning may depend on the leaf enumeration $\mathsf{leaf}[1], \ldots, \mathsf{leaf}[\mathsf{n}]$ that is used, we will see from Proposition 3 that, for any given $\mathfrak{F}_{\text{in}}$ and λ, P1–P4 uniquely determine $\mathfrak{F}_{\text{out}}$ *up to a level-preserving essential isomorphism*.

Figure 2.7 shows an FCTS that has been obtained by pruning the FCTS of Fig. 2.5 in this way.

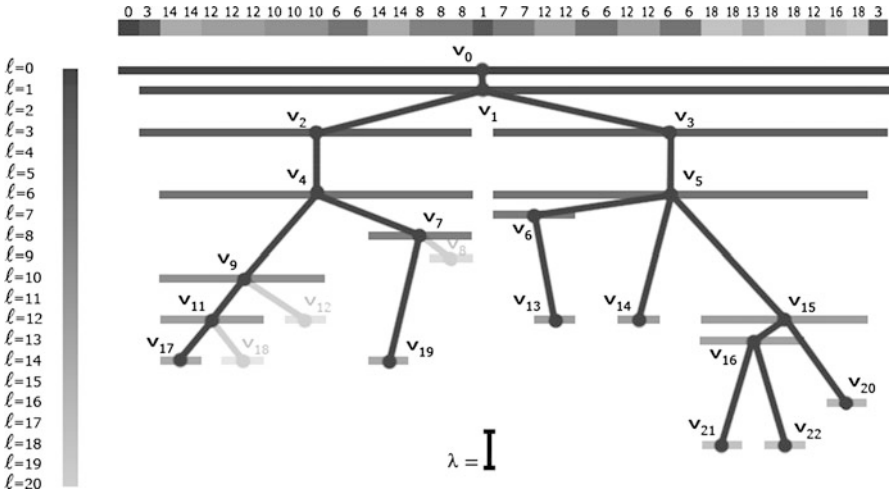


Fig. 2.7 The effect of pruning the FCTS of Fig. 2.5 by removing branches of length λ is shown, in the case $\lambda = 2$; the *black edges* are the edges of the resulting FCTS. Writing $(\mathscr{T}_1, \ell_1)$ for the FCTS of Fig. 2.5, it is assumed that pruning is done using an ℓ_1-increasing leaf enumeration in which the leaf v_{17} of $\mathscr{T}_1$ occurs later than the leaf v_{18}. The leaves v_8, v_{12}, and v_{18} are the only nodes of $\mathscr{T}_1$ that are removed; the leaf v_8 is removed because we have that $\ell_1(\mathsf{v}_8) - \ell_1(\bigwedge_{\mathscr{T}_1}\{\mathsf{v}_{19}, \mathsf{v}_8\}) = \ell_1(\mathsf{v}_8) - \ell_1(\mathsf{v}_7) \leq 2 = \lambda$ (and v_{19} occurs later in the ℓ_1-increasing leaf enumeration than v_8 because $\ell_1(\mathsf{v}_8) < \ell_1(\mathsf{v}_{19})$); v_{12} is removed because $\ell_1(\mathsf{v}_{12}) - \ell_1(\bigwedge_{\mathscr{T}_1}\{\mathsf{v}_{17}, \mathsf{v}_{12}\}) = \ell_1(\mathsf{v}_{12}) - \ell_1(\mathsf{v}_9) \leq 2 = \lambda$; v_{18} is removed because $\ell_1(\mathsf{v}_{18}) - \ell_1(\bigwedge_{\mathscr{T}_1}\{\mathsf{v}_{17}, \mathsf{v}_{18}\}) = \ell_1(\mathsf{v}_{18}) - \ell_1(\mathsf{v}_{11}) \leq 2 = \lambda$ and we are assuming (as mentioned above) that v_{17} occurs later in the ℓ_1-increasing leaf enumeration than v_{18}. In this example no non-leaf nodes of $\mathscr{T}_1$ are removed, as every non-leaf node of $\mathscr{T}_1$ is an ancestor of a leaf of $\mathscr{T}_1$ that is not removed

2.4.2 *An Easily Visualized Characterization of the Output of Simplification Step 2*

The main goal of this section is to present a result (Proposition 3) that is important for the following reasons:

1. It shows that the output of step 2 is independent of the leaf enumeration which is used for pruning (up to a level-preserving essential isomorphism).
2. It gives an easily visualized characterization of the output. (This will be further explained after Proposition 3.)
3. The linear-time implementation of step 2 that is described in Sect. 2.4.3 is based on this result.

For any rooted tree $\mathscr{T}$ and any $\mathbf{x} \in \mathbf{Nodes}(\mathscr{T})$, we write $\mathscr{T}[\mathbf{x}]$ to denote the subtree of $\mathscr{T}$ that is rooted at $\mathbf{x}$.

Now we define some other notation that will be used in Proposition 3. For this purpose, let $\mathfrak{F} = (\mathscr{T}, \ell)$ be any κ-FCTS and λ any positive value. Then we define $\mathrm{depth}_{\mathfrak{F}}(\mathbf{x}) = \max_{\mathbf{y} \in \mathbf{Leaves}(\mathscr{T}[\mathbf{x}])} \ell(\mathbf{y}) - \ell(\mathbf{x})$. Note that $\mathrm{depth}_{\mathfrak{F}}(\mathbf{x}) = 0$ for all $\mathbf{x} \in \mathbf{Leaves}(\mathscr{T})$. We also define:

$$\mathbf{U}^{\lambda}\langle\mathfrak{F}\rangle = \left\{\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}) \;\middle|\; \mathrm{depth}_{\mathfrak{F}}(\mathbf{v}) > \lambda\right\}$$
$$\mathbf{V}^{\lambda}\langle\mathfrak{F}\rangle = \left\{\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}) \;\middle|\; \mathbf{v} \notin \mathbf{U}^{\lambda}\langle\mathfrak{F}\rangle \text{ but } \mathbf{v}{\downarrow_{\mathscr{T}}} \subseteq \mathbf{U}^{\lambda}\langle\mathfrak{F}\rangle\right\}$$

If $\mathbf{U}^{\lambda}\langle\mathfrak{F}\rangle \neq \emptyset$, then $\mathbf{v} \in \mathbf{V}^{\lambda}\langle\mathfrak{F}\rangle$ if and only if $\mathbf{v} \in \mathbf{root}(\mathscr{T}){\uparrow_{\mathscr{T}}}$, $\mathrm{depth}_{\mathfrak{F}}(\mathbf{v}) \leq \lambda$, and $\mathrm{depth}_{\mathfrak{F}}(\mathbf{parent}_{\mathscr{T}}(\mathbf{v})) > \lambda$. If $\mathbf{U}^{\lambda}\langle\mathfrak{F}\rangle = \emptyset$, then $\mathbf{V}^{\lambda}\langle\mathfrak{F}\rangle = \{\mathbf{root}(\mathscr{T})\}$.

For any $\mathbf{x} \in \mathbf{Nodes}(\mathscr{T})$, either $\mathbf{x} \in \mathbf{U}^{\lambda}\langle\mathfrak{F}\rangle$ or $\mathbf{x}$ has a unique ancestor in $\mathbf{V}^{\lambda}\langle\mathfrak{F}\rangle$ (possibly itself), and $\mathbf{x}$ satisfies just one of those conditions. Hence:

$$\mathbf{Nodes}(\mathscr{T}) = \mathbf{U}^{\lambda}\langle\mathfrak{F}\rangle \cup \bigcup_{\mathbf{v} \in \mathbf{V}^{\lambda}\langle\mathfrak{F}\rangle} \mathbf{v}{\Uparrow_{\mathscr{T}}} \tag{2.1}$$

If $\mathbf{U}^{\lambda}\langle\mathfrak{F}\rangle \neq \emptyset$ (so that $\mathbf{root}(\mathscr{T})$ lies in $\mathbf{U}^{\lambda}\langle\mathfrak{F}\rangle$ and not in $\mathbf{V}^{\lambda}\langle\mathfrak{F}\rangle$), then we define:

$$\mathbf{V}_1^{\lambda}\langle\mathfrak{F}\rangle = \left\{\mathbf{v} \in \mathbf{V}^{\lambda}\langle\mathfrak{F}\rangle \;\middle|\; \mathrm{depth}_{\mathfrak{F}}(\mathbf{v}) + \ell(\mathbf{v}) - \ell\big(\mathbf{parent}_{\mathscr{T}}(\mathbf{v})\big) > \lambda\right\}$$

But if $\mathbf{U}^{\lambda}\langle\mathfrak{F}\rangle = \emptyset$, then we define $\mathbf{V}_1^{\lambda}\langle\mathfrak{F}\rangle = \{\mathbf{root}(\mathscr{T})\} = \mathbf{V}^{\lambda}\langle\mathfrak{F}\rangle$.

Let $\sigma = (\mathsf{leaf}[1], \ldots, \mathsf{leaf}[\mathsf{n}])$ be any ℓ-increasing enumeration of the leaves of the tree $\mathscr{T}$, and $\mathbf{v}$ any node of $\mathscr{T}$. Then we define $\mathsf{lastLeaf}_{\sigma}(\mathbf{v}, \mathscr{T})$ to be the leaf of $\mathscr{T}[\mathbf{v}]$ that occurs later in the ℓ-increasing enumeration σ than all other leaves of $\mathscr{T}[\mathbf{v}]$. (If $\mathscr{T}[\mathbf{v}]$ has just one leaf, then $\mathsf{lastLeaf}_{\sigma}(\mathbf{v}, \mathscr{T})$ is that leaf.) Thus we have that $\mathrm{depth}_{\mathfrak{F}}(\mathbf{v}) = \ell(\mathsf{lastLeaf}_{\sigma}(\mathbf{v}, \mathscr{T})) - \ell(\mathbf{v})$. We define $\mathsf{Path}_{\sigma}(\mathbf{v}, \mathscr{T}) = \{\mathbf{x} \in \mathbf{Nodes}(\mathscr{T}) \mid \mathbf{v} \preceq_{\mathscr{T}} \mathbf{x} \preceq_{\mathscr{T}} \mathsf{lastLeaf}_{\sigma}(\mathbf{v}, \mathscr{T})\}$. (Note that if $\mathbf{v}'$ is any node of $\mathscr{T}$ that is neither an ancestor nor a descendant of $\mathbf{v}$ in $\mathscr{T}$, then $\mathsf{lastLeaf}_{\sigma}(\mathbf{v}, \mathscr{T}) \neq \mathsf{lastLeaf}_{\sigma}(\mathbf{v}', \mathscr{T})$ and $\mathsf{Path}_{\sigma}(\mathbf{v}, \mathscr{T}) \cap \mathsf{Path}_{\sigma}(\mathbf{v}', \mathscr{T}) = \emptyset$.)

Using the notation we have just introduced, we now state the main result of this section, which is proved in Appendix A.

Proposition 3 *Let* $\mathfrak{F}_{\mathrm{in}} = (\mathscr{T}_{\mathrm{in}}, \ell_{\mathrm{in}})$ *be any* κ*-FCTS, let* $\lambda > 0$, *and let* $\mathfrak{F}_{\mathrm{out}} = (\mathscr{T}_{\mathrm{out}}, \ell_{\mathrm{out}})$ *be the* κ*-FCTS that results from pruning* $\mathfrak{F}_{\mathrm{in}}$ *by removing branches of length* $\leq \lambda$ *using an* ℓ_{in}*-increasing enumeration* σ *of* $\mathbf{Leaves}(\mathscr{T}_{\mathrm{in}})$. *Then the nodes of* $\mathfrak{F}_{\mathrm{out}}$ *consist just of*:

(i) *The nodes of* $\mathbf{U}^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle$.
(ii) *The nodes of* $\mathsf{Path}_{\sigma}(\mathbf{v}, \mathscr{T}_{\mathrm{in}})$ *for each node* $\mathbf{v}$ *in* $\mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle$.

Now let $\mathfrak{F}_{\mathrm{in}} = (\mathscr{T}_{\mathrm{in}}, \ell_{\mathrm{in}})$, λ, σ, and $\mathfrak{F}_{\mathrm{out}} = (\mathscr{T}_{\mathrm{out}}, \ell_{\mathrm{out}})$ be as in Proposition 3. Since $\mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle \subseteq \mathbf{V}^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle$, and since no node in $\mathbf{V}^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle$ is an ancestor in $\mathscr{T}_{\mathrm{in}}$ of a node in $\mathbf{U}^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle$ or of a different node in $\mathbf{V}^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle$, for all $\mathbf{v} \in \mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle$ we have that $\mathsf{Path}_{\sigma}(\mathbf{v}, \mathscr{T}_{\mathrm{in}}) \cap \mathbf{U}^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle = \emptyset$, and for all distinct $\mathbf{v}, \mathbf{v}' \in \mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle$ we have that $\mathsf{Path}_{\sigma}(\mathbf{v}, \mathscr{T}_{\mathrm{in}}) \cap \mathsf{Path}_{\sigma}(\mathbf{v}', \mathscr{T}_{\mathrm{in}}) = \emptyset$.

Thus Proposition 3 gives us an easily visualized characterization of the nodes of the FCTS $\mathfrak{F}_{\mathrm{out}} = (\mathscr{T}_{\mathrm{out}}, \ell_{\mathrm{out}})$ that results from pruning $\mathfrak{F}_{\mathrm{in}}$ by removing branches of length $\leq \lambda$ using the leaf enumeration σ (and hence an easily visualized characterization of $\mathfrak{F}_{\mathrm{out}}$ itself, since $\mathfrak{F}_{\mathrm{out}} \sqsubseteq \mathfrak{F}_{\mathrm{in}}$).

In Proposition 3, $\mathbf{U}^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle$ and $\mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle$ are determined by $\mathfrak{F}_{\mathrm{in}}$ and λ; they do not depend on σ. For any $\mathbf{v}$ in $\mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\mathrm{in}}\rangle$, the difference in level between $\mathbf{v}$ and the leaf node of $\mathsf{Path}_{\sigma}(\mathbf{v}, \mathscr{T}_{\mathrm{in}})$—i.e., the value of $\ell_{\mathrm{out}}(\mathsf{lastLeaf}_{\sigma}(\mathbf{v}, \mathscr{T}_{\mathrm{in}})) - \ell_{\mathrm{out}}(\mathbf{v}) =$

$\ell_{\text{in}}(\mathsf{lastLeaf}_\sigma(\mathbf{v}, \mathscr{T}_{\text{in}})) - \ell_{\text{in}}(\mathbf{v}) = \mathsf{depth}_{\mathfrak{F}_{\text{in}}}(\mathbf{v})$—also does not depend on σ. So even though the sets $\mathsf{Path}_\sigma(\mathbf{v}, \mathscr{T}_{\text{in}})$ may depend on the leaf enumeration σ, we see from Proposition 3 that $\mathfrak{F}_{\text{out}}$ is uniquely determined by $\mathfrak{F}_{\text{in}}$ and λ up to a level-preserving essential isomorphism.

2.4.3 Linear-Time Implementation of Simplification Step 2

In the rest of this chapter we assume that each FCTS $(\mathscr{T}, \ell)$ we use is represented in such a way that we can find the root of $\mathscr{T}$ in $O(1)$ time and can do all of the following in $O(1)$ time for any node $\mathbf{v}$ of $\mathscr{T}$:

- Create a clone of $\mathbf{v}$, and add it to another FCTS (as a new child of some specified node of the latter).
- Find the parent of $\mathbf{v}$ in $\mathscr{T}$, if $\mathbf{v}$ is not the root.
- Determine the value of $\ell(\mathbf{v})$.
- Determine whether or not $\mathbf{v}$ is a leaf of $\mathscr{T}$.

We also assume that, for any non-leaf node $\mathbf{v}$ of $\mathscr{T}$, we can find all the children of $\mathbf{v}$ in $O(|\mathbf{Children}_{\mathscr{T}}(\mathbf{v})|)$ time.

In the rest of this section we describe simple but efficient implementations of step 2 and of a variant of step 2.

Let $\mathfrak{F}_{\text{in}} = (\mathscr{T}_{\text{in}}, \ell_{\text{in}})$ be some κ-FCTS, and let σ be an ℓ_{in}-increasing leaf enumeration of $\mathbf{Leaves}(\mathscr{T}_{\text{in}})$ such that, whenever $\mathbf{x}$ and $\mathbf{y}$ are leaves of $\mathscr{T}_{\text{in}}$, the answer to the question

$$\text{Does } \mathbf{x} \text{ occur later than } \mathbf{y} \text{ in } \sigma? \tag{2.2}$$

can be determined in $O(1)$ time even if $\ell_{\text{in}}(\mathbf{x}) = \ell_{\text{in}}(\mathbf{y})$.

Our implementation of step 2 runs in $O(|\mathbf{Nodes}(\mathscr{T}_{\text{in}})|)$ time, and does *not* require the actual creation of the sequence σ: We allow σ to be *implicitly* defined by some function $f : \mathbf{Leaves}(\mathscr{T}_{\text{in}}) \times \mathbf{Leaves}(\mathscr{T}_{\text{in}}) \to \{\mathbf{Yes}, \mathbf{No}\}$ such that the answer to (2.2) for any two leaves $\mathbf{x}$ and $\mathbf{y}$ of $\mathscr{T}_{\text{in}}$ is $f(\mathbf{x}, \mathbf{y})$ and this can be computed in $O(1)$ time.[1]

For every $\lambda > 0$ let $\mathfrak{F}_{\text{out},\lambda}$ be the FCTS that should result from pruning $\mathfrak{F}_{\text{in}}$ by removing branches of length $\leq \lambda$ using the leaf enumeration σ. We now explain how $\mathfrak{F}_{\text{out},\lambda}$ can be constructed in $O(|\mathbf{Nodes}(\mathscr{T}_{\text{in}})|)$ time.

For each non-leaf node $\mathbf{w}$ of $\mathscr{T}_{\text{in}}$, we define $\mathsf{next}_\sigma(\mathbf{w}, \mathscr{T}_{\text{in}})$ to be the child of $\mathbf{w}$ in $\mathsf{Path}_\sigma(\mathbf{w}, \mathscr{T}_{\text{in}})$ (i.e., the child of $\mathbf{w}$ that is an ancestor of $\mathsf{lastLeaf}_\sigma(\mathbf{w}, \mathscr{T}_{\text{in}})$); if $\mathbf{w}$ is a leaf of $\mathscr{T}_{\text{in}}$ then we define $\mathsf{next}_\sigma(\mathbf{w}, \mathscr{T}_{\text{in}}) = \mathbf{w}$. During a single postorder traversal $\mathsf{next}_\sigma(\mathbf{w}, \mathscr{T}_{\text{in}})$, $\mathsf{lastLeaf}_\sigma(\mathbf{w}, \mathscr{T}_{\text{in}})$, and $\mathsf{depth}_{\mathfrak{F}_{\text{in}}}(\mathbf{w})$ can be computed for all nodes $\mathbf{w}$ of $\mathscr{T}_{\text{in}}$ in $\sum_{\mathbf{w} \in \mathbf{Nodes}(\mathscr{T}_{\text{in}})} O(1 + |\mathbf{Children}_{\mathscr{T}_{\text{in}}}(\mathbf{w})|) = O(|\mathbf{Nodes}(\mathscr{T}_{\text{in}})|)$ time. Then,

[1] Note that no algorithm which actually creates the sequence σ that is defined by any such function f can run in $O(|\mathbf{Nodes}(\mathscr{T}_{\text{in}})|)$ time in all cases, because any comparison sort must perform $\Omega(n \log n)$ comparisons to sort a set of n items (here, leaves) in the worst case [3, Thm. 8.1].

for any given node $\mathbf{v}$ of $\mathscr{T}_{\text{in}}$ it is easy to determine in $O(1)$ time whether $\mathbf{v}$ belongs to $\mathbf{U}^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle$, to $\mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle$, or to neither of those sets, and it is easy to find all the nodes of $\mathsf{Path}_{\sigma}(\mathbf{v}, \mathscr{T}_{\text{in}})$ by following a chain of $\mathsf{next}_{\sigma}(\mathbf{w}, \mathscr{T}_{\text{in}})$ nodes that starts with $\mathbf{w} = \mathbf{v}$. Hence we can construct $\mathfrak{F}_{\text{out},\lambda}$ in $O(|\mathbf{Nodes}(\mathscr{T}_{\text{in}})|)$ time, for any positive λ that the user may specify, in the following way:

1. Clone $\mathbf{root}(\mathscr{T}_{\text{in}})$, and initialize the output FCTS (i.e., the FCTS that will be output when the algorithm terminates) to be an FCTS whose only node is the clone of $\mathbf{root}(\mathscr{T}_{\text{in}})$.
2. Do a preorder traversal of the subgraph of $\mathscr{T}_{\text{in}}$ that is induced by the set of nodes $\mathbf{U}^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle \cup \mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle$. (This is the rooted tree that is derived from $\mathscr{T}_{\text{in}}$ by ignoring all nodes which do not lie in the set $\mathbf{U}^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle \cup \mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle$. Note that this set contains $\mathbf{root}(\mathscr{T}_{\text{in}})$ and all the ancestors of each node in the set.) When any node $\mathbf{v}$ is visited during the traversal, do the following:
   (2a) If $\mathbf{v} \in \mathbf{U}^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle \setminus \{\mathbf{root}(\mathscr{T}_{\text{in}})\}$, then create a clone of $\mathbf{v}$ and add it to the output FCTS.
   (2b) If $\mathbf{v} \in \mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle$, then find all the nodes of $\mathsf{Path}_{\sigma}(\mathbf{v}, \mathscr{T}_{\text{in}})$ and, for every such node $\mathbf{w}$, create a clone of $\mathbf{w}$ and add it to the output FCTS (unless $\mathbf{w} = \mathbf{root}(\mathscr{T}_{\text{in}})$).

It is evident that $\mathfrak{F}_{\text{out},\lambda}$ can be constructed in this way, since steps (2a) and (2b) will create clones of all nodes of types (i) and (ii) in Proposition 3 (except the root of $\mathscr{T}_{\text{in}}$) and add them to the output FCTS.

Step 3 of (λ, k)-simplification simplifies $\mathfrak{F}^{\mathbf{crit}}$, where $\mathfrak{F}$ is the output of step 2. We can construct $\mathfrak{F}^{\mathbf{crit}}_{\text{out},\lambda}$ directly, without constructing $\mathfrak{F}_{\text{out},\lambda}$, using a modified version of the algorithm described above in which (2a) and (2b) are replaced with:

(2a′) If $\mathbf{v} \in \mathbf{U}^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle \setminus \{\mathbf{root}(\mathscr{T}_{\text{in}})\}$, and $\mathbf{Children}_{\mathscr{T}_{\text{in}}}(\mathbf{v})$ contains two or more nodes in $\mathbf{U}^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle \cup \mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle$, then create a clone of $\mathbf{v}$ and add it to the output FCTS.

(2b′) If $\mathbf{v} \in \mathbf{V}_1^{\lambda}\langle\mathfrak{F}_{\text{in}}\rangle$, then create a clone of the node $\mathsf{lastLeaf}_{\sigma}(\mathbf{v}, \mathscr{T}_{\text{in}})$ and add it to the output FCTS.

Here (2b′) assumes that $\mathscr{T}_{\text{in}}$ has at least two nodes.

2.5 Elimination of Internal Edges of Length $\leq \lambda$ from $\mathfrak{F}^{\mathbf{crit}}$

Step 3 of (λ, k)-simplification is to eliminate internal edges of length $\leq \lambda$ from $\mathfrak{F}^{\mathbf{crit}}$, where $\mathfrak{F}$ is the FCTS that results from step 2 of (λ, k)-simplification. We now mathematically specify the output of step 3, and then present an algorithm which implements step 3.

2.5.1 Specification of Simplification Step 3

Let $\mathfrak{F} = (\mathscr{T}, \ell)$ be any κ-FCTS. Then, for each $\lambda > 0$, the result of eliminating internal edges of length $\leq \lambda$ from $\mathfrak{F}^{\mathbf{crit}}$ is the κ-FCTS $\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle$ that we will define below. The definition will use some notation which we now introduce.

The set $\{\ell(\mathbf{c}) - \ell(\mathbf{c}') \mid \mathbf{c}, \mathbf{c}' \in \mathbf{Crit}(\mathfrak{F}) \setminus \mathbf{Leaves}(\mathfrak{F}) \text{ and } \mathbf{c}' \in \mathbf{c}{\downarrow_{\mathfrak{F}}}\}$ will be denoted by $D(\mathfrak{F})$, and $d_1^{\mathfrak{F}} < d_2^{\mathfrak{F}} < \cdots < d_{|D(\mathfrak{F})|}^{\mathfrak{F}}$ will denote the elements of $D(\mathfrak{F})$ in ascending order. (Note that all elements of $D(\mathfrak{F})$ are positive.) We define $d_0^{\mathfrak{F}} = 0$. For any $\lambda > 0$, we define $\mathrm{pred}_{\mathfrak{F}}(\lambda) = \max\{d \in D(\mathfrak{F}) \cup \{0\} \mid d < \lambda\}$.

Example 5 Let $\mathfrak{F}$ be the FCTS shown in Fig. 2.7. Then we see from Fig. 2.7 that $\mathbf{Crit}(\mathfrak{F}) \setminus \mathbf{Leaves}(\mathfrak{F}) = \{\mathsf{v}_1, \mathsf{v}_4, \mathsf{v}_5, \mathsf{v}_{15}, \mathsf{v}_{16}\}$ and $D(\mathfrak{F}) = \{1, 5, 6, 7, 11, 12\}$. It follows, for example, that, $d_1^{\mathfrak{F}} = 1$, $d_2^{\mathfrak{F}} = 5$, and $\mathrm{pred}_{\mathfrak{F}}(\lambda) = 1$ for $1 < \lambda \leq 5$.

Now we define $\mathfrak{F}^{\mathbf{crit}}\langle 0\rangle = \mathfrak{F}^{\mathbf{crit}}$ and, for all $\lambda > 0$, we recursively define $\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle$ to be the κ-FCTS that has the following five properties:

E1: $\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle \sqsubseteq \mathfrak{F}^{\mathbf{crit}}$
E2: $\mathbf{LCN}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle) = \mathbf{LCN}(\mathfrak{F})$
E3: $\mathbf{Leaves}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle) = \mathbf{Leaves}(\mathfrak{F})$
E4: If $\lambda \notin D(\mathfrak{F})$, then $\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle = \mathfrak{F}^{\mathbf{crit}}\langle\mathrm{pred}_{\mathfrak{F}}(\lambda)\rangle$.
E5: For every $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}) \setminus (\mathbf{Leaves}(\mathfrak{F}) \cup \{\mathbf{LCN}(\mathfrak{F})\} \cup \{\mathbf{root}(\mathfrak{F})\})$ and every $i \in \{0, \ldots, |D(\mathfrak{F})| - 1\}$, we have that $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle d_{i+1}^{\mathfrak{F}}\rangle)$ if and only if $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle d_i^{\mathfrak{F}}\rangle)$ and $\ell(\mathbf{c}) - \ell(\mathbf{parent}_{\mathfrak{F}^{\mathbf{crit}}\langle d_i^{\mathfrak{F}}\rangle}(\mathbf{c})) > d_{i+1}^{\mathfrak{F}}$.

E1 implies that $\mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle) \subseteq \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}) = \mathbf{Crit}(\mathfrak{F}) \cup \{\mathbf{root}(\mathfrak{F})\}$, and also implies that $\mathbf{root}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle) = \mathbf{root}(\mathfrak{F})$.

Example 6 Figure 2.8 shows the FCTS $\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle$ in the case where $\mathfrak{F}$ is the FCTS that is shown in Fig. 2.7 and $1 \leq \lambda < 5$. Here $d_1^{\mathfrak{F}} = 1$ and $d_2^{\mathfrak{F}} = 5$ (as we observed in Example 5). Since $d_1^{\mathfrak{F}} \leq \lambda < d_2^{\mathfrak{F}}$, it follows from E4 that $\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle = \mathfrak{F}^{\mathbf{crit}}\langle d_1^{\mathfrak{F}}\rangle = \mathfrak{F}^{\mathbf{crit}}\langle 1\rangle$. The node v_{16} in Fig. 2.7 is not a node of $\mathfrak{F}^{\mathbf{crit}}\langle d_1^{\mathfrak{F}}\rangle$; indeed, when we put $i = 0$ and $\mathbf{c} = \mathsf{v}_{16}$, the condition $\ell(\mathbf{c}) - \ell(\mathbf{parent}_{\mathfrak{F}^{\mathbf{crit}}\langle d_i^{\mathfrak{F}}\rangle}(\mathbf{c})) > d_{i+1}^{\mathfrak{F}}$ in E5 is not met since $\mathbf{parent}_{\mathfrak{F}^{\mathbf{crit}}\langle d_0^{\mathfrak{F}}\rangle}(\mathsf{v}_{16}) = \mathsf{v}_{15}$ and $\ell(\mathsf{v}_{16}) - \ell(\mathsf{v}_{15}) = 1 = d_1^{\mathfrak{F}}$. But E1–E5 imply that the other 12 nodes of $\mathfrak{F}^{\mathbf{crit}}$ are nodes of $\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle$.

2.5.2 Implementation of Simplification Step 3

It is possible to perform simplification step 3 (i.e., to construct $\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle$ from $\mathfrak{F}^{\mathbf{crit}}$) by direct application of E1–E5. However, this would require computation of the sorted sequence $d_1^{\mathfrak{F}} < d_2^{\mathfrak{F}} < \cdots < d_k^{\mathfrak{F}}$, where $d_k^{\mathfrak{F}}$ is λ or $\mathrm{pred}_{\mathfrak{F}}(\lambda)$ according to whether $\lambda \in D(\mathfrak{F})$ or $\lambda \notin D(\mathfrak{F})$, followed by k tree traversals that successively find the nodes of $\mathfrak{F}^{\mathbf{crit}}\langle d_1^{\mathfrak{F}}\rangle, \mathfrak{F}^{\mathbf{crit}}\langle d_2^{\mathfrak{F}}\rangle, \ldots, \mathfrak{F}^{\mathbf{crit}}\langle d_k^{\mathfrak{F}}\rangle$.

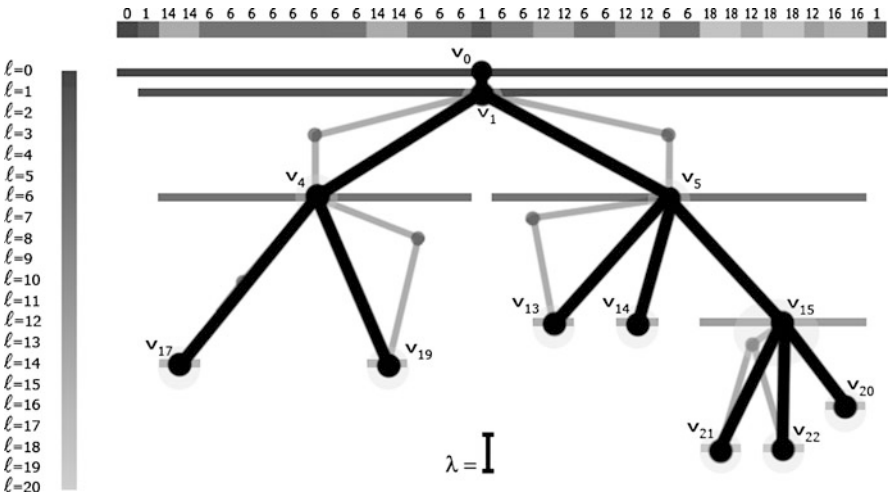


Fig. 2.8 The effect of eliminating internal edges of length $\leq \lambda$ from $\mathfrak{F}^{\mathbf{crit}} = (\mathscr{T}^{\mathbf{crit}}, \ell^{\mathbf{crit}})$ is shown here, in the case where $\mathfrak{F} = (\mathscr{T}, \ell)$ is the FCTS of Fig. 2.7 and $1 \leq \lambda < 5$. The nodes and edges of the resulting FCTS $\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle$ are shown as *fat/thick black nodes and edges*. Other nodes and edges of the tree $\mathscr{T}$ of Fig. 2.7 are colored *gray*, but two of those nodes (v_9 and v_{11} in Fig. 2.7) and three of those edges are partially or completely hidden by the *thick black edge* that joins v_4 to v_{17}. Note that, since 2 is a possible value of λ in this figure, and since $\mathfrak{F}$ is the result of applying steps 1 and 2 of (λ, k)-simplification to the FCTS shown in Fig. 2.3 in the case $\lambda = 2$ and $k = 1$, the FCTS shown in this figure is the $(2, 1)$-simplification of the FCTS shown in Fig. 2.3

Algorithm 1 below, which performs just one tree traversal after the initial cloning step, will usually be a much more efficient implementation of step 3. It inputs a κ-FCTS $\mathfrak{F}_{\text{in}} = (\mathscr{T}_{\text{in}}, \ell_{\text{in}})$ and a positive λ, and constructs $\mathfrak{F}_{\text{in}}^{\mathbf{crit}}\langle\lambda\rangle$ by creating a clone $(\mathscr{T}, \ell)$ of $\mathfrak{F}_{\text{in}}^{\mathbf{crit}} = (\mathscr{T}_{\text{in}}^{\mathbf{crit}}, \ell_{\text{in}}^{\mathbf{crit}})$ and then labeling each node $\mathbf{c}$ of $\mathscr{T}$ with a value $\mathbf{c}.\mathsf{label}$ such that $\mathfrak{F}_{\text{in}}^{\mathbf{crit}}\langle\lambda\rangle = (\mathscr{T}, \ell) - \{\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}) \mid \mathbf{v}.\mathsf{label} \leq \lambda\}$. The correctness of this algorithm is proved in Appendix A.

If we write $h(\mathfrak{F}_{\text{in}}, \lambda)$ to denote the length $l \geq 1$ of the longest chain of nodes $\mathbf{v}_1 \succ_{\mathscr{T}_{\text{in}}} \cdots \succ_{\mathscr{T}_{\text{in}}} \mathbf{v}_l$ in $\mathbf{Crit}(\mathfrak{F}_{\text{in}})$ for which $\ell_{\text{in}}(\mathbf{v}_1) - \ell_{\text{in}}(\mathbf{v}_l) \leq \lambda$, then we see from the initial step "$(\mathscr{T}, \ell) \longleftarrow$ a clone of $(\mathscr{T}_{\text{in}}^{\mathbf{crit}}, \ell_{\text{in}}^{\mathbf{crit}})$" of Algorithm 1 and from the **repeat** ... **until** loop in `labelDescendants` (Procedure 1) that, under the assumptions which are stated at the beginning of Sect. 2.4.3, the running time of Algorithm 1 is $O(|\mathbf{Nodes}(\mathfrak{F}_{\text{in}})| + h(\mathfrak{F}_{\text{in}}, \lambda)\,|\mathbf{Crit}(\mathfrak{F}_{\text{in}})|)$.

2.6 Demonstration of Potential Biological Applicability

To illustrate the potential usefulness of our simplified FCTSs in identifying structural differences between macromolecules, we looked for two structures that are very similar, but not identical. Appropriate data sets were kindly provided by Roberto Marabini of the Universidad Autónoma de Madrid.

Algorithm 1: Eliminate Internal Edges of Length $\leq \lambda$ from $\mathfrak{F}^{\mathbf{crit}}$

inputs: a κ-FCTS $\mathfrak{F}_{\text{in}} = (\mathscr{T}_{\text{in}}, \ell_{\text{in}})$; a positive real value λ
output: a κ-FCTS $\mathfrak{F}_{\text{out}}$ that satisfies $\mathfrak{F}_{\text{out}} \sqsubseteq \mathfrak{F}_{\text{in}}^{\mathbf{crit}}$

$(\mathscr{T}, \ell) \longleftarrow$ a clone of $(\mathscr{T}_{\text{in}}^{\mathbf{crit}}, \ell_{\text{in}}^{\mathbf{crit}})$;
root$(\mathscr{T})$.label $\longleftarrow \infty$;
LCN$(\mathscr{T})$.label $\longleftarrow \infty$;
foreach $\mathbf{x} \in$ **Children**$_{\mathscr{T}}$(**LCN**$(\mathscr{T})$) **do** `labelDescendants`$(\mathbf{x}, \mathscr{T}, \ell, \lambda)$;
$\mathfrak{F}_{\text{out}} \longleftarrow (\mathscr{T}, \ell) - \{\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}) \mid \mathbf{v}.\text{label} \leq \lambda\}$;

Procedure 1: `labelDescendants`$(\mathbf{c}, \mathscr{T}, \ell, \lambda)$

if $\mathbf{c} \notin$ **Leaves**$(\mathscr{T})$ **then**
  $\mathbf{c}' \longleftarrow \mathbf{c}$;
  repeat
    $\mathbf{c}' \longleftarrow$ **parent**$_{\mathscr{T}}(\mathbf{c}')$;
    $\mathbf{c}$.label $\longleftarrow \ell(\mathbf{c}) - \ell(\mathbf{c}')$;
  until ($\mathbf{c}$.label $> \lambda$ **or** $\mathbf{c}$.label $\leq \mathbf{c}'$.label);
  foreach $\mathbf{x} \in$ **Children**$_{\mathscr{T}}(\mathbf{c})$ **do** `labelDescendants`$(\mathbf{x}, \mathscr{T}, \ell, \lambda)$;
else $\mathbf{c}$.label $\longleftarrow \infty$;

These data sets originate from the work of San Martín et al. [9], which investigated some biological questions associated with adenoviruses. These viruses are responsible for a large number of diseases in humans such as gastrointestinal and respiratory infections, but can also be used in gene therapy and vaccine delivery [8]. They have an icosahedral shape with a diameter of approximately 900 Å. At each of the 12 vertices of the icosahedron there is a substructure referred to as a *penton*, and the rest of the surface of the icosahedron consists of 240 *hexons*. To reflect this, our simplified FCTSs of these viruses would be expected to have 252 leaves, one for each penton or hexon. This is indeed the case, as we will see.

In the course of their work, San Martín et al. [9] produced a *mutant* version of the *wildtype* version of the adenovirus they were investigating. The two are identical except for a change in a protein (called IIIa). Surface renderings and central cross-sections of the two versions are shown in Fig. 2.9. We now describe how, in spite of their great similarity, the two versions can be distinguished from each other by an obvious topological difference between their simplified FCTSs.

Each version of the virus studied by San Martín et al. [9] was represented by a grayscale volume image on a $275 \times 275 \times 275$ array of sample points. We further quantized the graylevels in each of these images to a set of just 256 equally spaced values represented by the integers $0, \ldots, 255$, where 0 corresponded to the minimum and 255 the maximum graylevel in the original image. For each resulting image I, we constructed $\mathbf{FCTS}_{\kappa}(I)$ using 6-adjacency as our adjacency relation κ, and computed the (λ, k)-simplification of $\mathbf{FCTS}_{\kappa}(I)$ for various choices of λ and k.

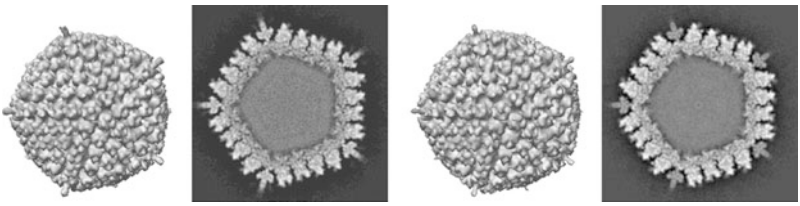


Fig. 2.9 Adenovirus. Surface rendering (**a**) and central cross-section (**b**) of the wildtype version. Surface rendering (**c**) and central cross-section (**d**) of the mutant version

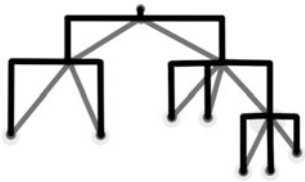


Fig. 2.10 The *gray lines* show an FCTS $(\mathscr{T}, \ell)$ using the tree representation of Figs. 2.1–2.8. The *black lines* show the same FCTS using the tree representation of Fig. 2.11. (In the latter representation, a node that is neither the root nor a leaf is represented by a *horizontal segment*, and an edge from a node **p** to one of its children **c** is represented by a *vertical segment* of length proportional to $\ell(\mathbf{c}) - \ell(\mathbf{p})$)

We found that $\lambda = 10$ and $k = 799$ were good choices that yielded topologically different simplified FCTSs for the two versions of the virus. These simplified FCTSs are shown in Fig. 2.11, using a tree representation that is explained in Fig. 2.10. Each simplified FCTS has 252 leaves, corresponding to the 12 pentons and 240 hexons. For the wildtype version, the lowest critical node is the parent of all 252 leaves; see Fig. 2.11(a). For the mutant version, the lowest critical node is the parent of the 12 leaves that correspond to pentons, but is the grandparent of the 240 leaves that correspond to hexons; see Fig. 2.11(b). These simplified FCTSs indicate that for the mutant version of the virus there is a substantial range of threshold levels (such as level A in Fig. 2.11(b)) at which the pentons are disconnected from each other and from the hexons, but the hexons are connected to each other; for the wildtype version there is no such range of threshold values. Interestingly, San Martín et al. [9] do not mention this difference between the two versions of the virus, although they do point out that in images of the mutant version pentons have lower graylevels than hexons. (The latter can be seen in Fig. 2.9(d), and is also indicated by Fig. 2.11(b); when the image of the mutant virus is thresholded at the graylevel B in Fig. 2.11(b), hexons are observable but pentons are not.)

So our simplified FCTSs may possibly have revealed a previously unobserved difference between the mutant and the wildtype versions of the virus: for the mutant version, there is a substantial range of threshold values at which the hexons are connected to each other, but no penton is connected to a hexon or to another penton. To investigate whether this is a genuine difference between the two versions of the

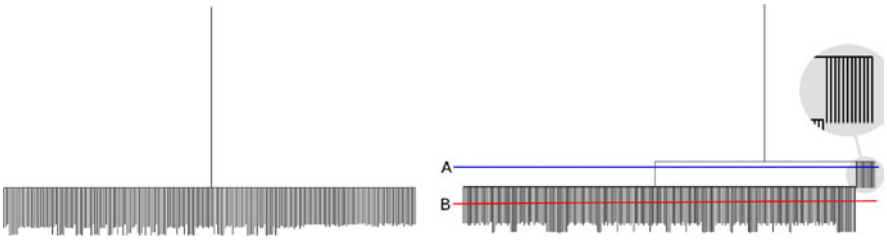


Fig. 2.11 (λ, k)-simplifications of FCTSs of wildtype (**a**) and mutant (**b**) adenoviruses, where $\lambda = 10$ and $k = 799$. (The tree representation used in this figure is explained in Fig. 2.10.) In (**a**), the lowest critical node (represented by the *horizontal line segment*) is the parent of all 252 leaves of the tree. In (**b**), the lowest critical node (represented by the *horizontal line segment* above *line A*) is the parent of the rightmost 12 leaves, which correspond to pentons, but is the grandparent of the other 240 leaves, which correspond to hexons

virus or merely a difference between the specific volume images from which we produced our FCTSs, we carried out a further study.

Ideally, we would have compared simplified FCTSs of, say, 10 independently reconstructed volume images of each version, but such data were not available to us. So we conducted the following approximation of such a study.

For each version of the virus, we randomly selected 2000 out of 3000 available projection images, and used them to reconstruct a volume image on a $275 \times 275 \times 275$ array of points. This was repeated 10 times.

For each of the 20 resulting volume images, we produced a simplified FCTS using the above-mentioned parameters. In each of the 10 simplified FCTSs of the mutant adenovirus, the lowest critical node had 13 children, 12 corresponding to the pentons and the 13th being the root of a subtree whose leaves corresponded to the hexons, as in Fig. 2.11(b). But this was not true of the 10 simplified FCTSs of the wildtype adenovirus; they were all similar to Fig. 2.11(a).

These results provide some evidence to support the hypothesis that images of the mutant version of the virus can be distinguished from images of the wildtype version by the existence in the former (but not the latter) of a substantial range of threshold values with the above-mentioned properties. More investigation would be needed to confirm this hypothesis.

In any event, this example illustrates how our simplified FCTSs may reveal interesting structural differences between two similar macromolecules.

2.7 Possibilities for Future Work

2.7.1 How Can Our Simplification Method and Theorem 1 Be Adapted to Contour Trees?

FCTSs are closely related to contour trees, which are also used to represent images (see, e.g., [12]). Intuitively, a contour tree of an image is an undirected graph each of

whose points represents a *contour*—i.e., a connected component of a level set—of a continuous scalar field derived from the image by interpolation. Contours of scalar fields derived from 3D images are often called *isosurfaces*.

To define the contour tree, let $I : \mathcal{S} \to \mathbb{R}$ be an image whose domain $\mathcal{S}$ is a finite set of points in Euclidean n-space $\mathbb{R}^n$ (for some n). As usual, we refer to the elements of $\mathcal{S}$ as spels. For simplicity in defining the contour tree, we require that I be 1-to-1—i.e., we require that no two spels have exactly the same graylevel in I. (This prevents distinct spels from lying on the same contour, and will allow the contour tree to be defined as a graph whose vertices are spels.) Note that a 1-to-1 image can be produced from any image by making arbitrarily small graylevel perturbations.

For any adjacency relation α on $\mathcal{S}$, we write $\mathbf{Graph}(\alpha)$ to denote the undirected simple graph whose vertex set is $\mathcal{S}$ and whose vertex adjacency relation is α. Recall that an undirected graph is said to be a *tree* if it is connected and acyclic.

We will be considering α-FCTSs of I and its negative image $-I$ (which is obtained from I by multiplying each spel's graylevel by -1). For any $x \in \mathcal{S}$, when discussing $\mathbf{FCTS}_\alpha(I)$ and $\mathbf{FCTS}_\alpha(-I)$ we write $\langle x \rangle$ to denote either the node $\mathcal{C}_\alpha(x, I)$ of $\mathbf{FCTS}_\alpha(I)$ or the node $\mathcal{C}_\alpha(x, -I)$ of $\mathbf{FCTS}_\alpha(-I)$.

Now suppose the adjacency relation α is unknown, but we know $\mathbf{Graph}(\alpha)$ is a tree. Then α is uniquely determined by $\mathbf{FCTS}_\alpha(I)$ and $\mathbf{FCTS}_\alpha(-I)$. Indeed, it is not hard to verify that s is an end vertex of $\mathbf{Graph}(\alpha)$ whose only α-neighbor is s' just if in one of $\mathbf{FCTS}_\alpha(I)$ and $\mathbf{FCTS}_\alpha(-I)$ we have that $\langle s \rangle = \{s\}$ is a leaf whose parent is $\langle s' \rangle$, and in the other of $\mathbf{FCTS}_\alpha(I)$ and $\mathbf{FCTS}_\alpha(-I)$ we have that $\langle s \rangle$ has exactly one child. Further, if s is any end vertex of $\mathbf{Graph}(\boldsymbol{\alpha})$ and the restrictions of I and α to $\mathcal{S} \setminus \{s\}$ are denoted by I' and $\boldsymbol{\alpha}'$, then $\mathbf{FCTS}_{\alpha'}(I') = \mathbf{FCTS}_\alpha(I) - \{\langle s \rangle\}$ and $\mathbf{FCTS}_{\alpha'}(-I') = \mathbf{FCTS}_\alpha(-I) - \{\langle s \rangle\}$, from which α' can be computed (e.g., recursively). Algorithm 4.2 in [1], which is based on these two facts, can be used to construct the tree $\mathbf{Graph}(\alpha)$ in $O(|\mathcal{S}|)$ time, given I, $\mathbf{FCTS}_\alpha(I)$, and $\mathbf{FCTS}_\alpha(-I)$.

To define a contour tree of I, we first choose a "good" adjacency relation κ on $\mathcal{S}$. Let $\mathscr{L}$ be a geometric simplicial complex whose vertex set is $\mathcal{S}$ and whose union is connected and simply connected. Let κ be the adjacency relation on $\mathcal{S}$ such that $(s, t) \in \kappa$ if and only if s and t are the endpoints of an edge of the complex $\mathscr{L}$.

Now let $f : \bigcup \mathscr{L} \to \mathbb{R}$ be the continuous scalar field obtained when we extend the image I by linear interpolation over each simplex of $\mathscr{L}$. Let $\lhd$ be the strict partial order on $\mathcal{S}$ such that $s \lhd s'$ if and only if $I(s) < I(s')$ and there is a path in $\bigcup \mathscr{L}$ from s to s' along which f's value increases monotonically from $I(s)$ to $I(s')$. Let $\alpha(I, \mathscr{L})$ be the adjacency relation on $\mathcal{S}$ such that $(s, s') \in \alpha(I, \mathscr{L})$ if and only if one of the spels s and s' is an immediate successor of the other with respect to $\lhd$. (We say y is an *immediate successor* of x with respect to $\lhd$ if $x \lhd y$ and there is no z such that $x \lhd z \lhd y$.) It can be shown, using the linearity of f on each simplex of $\mathscr{L}$ and, e.g., well known properties of Reeb graphs (see [1, 4]), that $\mathbf{FCTS}_{\alpha(I,\mathcal{L})}(I) = \mathbf{FCTS}_\kappa(I)$ and $\mathbf{FCTS}_{\alpha(I,\mathscr{L})}(-I) = \mathbf{FCTS}_\kappa(-I)$, and that $\mathbf{Graph}(\alpha(I, \mathscr{L}))$ is a tree.

We define the *κ-contour tree*[2] of I to be $\mathbf{Graph}(\alpha(I, \mathscr{L}))$. It follows from our remarks above that this tree is uniquely determined by $\mathbf{FCTS}_\kappa(I)$ and $\mathbf{FCTS}_\kappa(-I)$, and that the tree can be constructed in $O(|\mathcal{S}|)$ time from I and these two κ-FCTSs.

In view of the close relationship between contour trees and FCTSs, we are hopeful that it will be possible to formulate a simplification method for contour trees that is similar to our simplification method for FCTSs and is provably robust in the sense that it can be shown to satisfy an analog of Theorem 1.

2.7.2 Does the Bottleneck Stability Theorem Have an Analog for FCTSs That Implies Theorem 1?

Let $I : \mathcal{S} \to \mathbb{R}$ be any image whose domain $\mathcal{S}$ is finite, and κ any adjacency relation on $\mathcal{S}$ such that $\mathcal{S}$ is κ-connected. A descriptor of I that is related to (but contains less information than) $\mathbf{FCTS}_\kappa(I)$ is the 0th persistence diagram of $-I$ based on the adjacency relation κ. (Here the minus sign reflects the fact that persistence diagrams are defined in terms of the *sublevel* sets of filter functions[3] whereas FCTSs are defined in terms of the *superlevel* sets of images.) The 0th persistence diagram of $-I$ based on κ is a multiset of points in $\mathbb{R} \times (\mathbb{R} \cup \{+\infty\})$ that contains one point for each leaf of $\mathbf{FCTS}_\kappa(I)$. The diagram is easily computed[4] from $\mathbf{FCTS}_\kappa(I)$, but it is not possible to reconstruct $\mathbf{FCTS}_\kappa(I)$ from the diagram.

Step 2 of our simplification method eliminates those leaves of the FCTS that are represented in the 0th persistence diagram by points (x, y) for which $y - x \leq \lambda$. Moreover, for any two images $I, I' : \mathcal{S} \to \mathbb{R}$, the L_∞-distance between the filter functions used to define the 0th persistence diagrams of $-I$ and $-I'$ is $\|I - I'\|_\infty$. For these reasons, our Theorem 1 is vaguely reminiscent of the $p = 0$ case of the Bottleneck Stability Theorem for persistence diagrams [2], [4, p. 182], which states

[2]The tree defined here is the *augmented contour tree* of [1]. It may have many vertices s that have just two neighbors, of which one neighbor s' satisfies $I(s') < I(s)$ while the other neighbor s'' satisfies $I(s'') > I(s)$. Many authors define the contour tree to not include such vertices.

[3]Persistence diagrams are commonly defined (as in [4, pp. 150–152]) for a filter function $f : \mathscr{K} \to \mathbb{R}$, where $\mathscr{K}$ is a suitable simplicial complex. To define the 0th persistence diagram of $-I$ based on the adjacency relation κ, we can take the simplicial complex $\mathscr{K}$ to be the simple graph whose vertex set is $\mathcal{S}$ and whose edges join κ-adjacent elements of $\mathcal{S}$, and we can use the filter function $f : \mathscr{K} \to \mathbb{R}$ for which $f(v) = -I(v)$ if v is a vertex of $\mathscr{K}$, and $f(e) = -\min(I(x), I(y))$ if e is an edge of $\mathscr{K}$ that joins the vertices x and y.

[4]Let $\mathbf{FCTS}_\kappa(I) = (\mathscr{T}, \ell)$, and let $\mathsf{leaf}[1], \ldots, \mathsf{leaf}[\mathsf{n}]$ be any ℓ-increasing enumeration of the leaves of $\mathscr{T}$. For $1 \leq i < \mathsf{n}$, each leaf $\mathsf{leaf}[i]$ is represented in the persistence diagram by a point $(-\ell(\mathsf{leaf}[i]), -\ell(\mathbf{a}))$ where $\mathbf{a}$ is the closest ancestor of $\mathsf{leaf}[i]$ that is an ancestor of at least one of the leaves $\mathsf{leaf}[i+1], \ldots, \mathsf{leaf}[\mathsf{n}]$. The last leaf $\mathsf{leaf}[\mathsf{n}]$ of the ℓ-increasing enumeration is represented in the persistence diagram by the point $(-\ell(\mathsf{leaf}[\mathsf{n}]), +\infty)$. The diagram is defined to also contain, for each $z \in \mathbb{R}$, a point (z, z) with countably infinite multiplicity.

that the bottleneck distance[5] between the pth persistence diagrams of two filter functions cannot exceed the L_∞-distance between those functions.

The Bottleneck Stability Theorem appears not to imply our Theorem 1, because the FCTSs of two images I_1 and I_2 need not be essentially isomorphic even if $-I_1$ and $-I_2$ have the same persistence diagrams. However, it might be possible to prove an analogous stability theorem for FCTSs that does imply Theorem 1.

2.7.3 Can Images Be Simplified Using Variants of Our Method?

In view of the natural bijective correspondence between grayscale images (with finite connected domains) and FCTSs, our method of simplifying FCTSs might also be construed as a method of simplifying images. Unfortunately we have found that, when used for that purpose, it will often be unsatisfactory. (One reason is that the omission of non-critical non-root nodes before performing simplification step 3 may reduce the graylevels of some spels in the resulting image by too much.) Nevertheless, we believe that it may be worthwhile to investigate variants of our method that might be more useful for image simplification.

2.8 Conclusion

FCTSs can be used as descriptors of EM maps and other grayscale images, but unsimplified FCTSs are too sensitive to errors in the image. This chapter has specified a method of simplifying FCTSs that is provably robust (and capable of efficient implementation). Our main theorem and its corollary (Theorem 1 and Corollary 2) conservatively quantify the extent of the method's robustness. We have presented some experimental evidence that the simplified FCTSs produced by our method are useful for the exploration of macromolecular databases. We hope further experimentation will yield more evidence of this or suggest fruitful refinements of our method. Some other avenues for future research have also been discussed.

Acknowledgements We thank Edgar Garduño, Roberto Marabini, and Homeira Pajoohesh for discussions regarding this chapter. The work was supported by awards R01HL070472 from the National Heart, Lung, and Blood Institute and DMS-1114901 from the National Science Foundation. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Heart, Lung, and Blood Institute, the National Institutes of Health, or the National Science Foundation.

[5]The *bottleneck distance* between two persistence diagrams D_1 and D_2 is the infimum of $\sup_{d\in D_1} \|d - \eta(d)\|_\infty$ over all bijections $\eta : D_1 \to D_2$.

Appendix A: Some Properties of Simplification Steps 2 and 3, and a Proof of the Correctness of Algorithm 1

A.1 Properties of Simplification Step 2

Here we prove the main result of Sect. 2.4.2, and establish other properties of simplification step 2 that are used in our proof of the Main Theorem.

Lemma A1 *Let* $\mathfrak{F}_{\text{in}} = (\mathscr{T}_{\text{in}}, \ell_{\text{in}})$ *be any* κ*-FCTS, let* $\lambda > 0$*, and let* $\mathbf{s}$ *and* $\mathbf{s}'$ *be any two distinct leaves of a* κ*-FCTS* $\mathfrak{F}_{\text{out}} = (\mathscr{T}_{\text{out}}, \ell_{\text{out}})$ *that results from pruning* $\mathfrak{F}_{\text{in}}$ *by removing branches of length* $\leq \lambda$*. Then (regardless of which* ℓ_{in}*-increasing enumeration of* $\mathbf{Leaves}(\mathscr{T}_{\text{in}})$ *is used to perform the pruning*):

(i) $\bigwedge_{\mathscr{T}_{\text{out}}}\{\mathbf{s}, \mathbf{s}'\} = \bigwedge_{\mathscr{T}_{\text{in}}}\{\mathbf{s}, \mathbf{s}'\}$
(ii) $\min(\ell_{\text{out}}(\mathbf{s}), \ell_{\text{out}}(\mathbf{s}')) - \ell_{\text{out}}(\bigwedge_{\mathscr{T}_{\text{out}}}\{\mathbf{s}, \mathbf{s}'\}) > \lambda$

Proof The hypotheses imply that properties P1–P4 hold with respect to some ℓ_{in}-increasing enumeration of $\mathbf{Leaves}(\mathscr{T}_{\text{in}})$. It follows from P4 that, for all $\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}_{\text{out}})$, every node in $\mathbf{v}{\Downarrow}_{\mathscr{T}_{\text{in}}}$ is also a node in $\mathbf{v}{\Downarrow}_{\mathscr{T}_{\text{out}}}$. Therefore $\mathbf{v}{\Downarrow}_{\mathscr{T}}$ is the same set regardless of whether $\mathscr{T} = \mathscr{T}_{\text{out}}$ or $\mathscr{T} = \mathscr{T}_{\text{in}}$. So $\bigwedge_{\mathscr{T}}\{\mathbf{s}, \mathbf{s}'\}$ is the same node regardless of whether $\mathscr{T} = \mathscr{T}_{\text{out}}$ or $\mathscr{T} = \mathscr{T}_{\text{in}}$, since $\bigwedge_{\mathscr{T}}\{\mathbf{s}, \mathbf{s}'\}$ is just the element of $\mathbf{s}{\Downarrow}_{\mathscr{T}} \cap \mathbf{s}'{\Downarrow}_{\mathscr{T}}$ that is a descendant in $\mathscr{T}$ of every element of that set. Hence (i) holds.

To prove (ii), we may assume without loss of generality that, in the ℓ_{in}-increasing leaf enumeration that is used for pruning, $\mathbf{s}$ occurs later than $\mathbf{s}'$. (This assumption implies that $\min(\ell_{\text{in}}(\mathbf{s}), \ell_{\text{in}}(\mathbf{s}')) = \ell_{\text{in}}(\mathbf{s}')$.) Then, since $\mathbf{s}' \in \mathbf{Leaves}(\mathscr{T}_{\text{out}})$, property P3 implies that $\ell_{\text{in}}(\mathbf{s}') - \ell_{\text{in}}(\bigwedge_{\mathscr{T}_{\text{in}}}\{\mathbf{s}, \mathbf{s}'\}) > \lambda$, which is equivalent to:

$$\min\big(\ell_{\text{in}}(\mathbf{s}), \ell_{\text{in}}\big(\mathbf{s}'\big)\big) - \ell_{\text{in}}\Big(\bigwedge_{\mathscr{T}_{\text{in}}}\big\{\mathbf{s}, \mathbf{s}'\big\}\Big) > \lambda \tag{A1}$$

But (A1) is equivalent to assertion (ii), because of assertion (i) and the fact that ℓ_{out} is just the restriction of ℓ_{in} to $\mathbf{Nodes}(\mathscr{T}_{\text{out}})$. □

Corollary A2 *Let* λ *be any positive value, and* $\mathfrak{F}_{\text{out}}$ *any* κ*-FCTS that results from pruning a* κ*-FCTS* $\mathfrak{F}_{\text{in}}$ *by removing branches of length* $\leq \lambda$*. Then, for all* $\mathbf{v} \in \mathbf{Crit}(\mathfrak{F}_{\text{out}}) \setminus \mathbf{Leaves}(\mathfrak{F}_{\text{out}})$*, we have that* $\mathbf{v} \in \mathbf{Crit}(\mathfrak{F}_{\text{in}}) \setminus \mathbf{Leaves}(\mathfrak{F}_{\text{in}})$ *and* $\text{depth}_{\mathfrak{F}_{\text{out}}}(\mathbf{v}) > \lambda$.

Proof Let $\mathfrak{F}_{\text{out}} = (\mathscr{T}_{\text{out}}, \ell_{\text{out}})$, and let $\mathbf{v} \in \mathbf{Crit}(\mathfrak{F}_{\text{out}}) \setminus \mathbf{Leaves}(\mathfrak{F}_{\text{out}})$. Then $\mathbf{v} = \bigwedge_{\mathscr{T}_{\text{out}}}\{\mathbf{s}, \mathbf{s}'\}$ for some distinct leaves $\mathbf{s}$ and $\mathbf{s}'$ of $\mathfrak{F}_{\text{out}}$. Now $\mathbf{v} = \bigwedge_{\mathscr{T}_{\text{in}}}\{\mathbf{s}, \mathbf{s}'\}$ (by assertion (i) of Lemma A1), and so $\mathbf{v} \in \mathbf{Crit}(\mathfrak{F}_{\text{in}}) \setminus \mathbf{Leaves}(\mathfrak{F}_{\text{in}})$. Moreover, we have that $\text{depth}_{\mathfrak{F}_{\text{out}}}(\mathbf{v}) \geq \ell_{\text{out}}(\mathbf{s}) - \ell_{\text{out}}(\mathbf{v}) = \ell_{\text{out}}(\mathbf{s}) - \ell_{\text{out}}(\bigwedge_{\mathscr{T}_{\text{out}}}\{\mathbf{s}, \mathbf{s}'\}) > \lambda$, where the second inequality follows from assertion (ii) of Lemma A1. □

Lemma A3 *Let* $\mathfrak{F}_{\text{in}} = (\mathscr{T}_{\text{in}}, \ell_{\text{in}})$ *be a* κ*-FCTS, let* $\lambda > 0$*, and let* $\mathfrak{F}_{\text{out}} = (\mathscr{T}_{\text{out}}, \ell_{\text{out}})$ *be the* κ*-FCTS that results from pruning* $\mathfrak{F}_{\text{in}}$ *by removing branches of length* $\leq \lambda$

using an ℓ_{in}*-increasing leaf enumeration* $\sigma = (\mathsf{leaf}[1], \ldots, \mathsf{leaf}[\mathsf{n}])$ *of* $\mathbf{Leaves}(\mathscr{T}_{\text{in}})$. *Then*:

(a) *For all* $\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}_{\text{in}}) \setminus \mathbf{Nodes}(\mathscr{T}_{\text{out}})$, $\mathbf{v}\Uparrow_{\mathscr{T}_{\text{in}}} \cap \mathbf{Nodes}(\mathscr{T}_{\text{out}}) = \emptyset$.
(b) *For all* $\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}_{\text{in}})$, $\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}_{\text{out}})$ *if and only if* $\mathsf{lastLeaf}_\sigma(\mathbf{v}, \mathscr{T}_{\text{in}}) \in \mathbf{Leaves}(\mathscr{T}_{\text{out}})$.
(c) *For all* $\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}_{\text{out}})$, $\text{depth}_{\mathfrak{F}_{\text{out}}}(\mathbf{v}) = \text{depth}_{\mathfrak{F}_{\text{in}}}(\mathbf{v})$.

Proof For brevity, we write $\mathsf{lastLeaf}_\sigma(\mathbf{v})$ for $\mathsf{lastLeaf}_\sigma(\mathbf{v}, \mathscr{T}_{\text{in}})$. Evidently, (a) follows from P4, and the "if" part of (b) follows from (a). To establish the "only if" part of (b), let $\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}_{\text{out}})$, and let $\mathsf{leaf}[i] = \mathsf{lastLeaf}_\sigma(\mathbf{v})$. We need to show that $\mathsf{leaf}[i] \in \mathbf{Nodes}(\mathscr{T}_{\text{out}})$. If $i = \mathsf{n}$ then this is true (by property P2), so let us assume $i < \mathsf{n}$. Let j be any element of the set $\{i+1, \ldots, \mathsf{n}\}$ (so that $\mathsf{leaf}[j] \notin \mathbf{Leaves}(\mathscr{T}_{\text{in}}[\mathbf{v}])$). Now we claim that $\mathsf{leaf}[j]$ must satisfy $\ell_{\text{in}}(\mathsf{leaf}[i]) - \ell_{\text{in}}(\bigwedge_{\mathscr{T}_{\text{in}}}\{\mathsf{leaf}[j], \mathsf{leaf}[i]\}) > \lambda$.

To see this, let $\mathsf{leaf}[k]$ be any leaf of $\mathscr{T}_{\text{out}}[\mathbf{v}]$; such a leaf must exist, by P4. As $\mathsf{leaf}[i] = \mathsf{lastLeaf}_\sigma(\mathbf{v})$, we have that $i \geq k$ and $\ell_{\text{in}}(\mathsf{leaf}[i]) \geq \ell_{\text{in}}(\mathsf{leaf}[k])$. As $j \in \{i+1, \ldots, \mathsf{n}\}$, we have that $j \in \{k+1, \ldots, \mathsf{n}\}$. Therefore, since $\mathsf{leaf}[k] \in \mathbf{Leaves}(\mathscr{T}_{\text{out}})$, property P3 implies that:

$$\ell_{\text{in}}\big(\mathsf{leaf}[k]\big) - \ell_{\text{in}}\Big(\bigwedge\nolimits_{\mathscr{T}_{\text{in}}}\big\{\mathsf{leaf}[j], \mathsf{leaf}[k]\big\}\Big) > \lambda \tag{A2}$$

But, since $\mathsf{leaf}[i]$ and $\mathsf{leaf}[k]$ are leaves of $\mathscr{T}_{\text{in}}[\mathbf{v}]$ but $\mathsf{leaf}[j]$ is not,

$$\bigwedge\nolimits_{\mathscr{T}_{\text{in}}}\big\{\mathsf{leaf}[j], \mathsf{leaf}[i]\big\} = \bigwedge\nolimits_{\mathscr{T}_{\text{in}}}\big\{\mathsf{leaf}[j], \mathbf{v}\big\} = \bigwedge\nolimits_{\mathscr{T}_{\text{in}}}\big\{\mathsf{leaf}[j], \mathsf{leaf}[k]\big\}$$

and (since $\ell_{\text{in}}(\mathsf{leaf}[i]) \geq \ell_{\text{in}}(\mathsf{leaf}[k])$) this implies:

$$\begin{aligned}&\ell_{\text{in}}\big(\mathsf{leaf}[i]\big) - \ell_{\text{in}}\Big(\bigwedge\nolimits_{\mathscr{T}_{\text{in}}}\big\{\mathsf{leaf}[j], \mathsf{leaf}[i]\big\}\Big)\\ &\quad\geq \ell_{\text{in}}\big(\mathsf{leaf}[k]\big) - \ell_{\text{in}}\Big(\bigwedge\nolimits_{\mathscr{T}_{\text{in}}}\big\{\mathsf{leaf}[j], \mathsf{leaf}[k]\big\}\Big)\end{aligned}$$

This and (A2) imply that our claim is valid (for any j in $\{i+1, \ldots, \mathsf{n}\}$). The "only if" part of (b) follows from this and property P3.

To prove (c), let $\mathbf{v} \in \mathbf{Nodes}(\mathscr{T}_{\text{out}})$. Then $\mathsf{lastLeaf}_\sigma(\mathbf{v}) \in \mathbf{Leaves}(\mathscr{T}_{\text{out}}[\mathbf{v}])$ (by (b)), and every $\mathbf{w} \in \mathbf{Nodes}(\mathscr{T}_{\text{out}}[\mathbf{v}]) \subseteq \mathbf{Nodes}(\mathscr{T}_{\text{in}}[\mathbf{v}])$ satisfies $\ell_{\text{out}}(\mathbf{w}) = \ell_{\text{in}}(\mathbf{w}) \leq \ell_{\text{in}}(\mathsf{lastLeaf}_\sigma(\mathbf{v})) = \ell_{\text{out}}(\mathsf{lastLeaf}_\sigma(\mathbf{v}))$.

It follows that $\text{depth}_{\mathfrak{F}_{\text{out}}}(\mathbf{v}) = \ell_{\text{out}}(\mathsf{lastLeaf}_\sigma(\mathbf{v})) - \ell_{\text{out}}(\mathbf{v}) = \ell_{\text{in}}(\mathsf{lastLeaf}_\sigma(\mathbf{v})) - \ell_{\text{in}}(\mathbf{v}) = \text{depth}_{\mathfrak{F}_{\text{in}}}(\mathbf{v})$. □

Lemma A4 *Let* $\mathfrak{F}_{\text{in}} = (\mathscr{T}_{\text{in}}, \ell_{\text{in}})$ *be a* κ*-FCTS, let* $\lambda > 0$, *and let* $\mathfrak{F}_{\text{out}} = (\mathscr{T}_{\text{out}}, \ell_{\text{out}})$ *be the* κ*-FCTS that results from pruning* $\mathfrak{F}_{\text{in}}$ *by removing branches of length* $\leq \lambda$ *using an* ℓ_{in}*-increasing leaf enumeration* $\sigma = (\mathsf{leaf}[1], \ldots, \mathsf{leaf}[\mathsf{n}])$ *of* $\mathbf{Leaves}(\mathscr{T}_{\text{in}})$. *Then*:

(a) $\mathbf{Nodes}(\mathscr{T}_{\text{out}}) \setminus \mathbf{Leaves}(\mathscr{T}_{\text{out}}) \supseteq \mathbf{U}^\lambda\langle\mathfrak{F}_{\text{in}}\rangle \supseteq \mathbf{Crit}(\mathscr{T}_{\text{out}}) \setminus \mathbf{Leaves}(\mathscr{T}_{\text{out}})$
(b) *For all* $\mathbf{v} \in \mathbf{V}^\lambda\langle\mathfrak{F}_{\text{in}}\rangle \setminus \mathbf{V}_1^\lambda\langle\mathfrak{F}_{\text{in}}\rangle$, $\mathbf{v}\Uparrow_{\mathscr{T}_{\text{in}}} \cap \mathbf{Nodes}(\mathscr{T}_{\text{out}}) = \emptyset$.
(c) *For all* $\mathbf{v} \in \mathbf{V}_1^\lambda\langle\mathfrak{F}_{\text{in}}\rangle$, $\mathbf{v}\Uparrow_{\mathscr{T}_{\text{in}}} \cap \mathbf{Nodes}(\mathscr{T}_{\text{out}}) = \mathsf{Path}_\sigma(\mathbf{v}, \mathscr{T}_{\text{in}})$.

Proof For brevity, we shall write $\mathbf{U}^\lambda$, $\mathbf{V}^\lambda$, $\mathbf{V}_1^\lambda$, $\mathsf{lastLeaf}_\sigma(\mathbf{v})$, and $\mathsf{Path}_\sigma(\mathbf{v})$ for $\mathbf{U}^\lambda\langle\mathfrak{F}_{\mathrm{in}}\rangle$, $\mathbf{V}^\lambda\langle\mathfrak{F}_{\mathrm{in}}\rangle$, $\mathbf{V}_1^\lambda\langle\mathfrak{F}_{\mathrm{in}}\rangle$, $\mathsf{lastLeaf}_\sigma(\mathbf{v},\mathscr{T}_{\mathrm{in}})$, and $\mathsf{Path}_\sigma(\mathbf{v},\mathscr{T}_{\mathrm{in}})$.

First, we prove (a). The inclusion $\mathbf{U}^\lambda \supseteq \mathbf{Crit}(\mathscr{T}_{\mathrm{out}}) \setminus \mathbf{Leaves}(\mathscr{T}_{\mathrm{out}})$ follows from Corollary A2 and Lemma A3(c). Moreover, since P4 implies that $\mathbf{Leaves}(\mathscr{T}_{\mathrm{out}}) \subseteq \mathbf{Leaves}(\mathscr{T}_{\mathrm{in}})$, we have that $\mathbf{u} \notin \mathbf{Leaves}(\mathscr{T}_{\mathrm{out}})$ if $\mathbf{u} \in \mathbf{U}^\lambda$. So the other inclusion of (a) will follow if we can show that $\mathbf{u} \in \mathbf{Nodes}(\mathscr{T}_{\mathrm{out}})$ whenever $\mathbf{u} \in \mathbf{U}^\lambda$.

Let $\mathbf{u}$ be any element of $\mathbf{U}^\lambda$, and let $\mathsf{leaf}[i] = \mathsf{lastLeaf}_\sigma(\mathbf{u})$. If $i = \mathsf{n}$, then $\mathsf{lastLeaf}_\sigma(\mathbf{u}) \in \mathbf{Nodes}(\mathscr{T}_{\mathrm{out}})$ (by property P2) and so $\mathbf{u} \in \mathbf{Nodes}(\mathscr{T}_{\mathrm{out}})$ (because of P4), as required. Now suppose $i < \mathsf{n}$. Let j be any element of the set $\{i+1, \ldots, \mathsf{n}\}$ (so $\mathsf{leaf}[j] \notin \mathbf{Leaves}(\mathscr{T}_{\mathrm{in}}[\mathbf{u}])$). Since $\mathsf{leaf}[i]$ is a leaf of $\mathscr{T}_{\mathrm{in}}[\mathbf{u}]$ but $\mathsf{leaf}[j]$ is not, we have that $\bigwedge_{\mathscr{T}_{\mathrm{in}}}\{\mathsf{leaf}[j], \mathsf{leaf}[i]\} \prec_{\mathscr{T}_{\mathrm{in}}} \mathbf{u}$. Hence:

$$\begin{aligned}\ell_{\mathrm{in}}\big(\mathsf{leaf}[i]\big) - \ell_{\mathrm{in}}\Big(\bigwedge\nolimits_{\mathscr{T}_{\mathrm{in}}}\big\{\mathsf{leaf}[j], \mathsf{leaf}[i]\big\}\Big) &> \ell_{\mathrm{in}}\big(\mathsf{leaf}[i]\big) - \ell_{\mathrm{in}}(\mathbf{u}) \\ &= \mathrm{depth}_{\mathfrak{F}_{\mathrm{in}}}(\mathbf{u}) > \lambda\end{aligned}$$

We see from this and property P3 that $\mathsf{lastLeaf}_\sigma(\mathbf{u}) = \mathsf{leaf}[i] \in \mathbf{Leaves}(\mathscr{T}_{\mathrm{out}})$, and hence (in view of P4) that $\mathbf{u} \in \mathbf{Nodes}(\mathscr{T}_{\mathrm{out}})$, as required. This proves (a).

Next, we prove (b). Let $\mathbf{v}$ be any node in $\mathbf{V}^\lambda \setminus \mathbf{V}_1^\lambda$. Then it follows from the definitions of $\mathbf{V}^\lambda$ and $\mathbf{V}_1^\lambda$ that $\mathbf{v} \neq \mathbf{root}(\mathscr{T}_{\mathrm{in}})$.

Let $\mathbf{p} = \mathbf{parent}_{\mathscr{T}_{\mathrm{in}}}(\mathbf{v})$. Then $\mathbf{p} \in \mathbf{v}{\downarrow_{\mathscr{T}_{\mathrm{in}}}} \subseteq \mathbf{U}^\lambda$, so we have that:

$$\ell_{\mathrm{in}}\big(\mathsf{lastLeaf}_\sigma(\mathbf{p})\big) - \ell_{\mathrm{in}}(\mathbf{p}) = \mathrm{depth}_{\mathfrak{F}_{\mathrm{in}}}(\mathbf{p}) > \lambda \tag{A3}$$

Now $\ell_{\mathrm{in}}(\mathbf{d}) - \ell_{\mathrm{in}}(\mathbf{v}) \leq \mathrm{depth}_{\mathfrak{F}_{\mathrm{in}}}(\mathbf{v})$ for all $\mathbf{d} \succeq_{\mathscr{T}_{\mathrm{in}}} \mathbf{v}$. Therefore:

$$\ell_{\mathrm{in}}(\mathbf{d}) - \ell_{\mathrm{in}}(\mathbf{p}) \leq \mathrm{depth}_{\mathfrak{F}_{\mathrm{in}}}(\mathbf{v}) + \ell_{\mathrm{in}}(\mathbf{v}) - \ell_{\mathrm{in}}(\mathbf{p}) \leq \lambda \quad \text{for all } \mathbf{d} \succeq_{\mathscr{T}_{\mathrm{in}}} \mathbf{v} \tag{A4}$$

Here the second inequality follows from the definition of $\mathbf{V}_1^\lambda$ and the facts that $\mathbf{p} = \mathbf{parent}_{\mathscr{T}_{\mathrm{in}}}(\mathbf{v})$ and $\mathbf{v} \in \mathbf{V}^\lambda \setminus \mathbf{V}_1^\lambda$. It follows from (A3) and (A4) that $\mathsf{lastLeaf}_\sigma(\mathbf{p})$ is not a descendant of $\mathbf{v}$ in $\mathscr{T}_{\mathrm{in}}$, and so

$$\bigwedge\nolimits_{\mathscr{T}_{\mathrm{in}}}\big\{\mathsf{lastLeaf}_\sigma(\mathbf{p}), \mathsf{lastLeaf}_\sigma(\mathbf{v})\big\} = \mathbf{p} \tag{A5}$$

Since $\mathsf{lastLeaf}_\sigma(\mathbf{v}) \succeq_{\mathscr{T}_{\mathrm{in}}} \mathbf{v}$, we deduce from (A4) and (A5) that

$$\ell_{\mathrm{in}}\big(\mathsf{lastLeaf}_\sigma(\mathbf{v})\big) - \ell_{\mathrm{in}}\Big(\bigwedge\nolimits_{\mathscr{T}_{\mathrm{in}}}\big\{\mathsf{lastLeaf}_\sigma(\mathbf{p}), \mathsf{lastLeaf}_\sigma(\mathbf{v})\big\}\Big) \leq \lambda \tag{A6}$$

Since $\mathbf{p} = \mathbf{parent}_{\mathscr{T}_{\mathrm{in}}}(\mathbf{v})$ and $\mathsf{lastLeaf}_\sigma(\mathbf{p}) \neq \mathsf{lastLeaf}_\sigma(\mathbf{v})$ (e.g., by (A5)), the leaf $\mathsf{lastLeaf}_\sigma(\mathbf{p})$ must occur later in the ℓ_{in}-increasing enumeration σ than the leaf $\mathsf{lastLeaf}_\sigma(\mathbf{v})$. This, (A6), and P3 imply that $\mathsf{lastLeaf}_\sigma(\mathbf{v}) \notin \mathbf{Leaves}(\mathscr{T}_{\mathrm{out}})$. It now follows from assertion (b) of Lemma A3 that $\mathbf{v} \notin \mathbf{Nodes}(\mathscr{T}_{\mathrm{out}})$. This and assertion (a) of Lemma A3 imply $\mathbf{v}{\Uparrow_{\mathscr{T}_{\mathrm{in}}}} \cap \mathbf{Nodes}(\mathscr{T}_{\mathrm{out}}) = \emptyset$, which proves (b).

Finally, we prove (c). Let $\mathbf{v}$ be any node in $\mathbf{V}_1^\lambda$. We first make the claim that $\mathsf{lastLeaf}_\sigma(\mathbf{v})$ is a leaf of $\mathscr{T}_{\mathrm{out}}$.

If $\mathbf{v} = \mathbf{root}(\mathscr{T}_{\mathrm{in}})$ then the claim is certainly true (by property P2), so let us assume $\mathbf{v} \neq \mathbf{root}(\mathscr{T}_{\mathrm{in}})$. Let $\mathbf{p} = \mathbf{parent}_{\mathscr{T}_{\mathrm{in}}}(\mathbf{v})$, and let $\mathbf{s}$ be any leaf of $\mathscr{T}_{\mathrm{in}}$ that occurs later in the ℓ_{in}-increasing enumeration σ than $\mathsf{lastLeaf}_\sigma(\mathbf{v})$. Then $\mathbf{s} \notin \mathbf{Leaves}(\mathscr{T}_{\mathrm{in}}[\mathbf{v}])$, and so $\bigwedge_{\mathscr{T}_{\mathrm{in}}}\{\mathbf{s}, \mathsf{lastLeaf}_\sigma(\mathbf{v})\} \preceq_{\mathscr{T}_{\mathrm{in}}} \mathbf{p}$, which implies that:

$$\ell_{\text{in}}\big(\mathsf{lastLeaf}_\sigma(\mathbf{v})\big) - \ell_{\text{in}}\Big(\bigwedge\nolimits_{\mathscr{T}_{\text{in}}}\big\{\mathbf{s}, \mathsf{lastLeaf}_\sigma(\mathbf{v})\big\}\Big)$$
$$\geq \ell_{\text{in}}\big(\mathsf{lastLeaf}_\sigma(\mathbf{v})\big) - \ell_{\text{in}}(\mathbf{p}) \tag{A7}$$

But, since $\text{depth}_{\mathfrak{F}_{\text{in}}}(\mathbf{v}) = \ell_{\text{in}}(\mathsf{lastLeaf}_\sigma(\mathbf{v})) - \ell_{\text{in}}(\mathbf{v})$, we also have that

$$\ell_{\text{in}}\big(\mathsf{lastLeaf}_\sigma(\mathbf{v})\big) - \ell_{\text{in}}(\mathbf{p}) = \text{depth}_{\mathfrak{F}_{\text{in}}}(\mathbf{v}) + \ell_{\text{in}}(\mathbf{v}) - \ell_{\text{in}}(\mathbf{p}) > \lambda \tag{A8}$$

where the inequality follows from the definition of $\mathbf{V}_1^\lambda$ and the facts that $\mathbf{p} = \textbf{parent}_{\mathscr{T}_{\text{in}}}(\mathbf{v})$ and $\mathbf{v} \in \mathbf{V}_1^\lambda$. Now it follows from (A7) and (A8) that:

$$\ell_{\text{in}}\big(\mathsf{lastLeaf}_\sigma(\mathbf{v})\big) - \ell_{\text{in}}\Big(\bigwedge\nolimits_{\mathscr{T}_{\text{in}}}\big\{\mathbf{s}, \mathsf{lastLeaf}_\sigma(\mathbf{v})\big\}\Big) > \lambda$$

Since this is true for every leaf $\mathbf{s}$ of $\mathscr{T}_{\text{in}}$ that occurs later in the ℓ_{in}-increasing enumeration σ than $\mathsf{lastLeaf}_\sigma(\mathbf{v})$, our claim is justified (by property P3).

If $\mathbf{w}$ is any node in $\mathsf{Path}_\sigma(\mathbf{v})$, then $\mathbf{w} \in \mathsf{lastLeaf}_\sigma(\mathbf{v})\Downarrow_{\mathscr{T}_{\text{in}}}$ and so it follows from our claim (and P4) that $\mathbf{w} \in \textbf{Nodes}(\mathscr{T}_{\text{out}})$. Thus every node in $\mathsf{Path}_\sigma(\mathbf{v})$ lies in $\mathbf{v}\Uparrow_{\mathscr{T}_{\text{in}}} \cap \textbf{Nodes}(\mathscr{T}_{\text{out}})$.

It remains only to prove that $\mathbf{v}\Uparrow_{\mathscr{T}_{\text{in}}} \cap \textbf{Nodes}(\mathscr{T}_{\text{out}}) \setminus \mathsf{Path}_\sigma(\mathbf{v}) = \emptyset$. To do this, we suppose there is a node $\mathbf{x} \in \mathbf{v}\Uparrow_{\mathscr{T}_{\text{in}}} \cap \textbf{Nodes}(\mathscr{T}_{\text{out}}) \setminus \mathsf{Path}_\sigma(\mathbf{v})$ and deduce a contradiction. As $\mathbf{x} \in \mathbf{v}\Uparrow_{\mathscr{T}_{\text{in}}} \setminus \mathsf{Path}_\sigma(\mathbf{v})$, we have that $\mathbf{x} \notin \mathsf{lastLeaf}_\sigma(\mathbf{v})\Downarrow_{\mathscr{T}_{\text{in}}}$ and so $\mathsf{lastLeaf}_\sigma(\mathbf{v}) \neq \mathsf{lastLeaf}_\sigma(\mathbf{x})$. Moreover, each of the nodes $\mathsf{lastLeaf}_\sigma(\mathbf{x})$ and $\mathsf{lastLeaf}_\sigma(\mathbf{v})$ is a leaf of $\mathscr{T}_{\text{out}}$ (by Lemma A3(b) and our claim).

Let $\mathbf{c} = \bigwedge_{\mathscr{T}_{\text{out}}}\{\mathsf{lastLeaf}_\sigma(\mathbf{x}), \mathsf{lastLeaf}_\sigma(\mathbf{v})\}$. Then we have that $\mathbf{c} \in \textbf{Crit}(\mathscr{T}_{\text{out}})$, $c \notin \textbf{Leaves}(\mathscr{T}_{\text{out}})$, and $\mathbf{c} = \bigwedge_{\mathscr{T}_{\text{in}}}\{\mathsf{lastLeaf}_\sigma(\mathbf{x}), \mathsf{lastLeaf}_\sigma(\mathbf{v})\}$ (by assertion (i) of Lemma A1). The latter implies $\mathbf{c} \succeq_{\mathscr{T}_{\text{in}}} \mathbf{v}$ (as $\mathsf{lastLeaf}_\sigma(\mathbf{x}) \succeq_{\mathscr{T}_{\text{in}}} \mathbf{x} \succeq_{\mathscr{T}_{\text{in}}} \mathbf{v}$ and $\mathsf{lastLeaf}_\sigma(\mathbf{v}) \succeq_{\mathscr{T}_{\text{in}}} \mathbf{v}$); and $\mathbf{c} \succeq_{\mathscr{T}_{\text{in}}} \mathbf{v}$ implies $\text{depth}_{\mathfrak{F}_{\text{in}}}\mathbf{c} \leq \text{depth}_{\mathfrak{F}_{\text{in}}}\mathbf{v} \leq \lambda$ (where the second inequality follows from the fact that $\mathbf{v} \in \mathbf{V}_1^\lambda \subseteq \mathbf{V}^\lambda$). Hence $\mathbf{c} \notin \mathbf{U}^\lambda$. But this contradicts assertion (a) (since $\mathbf{c} \in \textbf{Crit}(\mathscr{T}_{\text{out}}) \setminus \textbf{Leaves}(\mathscr{T}_{\text{out}})$). It follows that $\mathbf{x}$ cannot exist, and so our proof of (c) is complete. □

We can now prove the main result of Sect. 2.4.2:

Proposition *Let $\mathfrak{F}_{\text{in}} = (\mathscr{T}_{\text{in}}, \ell_{\text{in}})$ be a κ-FCTS, let $\lambda > 0$, and let $\mathfrak{F}_{\text{out}} = (\mathscr{T}_{\text{out}}, \ell_{\text{out}})$ be the κ-FCTS that results from pruning $\mathfrak{F}_{\text{in}}$ by removing branches of length $\leq \lambda$ using an ℓ_{in}-increasing enumeration σ of $\textbf{Leaves}(\mathscr{T}_{\text{in}})$. Then the nodes of $\mathfrak{F}_{\text{out}}$ consist just of*:

(i) *The nodes of $\mathbf{U}^\lambda\langle\mathfrak{F}_{\text{in}}\rangle$.*
(ii) *The nodes of $\mathsf{Path}_\sigma(\mathbf{v}, \mathscr{T}_{\text{in}})$ for each node $\mathbf{v}$ in $\mathbf{V}_1^\lambda\langle\mathfrak{F}_{\text{in}}\rangle$.*

Proof As $\mathbf{U}^\lambda\langle\mathfrak{F}_{\text{in}}\rangle \subseteq \textbf{Nodes}(\mathscr{T}_{\text{out}})$ by Lemma A4(a), on putting $\mathscr{T} = \mathscr{T}_{\text{in}}$ and $\mathfrak{F} = \mathfrak{F}_{\text{in}}$ in (2.1) and taking the intersection of each side with $\textbf{Nodes}(\mathscr{T}_{\text{out}})$ we see that:

$$\textbf{Nodes}(\mathscr{T}_{\text{out}}) = \mathbf{U}^\lambda\langle\mathfrak{F}_{\text{in}}\rangle \cup \bigcup_{\mathbf{v}\in\mathbf{V}^\lambda\langle\mathfrak{F}_{\text{in}}\rangle} \big(\mathbf{v}\Uparrow_{\mathscr{T}_{\text{in}}} \cap \textbf{Nodes}(\mathscr{T}_{\text{out}})\big)$$

The proposition follows from this and assertions (b) and (c) of Lemma A4. □

A.2 Properties of Simplification Step 3

Here we establish some properties of simplification step 3 that are used in our proof of the Main Theorem and our justification of Algorithm 1.

For all $j \in \{1, \ldots, |D(\mathfrak{F})|\}$, we see from E1–E5 that $\mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\delta\rangle) \subseteq \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\delta'\rangle)$ whenever $\delta \geq \delta'$. It follows that $\mathfrak{F}^{\mathbf{crit}}\langle\cdot\rangle$ has the following monotonicity property:

$$\mathfrak{F}^{\mathbf{crit}}\langle\delta\rangle \sqsubseteq \mathfrak{F}^{\mathbf{crit}}\langle\delta'\rangle \quad \text{whenever } \delta \geq \delta' \tag{A9}$$

In addition, $\mathfrak{F}^{\mathbf{crit}}\langle\cdot\rangle$ has the following four properties for every $\lambda > 0$ (as we will explain below):

E6: For every $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}) \setminus (\mathbf{Leaves}(\mathfrak{F}) \cup \{\mathbf{LCN}(\mathfrak{F})\} \cup \{\mathbf{root}(\mathfrak{F})\})$ and every $i \in \{0, \ldots, |D(\mathfrak{F})| - 1\}$, $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle d_{i+1}^{\mathfrak{F}}\rangle)$ if and only if, for every $j \in \{0, \ldots, i\}$, $\ell(\mathbf{c}) - \ell(\mathbf{parent}_{\mathfrak{F}^{\mathbf{crit}}\langle d_j^{\mathfrak{F}}\rangle}(\mathbf{c})) > d_{j+1}^{\mathfrak{F}}$.

E7: For every $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}) \setminus (\mathbf{Leaves}(\mathfrak{F}) \cup \{\mathbf{LCN}(\mathfrak{F})\} \cup \{\mathbf{root}(\mathfrak{F})\})$, $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle)$ if and only if there is no critical proper ancestor $\mathbf{c}'$ of $\mathbf{c}$ in $\mathfrak{F}$ such that $\ell(\mathbf{c}) - \ell(\mathbf{c}') \leq \lambda$ and $\mathbf{c}' \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\mathrm{pred}_{\mathfrak{F}}(\ell(\mathbf{c}) - \ell(\mathbf{c}'))\rangle)$.

E8: For every $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}) \setminus (\mathbf{Leaves}(\mathfrak{F}) \cup \{\mathbf{LCN}(\mathfrak{F})\} \cup \{\mathbf{root}(\mathfrak{F})\})$, $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle)$ if $\ell(\mathbf{c}) - \ell(\mathbf{parent}_{\mathfrak{F}^{\mathbf{crit}}}(\mathbf{c})) > \lambda$.

E9: For every $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}) \setminus (\mathbf{Leaves}(\mathfrak{F}) \cup \{\mathbf{LCN}(\mathfrak{F})\} \cup \{\mathbf{root}(\mathfrak{F})\})$, if $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle)$ then $\ell(\mathbf{c}) - \ell(\mathbf{parent}_{\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle}(\mathbf{c})) > \lambda$.

Our proof of the correctness of Algorithm 1 will be based on property E7. However, E1–E3, E8, and E9 are the only properties of simplification step 3 that will be used in our proof of the Main Theorem.

E6 is easily deduced from E5 by induction on i. Now we establish E7–E9. Let $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}) \setminus (\mathbf{Leaves}(\mathfrak{F}) \cup \{\mathbf{LCN}(\mathfrak{F})\} \cup \{\mathbf{root}(\mathfrak{F})\})$, and let λ be any positive value. We first claim that, for any critical proper ancestor $\mathbf{c}'$ of $\mathbf{c}$ in $\mathfrak{F}$, the following four conditions are equivalent:

(a) There is some $j \in \{0, \ldots, |D(\mathfrak{F})| - 1\}$ such that $\ell(\mathbf{c}) - \ell(\mathbf{c}') \leq d_{j+1}^{\mathfrak{F}} \leq \lambda$ and $\mathbf{c}' \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle d_j^{\mathfrak{F}}\rangle)$.

(b) There is some $j \in \{0, \ldots, |D(\mathfrak{F})| - 1\}$ such that $\ell(\mathbf{c}) - \ell(\mathbf{c}') \leq d_{j+1}^{\mathfrak{F}} \leq \lambda$ and $\mathbf{c}' \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\mathrm{pred}_{\mathfrak{F}}(\ell(\mathbf{c}) - \ell(\mathbf{c}'))\rangle)$.

(c) $\ell(\mathbf{c}) - \ell(\mathbf{c}') \leq \lambda$ and $\mathbf{c}' \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\mathrm{pred}_{\mathfrak{F}}(\ell(\mathbf{c}) - \ell(\mathbf{c}'))\rangle)$.

(d) There is some $j \in \{0, \ldots, |D(\mathfrak{F})| - 1\}$ such that $\ell(\mathbf{c}) - \ell(\mathbf{c}') = d_{j+1}^{\mathfrak{F}} \leq \lambda$ and $\mathbf{c}' \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle d_j^{\mathfrak{F}}\rangle)$.

Here (a) implies (b) because of the monotonicity property (A9) and the fact that if $\ell(\mathbf{c}) - \ell(\mathbf{c}') \leq d_{j+1}^{\mathfrak{F}}$ then $\mathrm{pred}_{\mathfrak{F}}(\ell(\mathbf{c}) - \ell(\mathbf{c}')) \leq d_j^{\mathfrak{F}}$. Evidently, (b) implies (c), and (d) implies (a). For any critical proper ancestor $\mathbf{c}'$ of $\mathbf{c}$ in $\mathfrak{F}$, $\ell(\mathbf{c}) - \ell(\mathbf{c}') = d_{j+1}^{\mathfrak{F}}$ and $\mathrm{pred}_{\mathfrak{F}}(\ell(\mathbf{c}) - \ell(\mathbf{c}')) = d_j^{\mathfrak{F}}$ for some $j \in \{0, \ldots, |D(\mathfrak{F})| - 1\}$, and so (c) implies (d). This justifies our claim that (a)–(d) are equivalent.

Next, we observe that $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle)$ holds if and only if $\mathbf{c}$ satisfies $\ell(\mathbf{c}) - \ell(\mathbf{parent}_{\mathfrak{F}^{\mathbf{crit}}\langle d_j^{\mathfrak{F}}\rangle}(\mathbf{c})) > d_{j+1}^{\mathfrak{F}}$ for all $j \in \{0, \ldots, |D(\mathfrak{F})| - 1\}$ such that $d_{j+1}^{\mathfrak{F}} \le \lambda$. (This follows from E6 when $\lambda \in D(\mathfrak{F})$. It remains true if $\lambda \notin D(\mathfrak{F})$, because of E4.) So $\mathbf{c} \notin \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle)$ just if there is some $j \in \{0, \ldots, |D(\mathfrak{F})| - 1\}$ such that $\ell(\mathbf{c}) - \ell(\mathbf{parent}_{\mathfrak{F}^{\mathbf{crit}}\langle d_j^{\mathfrak{F}}\rangle}(\mathbf{c})) \le d_{j+1}^{\mathfrak{F}} \le \lambda$. Thus $\mathbf{c} \notin \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle)$ just if (a) holds for some critical proper ancestor $\mathbf{c}'$ of $\mathbf{c}$ in $\mathfrak{F}$. Equivalently, $\mathbf{c} \notin \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle)$ just if (c) holds for some critical proper ancestor $\mathbf{c}'$ of $\mathbf{c}$ in $\mathfrak{F}$. This proves E7. E8 follows from the "if" part of E7.

Suppose the node $\mathbf{c}$ violated E9. Then $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle)$. Moreover, when $\mathbf{c}' = \mathbf{parent}_{\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle}(\mathbf{c})$ we would have that $\ell(\mathbf{c}) - \ell(\mathbf{c}') \le \lambda$ and also that $\mathbf{c}' \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle \mathrm{pred}_{\mathfrak{F}}(\ell(\mathbf{c}) - \ell(\mathbf{c}'))\rangle)$, where the latter follows from the former, the fact that $\mathbf{c}' \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\lambda\rangle)$, and the monotonicity property (A9). But this would contradict the "only if" part of E7. So E9 holds.

A.3 Justification of Algorithm 1

The correctness of Algorithm 1 will be deduced from Lemma A5 and Corollary A6 below.

Let $\mathfrak{F} = (\mathscr{T}, \ell)$ be any κ-FCTS, and let $\mathbf{c}$ be any node of $\mathfrak{F}^{\mathbf{crit}}$. Then we define $\delta_\lambda(\mathbf{c}, \mathfrak{F}) = \infty$ if $\mathbf{c} \in \mathbf{Leaves}(\mathfrak{F}) \cup \{\mathbf{LCN}(\mathfrak{F})\} \cup \{\mathbf{root}(\mathfrak{F})\}$, and we define $\delta_\lambda(\mathbf{c}, \mathfrak{F}) = \ell(\mathbf{c}) - \ell(\mathbf{a}_\lambda(\mathbf{c}, \mathfrak{F}))$ otherwise, where $\mathbf{a}_\lambda(\mathbf{c}, \mathfrak{F})$ is the closest critical proper ancestor $\mathbf{c}'$ of $\mathbf{c}$ in $\mathfrak{F}$ such that

$$\begin{array}{ll} \textbf{either} & \ell(\mathbf{c}) - \ell(\mathbf{c}') > \lambda \\ \textbf{or} & \ell(\mathbf{c}) - \ell\big(\mathbf{c}'\big) \le \lambda \text{ and } \mathbf{c}' \in \mathbf{Nodes}\big(\mathfrak{F}^{\mathbf{crit}}\langle \mathrm{pred}_{\mathfrak{F}}\big(\ell(\mathbf{c}) - \ell\big(\mathbf{c}'\big)\big)\big\rangle\big) \end{array}$$

$\mathbf{a}_\lambda(\mathbf{c}, \mathfrak{F})$ exists for all $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}) \setminus (\mathbf{Leaves}(\mathfrak{F}) \cup \{\mathbf{LCN}(\mathfrak{F})\} \cup \{\mathbf{root}(\mathfrak{F})\})$, because when $\mathbf{c}' = \mathbf{LCN}(\mathfrak{F})$ we see from E2 that $\mathbf{c}' \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\mu\rangle)$ for every $\mu \ge 0$ and so $\mathbf{c}'$ must satisfy the "**either**" or the "**or**" condition. Now $\delta_\lambda(\cdot, \mathfrak{F})$ satisfies the following condition:

Lemma A5 *Let $0 \le \mu \le \lambda$ and let $\mathfrak{F} = (\mathscr{T}, \ell)$ be any κ-FCTS. Then for all $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}})$ we have that $\delta_\lambda(\mathbf{c}, \mathfrak{F}) > \mu$ if and only if $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\mu\rangle)$.*

Proof Suppose $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}) \setminus (\mathbf{Leaves}(\mathfrak{F}) \cup \{\mathbf{LCN}(\mathfrak{F})\} \cup \{\mathbf{root}(\mathfrak{F})\})$. Then $\delta_\lambda(\mathbf{c}, \mathfrak{F}) > \mu$ holds just if $\ell(\mathbf{c}) - \ell(\mathbf{a}_\lambda(\mathbf{c}, \mathfrak{F})) > \mu$, and since $\mu \le \lambda$ we see from the definition of $\mathbf{a}_\lambda(\mathbf{c}, \mathfrak{F})$ that this holds just if no critical proper ancestor $\mathbf{c}'$ of $\mathbf{c}$ in $\mathfrak{F}$ satisfies $\ell(\mathbf{c}) - \ell(\mathbf{c}') \le \mu$ and $\mathbf{c}' \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle \mathrm{pred}_{\mathfrak{F}}(\ell(\mathbf{c}) - \ell(\mathbf{c}'))\rangle)$. So in this case the lemma follows from E7.

The lemma also holds if $\mathbf{c} \in \mathbf{Leaves}(\mathfrak{F}) \cup \{\mathbf{LCN}(\mathfrak{F})\} \cup \{\mathbf{root}(\mathfrak{F})\}$, because in that case $\delta_\lambda(\mathbf{c}, \mathfrak{F}) = \infty > \mu$ and E1–E3 imply $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle\mu\rangle)$. □

Corollary A6 *Let λ be any positive value, let $\mathfrak{F} = (\mathscr{T}, \ell)$ be any κ-FCTS, and let $\mathbf{c} \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}) \setminus (\mathbf{Leaves}(\mathfrak{F}) \cup \{\mathbf{LCN}(\mathfrak{F})\} \cup \{\mathbf{root}(\mathfrak{F})\})$. Then $\delta_\lambda(\mathbf{c}, \mathfrak{F}) = \ell(\mathbf{c}) - \ell(\mathbf{a})$, where $\mathbf{a}$ is the closest critical proper ancestor $\mathbf{c}'$ of $\mathbf{c}$ in $\mathfrak{F}$ such that*

$$\begin{array}{ll} \textbf{\textit{either}} & \ell(\mathbf{c}) - \ell\big(\mathbf{c}'\big) > \lambda \\ \textbf{\textit{or}} & \ell(\mathbf{c}) - \ell\big(\mathbf{c}'\big) \le \lambda \text{ and } \ell(\mathbf{c}) - \ell\big(\mathbf{c}'\big) \le \delta_\lambda\big(\mathbf{c}', \mathfrak{F}\big) \end{array}$$

Proof We just have to show that $\mathbf{a} = \mathbf{a}_\lambda(\mathbf{c}, \mathfrak{F})$. The definition of $\mathbf{a}_\lambda(\mathbf{c}, \mathfrak{F})$ differs from the definition of $\mathbf{a}$ only in the **or** condition "$\ell(\mathbf{c}) - \ell(\mathbf{c}') \le \lambda$ and $\mathbf{c}' \in \mathbf{Nodes}(\mathfrak{F}^{\mathbf{crit}}\langle \mathrm{pred}_{\mathfrak{F}}(\ell(\mathbf{c}) - \ell(\mathbf{c}'))\rangle)$".

On putting $\mu = \mathrm{pred}_{\mathfrak{F}}(\ell(\mathbf{c}) - \ell(\mathbf{c}'))$ in Lemma A5, we see that this condition holds if and only if $\ell(\mathbf{c}) - \ell(\mathbf{c}') \le \lambda$ and $\mathrm{pred}_{\mathfrak{F}}(\ell(\mathbf{c}) - \ell(\mathbf{c}')) < \delta_\lambda(\mathbf{c}', \mathfrak{F})$, which is equivalent to the **or** condition in the definition of $\mathbf{a}$ (because either $\delta_\lambda(\mathbf{c}', \mathfrak{F}) = \ell(\mathbf{c}) - \ell(\mathbf{a}_\lambda(\mathbf{c}', \mathfrak{F})) \in D(\mathfrak{F})$ or $\delta_\lambda(\mathbf{c}', \mathfrak{F}) = \infty$). So $\mathbf{a} = \mathbf{a}_\lambda(\mathbf{c}, \mathfrak{F})$, as required. □

We can now explain why Algorithm 1 is correct. The algorithm sets $(\mathscr{T}, \ell)$ to a clone of $\mathfrak{F}^{\mathbf{crit}}_{\mathrm{in}} = (\mathscr{T}^{\mathbf{crit}}_{\mathrm{in}}, \ell^{\mathbf{crit}}_{\mathrm{in}})$. Writing $\mathfrak{F}$ for $(\mathscr{T}, \ell)$, we claim that the label $\mathbf{c}$.label given by the algorithm to each node $\mathbf{c}$ of $\mathfrak{F} = \mathfrak{F}^{\mathbf{crit}}$ is just the value $\delta_\lambda(\mathbf{c}, \mathfrak{F})$. Assuming this claim is valid, the correctness of the algorithm follows from Lemma A5. So it remains only to verify the claim.

The claim is certainly valid if $\mathbf{c}$ is $\mathbf{root}(\mathfrak{F})$ or $\mathbf{LCN}(\mathfrak{F})$, because those nodes are given the label ∞.

We see that the algorithm does a top-down traversal of $\mathscr{T}[\mathbf{LCN}(\mathfrak{F})]$, during which the procedure `labelDescendants` is executed once for each proper descendant $\mathbf{c}$ of $\mathbf{LCN}(\mathfrak{F})$ in $\mathfrak{F}$. When `labelDescendants` is executed for such a node $\mathbf{c}$ that is a leaf, it gives $\mathbf{c}$ the label ∞. So the claim is valid for each proper descendant $\mathbf{c}$ of $\mathbf{LCN}(\mathfrak{F})$ that is a leaf.

When `labelDescendants` is executed for a proper descendant $\mathbf{c}$ of $\mathbf{LCN}(\mathfrak{F})$ that is not a leaf, the **repeat** loop in the procedure is executed. It follows from Corollary A6 that this loop labels $\mathbf{c}$ with the value $\delta_\lambda(\mathbf{c}, \mathfrak{F})$. (Note that, when the loop is executed, $\mathbf{c}'$.label $= \delta_\lambda(\mathbf{c}', \mathfrak{F})$ for each proper ancestor $\mathbf{c}'$ of $\mathbf{c}$ in $\mathfrak{F}$.) Therefore the claim is also valid for each proper descendant $\mathbf{c}$ of $\mathbf{LCN}(\mathfrak{F})$ that is not a leaf.

Thus the claim is valid for all nodes $\mathbf{c}$ of $\mathfrak{F} = \mathfrak{F}^{\mathbf{crit}}$, and Algorithm 1 is correct.

Appendix B: A Constructive Proof of Theorem 1

For any adjacency relation κ, any image I whose domain is finite and κ-connected, any $\lambda > 0$, and any integer $k \ge 0$, let us say that the image I is (λ, k)-*good* with respect to κ if $\Lambda_\kappa(I) > \lambda$ and $K_\kappa(I) > k$. Also, let us say that an image I' is an ε-*perturbation* of an image I if I' has the same domain as I and $\|I' - I\|_\infty \le \varepsilon$. Then Theorem 1 can be deduced from the following lemma:

Fundamental Lemma *Let κ be any adjacency relation and $I_{\mathbf{good}} : \mathcal{S} \to \mathbb{R}$ an image whose domain $\mathcal{S}$ is finite and κ-connected. Let ε be a positive value, let k be a nonnegative integer for which $I_{\mathbf{good}}$ is $(4\varepsilon, k)$-good with respect to κ, and let I' be an ε-perturbation of $I_{\mathbf{good}}$. Then there is an essential isomorphism of $\mathbf{FCTS}_\kappa(I_{\mathbf{good}})$ to the $(2\varepsilon, k)$-simplification of $\mathbf{FCTS}_\kappa(I')$ that is level-preserving to within ε.*

Proof of Theorem 1, assuming the Fundamental Lemma is valid Suppose I, λ, and k satisfy the hypotheses of Theorem 1, so that $0 < \lambda < \Lambda_\kappa(I)/2$ and $0 \le k < K_\kappa(I)$. Let I' be any image that satisfies the conditions stated in the theorem (i.e., let I' be any image whose domain is the same as that of I and which satisfies the condition $\|I' - I\|_\infty \le \lambda/2$). Then we need to show that the conclusion of Theorem 1 holds—i.e., that there is an essential isomorphism of the (λ, k)-simplification of $\mathbf{FCTS}_\kappa(I')$ to $\mathbf{FCTS}_\kappa(I)$ that is level-preserving to within $\lambda/2$. We now deduce this from the Fundamental Lemma.

Let $I_{\mathbf{good}} = I$, and let $\varepsilon = \lambda/2$. Then $4\varepsilon = 2\lambda < \Lambda_\kappa(I) = \Lambda_\kappa(I_{\mathbf{good}})$ and $k < K_\kappa(I) = K_\kappa(I_{\mathbf{good}})$, so that $I_{\mathbf{good}}$ is $(4\varepsilon, k)$-good with respect to κ. We also have that $\|I' - I_{\mathbf{good}}\|_\infty = \|I' - I\|_\infty \le \lambda/2 = \varepsilon$, so that I' is a ε-perturbation of $I_{\mathbf{good}}$. Thus $I_{\mathbf{good}} = I$ and I' satisfy the hypotheses of the Fundamental Lemma, and must therefore satisfy the conclusion of the lemma, which implies the conclusion of Theorem 1 since $2\varepsilon = \lambda$. □

We now prove the Fundamental Lemma by constructing an explicit essential isomorphism of $\mathbf{FCTS}_\kappa(I_{\mathbf{good}})$ to the $(2\varepsilon, k)$-simplification of $\mathbf{FCTS}_\kappa(I')$ that is level-preserving to within ε.

Let $\mathfrak{F}_{\mathbf{good}} = (\mathcal{T}_{\mathbf{good}}, \ell_{\mathbf{good}}) = \mathbf{FCTS}_\kappa(I_{\mathbf{good}})$, and let $\mathfrak{F}' = (\mathcal{T}', \ell') = \mathbf{FCTS}_\kappa(I')$. Let $\mathfrak{F}_1 = (\mathcal{T}_1, \ell_1)$ be the κ-FCTS that results from pruning $\mathfrak{F}'$ by removing nodes of size $\le k$, and let I_1 be the image $I_{\mathfrak{F}_1}$, so that $\mathfrak{F}_1 = \mathbf{FCTS}_\kappa(I_1)$. Let $\mathfrak{F}_2 = (\mathcal{T}_2, \ell_2)$ be the κ-FCTS that results from pruning $\mathfrak{F}_1$ by removing branches of length $\le 2\varepsilon$, and let $\mathfrak{F}_3 = (\mathcal{T}_3, \ell_3)$ be the κ-FCTS that results from eliminating internal edges of length $\le 2\varepsilon$ from $\mathfrak{F}_2^{\mathbf{crit}}$. Then $\mathfrak{F}_3 = (\mathcal{T}_3, \ell_3)$ is the $(2\varepsilon, k)$-simplification of $\mathbf{FCTS}_\kappa(I')$, so what we want to do is to construct an essential isomorphism of $\mathfrak{F}_{\mathbf{good}}$ to $\mathfrak{F}_3$ that is level-preserving to within ε. We will do this in three steps:

Step 1: We define a suitable mapping $\phi : \mathbf{Leaves}(\mathcal{T}_{\mathbf{good}}) \to \mathbf{Leaves}(\mathcal{T}_1)$.

Step 2: We show that ϕ is 1-to-1, and that the range of the mapping ϕ is exactly the set of all the leaves of the subtree $\mathcal{T}_2$ of $\mathcal{T}_1$. Thereafter, we regard ϕ as a bijection $\phi : \mathbf{Leaves}(\mathcal{T}_{\mathbf{good}}) \to \mathbf{Leaves}(\mathcal{T}_2)$.

Step 3: We extend ϕ to a mapping $\varphi : \mathbf{Crit}(\mathcal{T}_{\mathbf{good}}) \to \mathbf{Crit}(\mathcal{T}_2)$ by defining $\varphi(\mathbf{u}) = \bigwedge_{\mathcal{T}_2} \phi[\mathbf{Leaves}(\mathcal{T}_{\mathbf{good}}[\mathbf{u}])]$. We then establish that, for all $\mathbf{u}, \mathbf{u}' \in \mathbf{Crit}(\mathcal{T}_{\mathbf{good}})$, $\varphi(\mathbf{u}) \preceq_{\mathcal{T}_2} \varphi(\mathbf{u}')$ if and only if $\mathbf{u} \preceq_{\mathcal{T}_{\mathbf{good}}} \mathbf{u}'$, so that φ is 1-to-1 and order-preserving. We also show that the range of φ is the subset $\mathbf{Crit}(\mathcal{T}_3)$ of $\mathbf{Crit}(\mathcal{T}_2)$, and that $|\ell_3(\varphi(\mathbf{u})) - \ell_{\mathbf{good}}(\mathbf{u})| \le \varepsilon$ for every $\mathbf{u} \in \mathbf{Crit}(\mathcal{T}_{\mathbf{good}})$. Hence we can regard φ as a mapping $\varphi : \mathbf{Crit}(\mathcal{T}_{\mathbf{good}}) \to \mathbf{Crit}(\mathcal{T}_3)$ and, when so regarded, φ is an essential isomorphism of $\mathfrak{F}_{\mathbf{good}}$ to $\mathfrak{F}_3$ that is level-preserving to within ε.

Note that the extension of ϕ to φ in step 3 is very natural because, if $\mathscr{T}$ is any rooted tree and $\mathbf{u} \in \mathbf{Crit}(\mathscr{T})$, then $\mathbf{u} = \bigwedge_{\mathscr{T}} \mathbf{Leaves}(\mathscr{T}[\mathbf{u}])$. (In fact $\mathbf{u} \in \mathbf{Crit}(\mathscr{T})$ if and only if $\mathbf{u} \in \mathbf{Nodes}(\mathscr{T})$ and $\mathbf{u} = \bigwedge_{\mathscr{T}} \mathbf{Leaves}(\mathscr{T}[\mathbf{u}])$.)

B.1 Step 1 of the Proof of the Fundamental Lemma

We begin by defining a class of symmetric and transitive relations (on spels) that will be used in our definition of the mapping ϕ.

If $I : \mathcal{S} \to \mathbb{R}$ is an image and $\tau \in \mathbb{R}$, then we write $s \Leftrightarrow_{I \geq \tau} t$ to mean that $s, t \in \mathcal{S}$ and $t \in \mathcal{C}_\kappa(s, I, \tau)$. It is readily confirmed that $\Leftrightarrow_{I \geq \tau}$ is a symmetric and transitive relation (which depends on κ), and that $s \Leftrightarrow_{I \geq \tau} s$ if and only if $I(s) \geq \tau$. Moreover, if $s \Leftrightarrow_{I \geq \tau_1} t$ and $t \Leftrightarrow_{I \geq \tau_2} u$ then $s \Leftrightarrow_{I \geq \min(\tau_1, \tau_2)} u$.

Now let $\mathcal{C}_\kappa(v, I_{\mathbf{good}})$ be any leaf of $\mathscr{T}_{\mathbf{good}}$, and let z be any spel such that

$$z \in \operatorname*{arg\,min}_{u \Leftrightarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v) - 2\varepsilon} v} I_1(u) \tag{B1}$$

It follows from (B1) that:

$$\mathcal{C}_\kappa(z, I_1) \supseteq \left\{ u \;\middle|\; u \Leftrightarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v) - 2\varepsilon} v \right\} = \mathcal{C}_\kappa\left(v, I_{\mathbf{good}}, I_{\mathbf{good}}(v) - 2\varepsilon\right) \tag{B2}$$

Next, we define:

$$\mathcal{M}\left(\mathcal{C}_\kappa(v, I_{\mathbf{good}})\right) = \mathbf{Leaves}\left(\mathscr{T}_1\left[\mathcal{C}_\kappa(z, I_1)\right]\right) \tag{B3}$$

The set $\mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$ is well defined by (B3) for the following reasons. First, if v' is any spel such that $\mathcal{C}_\kappa(v', I_{\mathbf{good}}) = \mathcal{C}_\kappa(v, I_{\mathbf{good}})$ (so that $I_{\mathbf{good}}(v') = I_{\mathbf{good}}(v)$) then the condition obtained from (B1) when we replace v with v' is equivalent to (B1). Second, if z' is any spel that belongs to the set in (B1), then $\mathcal{C}_\kappa(z', I_1) = \mathcal{C}_\kappa(z, I_1)$ (since $I_1(z') = I_1(z)$, and (B2) implies $z' \in \mathcal{C}_\kappa(z, I_1)$).

We can now define the mapping $\phi : \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}}) \to \mathbf{Leaves}(\mathscr{T}_1)$ by defining $\phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$ to be the element of $\mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$ that occurs later in the ℓ_1-increasing leaf enumeration that is used in pruning $(\mathscr{T}_1, \ell_1)$ (to produce $(\mathscr{T}_2, \ell_2)$) than all other elements of $\mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$. Note that if $\mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$ has just one element, then $\phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$ is that element.

This completes step 1 of the proof of the Fundamental Lemma.

B.2 Some Useful Observations

Steps 2 and 3 of the proof of the Fundamental Lemma will be based on the following observations:

A. If $(\mathscr{T}, \ell) = \mathbf{FCTS}_\kappa(I)$, where I is an arbitrary image whose domain is finite and κ-connected, and $\emptyset \neq \mathbf{S} \subseteq \mathbf{Nodes}(\mathscr{T})$, then $\ell(\bigwedge_{\mathscr{T}} \mathbf{S})$ is the greatest real value τ such that $s \Leftrightarrow_{I \geq \tau} t$ for all spels $s, t \in \bigcup \mathbf{S}$.

B. Whenever $\emptyset \neq \mathbf{L} \subsetneq \mathbf{L}' \subseteq \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$ and $\bigwedge_{\mathscr{T}_{\mathbf{good}}} \mathbf{L}' \neq \bigwedge_{\mathscr{T}_{\mathbf{good}}} \mathbf{L}$, we have that $\ell_{\mathbf{good}}(\bigwedge_{\mathscr{T}_{\mathbf{good}}} \mathbf{L}') < \ell_{\mathbf{good}}(\bigwedge_{\mathscr{T}_{\mathbf{good}}} \mathbf{L}) - 4\varepsilon$.

C. If $v \in \mathbf{v} \in \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$, $\mathbf{u} \in \mathbf{Nodes}(\mathscr{T}_{\mathbf{good}})$, and $\mathbf{v} \not\succeq_{\mathscr{T}_{\mathbf{good}}} \mathbf{u}$, then we have that $\ell_{\mathbf{good}}(\bigwedge_{\mathscr{T}_{\mathbf{good}}} \{\mathbf{u}, \mathbf{v}\}) < \ell_{\mathbf{good}}(\mathbf{v}) - 4\varepsilon = I_{\mathbf{good}}(v) - 4\varepsilon$.

D. If $\mathcal{C}_\kappa(v, I_{\mathbf{good}}) \in \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$ and $u \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v) - 4\varepsilon} \Rightarrow v$, then we have that $u \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(u)} \Rightarrow v$ or, equivalently, $\mathcal{C}_\kappa(u, I_{\mathbf{good}}) \supseteq \mathcal{C}_\kappa(v, I_{\mathbf{good}})$.

E. If $\mathcal{C}_\kappa(x, I_1) \in \mathbf{Leaves}(\mathscr{T}_1)$, then $\mathcal{C}_\kappa(x, I_1) \in \mathbf{Leaves}(\mathscr{T}_2)$ if and only if there is no node $\mathcal{C}_\kappa(y, I_1) \in \mathbf{Leaves}(\mathscr{T}_1)$ that satisfies both of the following conditions:
   (i) $x \Leftarrow_{I_1 \geq I_1(x) - 2\varepsilon} \Rightarrow y$
   (ii) The leaf $\mathcal{C}_\kappa(y, I_1)$ occurs later in the ℓ_1-increasing leaf enumeration that is used in pruning $(\mathscr{T}_1, \ell_1)$ to produce $(\mathscr{T}_2, \ell_2)$ than the leaf $\mathcal{C}_\kappa(x, I_1)$.

Here A is a consequence of the definitions of $\mathbf{FCTS}_\kappa(I)$ and $\bigwedge_{\mathscr{T}} \mathbf{S}$. (The special case of A in which $\mathbf{S} \subseteq \mathbf{Leaves}(\mathscr{T})$ is of particular interest; note that in this case $s \in \bigcup \mathbf{S}$ if and only if $\mathcal{C}_\kappa(s, I) \in \mathbf{S}$.) B is a consequence of the fact that $\Lambda_\kappa(I_{\mathbf{good}}) > 4\varepsilon$, C can be deduced from B by putting $\mathbf{L} = \{\mathbf{v}\}$ and $\mathbf{L}' = \{\mathbf{v}\} \cup \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}}[\mathbf{u}])$, and D can be deduced from A and C.

Assertion E is a consequence of A and the fact that $(\mathscr{T}_2, \ell_2)$ is the result of pruning $(\mathscr{T}_1, \ell_1)$ by removing branches of length $\leq 2\varepsilon$. In view of assertion (ii) of Lemma A1, we also have the following related fact:

E′. $\ell_1(\bigwedge_{\mathscr{T}_1} \{\mathbf{z}, \mathbf{z}'\}) < \min(\ell_1(\mathbf{z}), \ell_1(\mathbf{z}')) - 2\varepsilon$ whenever $\mathbf{z}$ and $\mathbf{z}'$ are distinct leaves of $\mathscr{T}_2$.

We could of course replace ℓ_1 with ℓ_2 in E′. Moreover, in view of assertion (i) of Lemma A1, we could also replace $\bigwedge_{\mathscr{T}_1}$ with $\bigwedge_{\mathscr{T}_2}$.

Now let x be any spel in $\mathcal{S}$. As $\mathfrak{F}_1$ is the result of pruning $\mathbf{FCTS}_\kappa(I') = (\mathscr{T}', \ell')$ by removing nodes of size $\leq k$, and $I_1 = I_{\mathfrak{F}_1}$, we see from the definition of $I_{\mathfrak{F}_1}$ that $I_1(x) = \max\{\ell'(\mathbf{u}) \mid \mathbf{u} \in \mathbf{Nodes}(\mathscr{T}'), |\mathbf{u}| \geq k+1, \text{ and } x \in \mathbf{u}\}$. This is equivalent to

$$I_1(x) = \max\left\{I'(y) \;\middle|\; y \in \mathcal{S},\ x \in \mathcal{C}_\kappa\left(y, I'\right),\ \text{and } \left|\mathcal{C}_\kappa\left(y, I'\right)\right| \geq k+1\right\} \tag{B4}$$

since the nodes $\mathbf{u} \in \mathbf{Nodes}(\mathscr{T}')$ for which $x \in \mathbf{u}$ are just the sets $\mathcal{C}_\kappa(y, I')$ for which $x \in \mathcal{C}_\kappa(y, I')$. Now we claim that:

$$I_1(x) = \max\left\{\tau \;\middle|\; \left|\mathcal{C}_\kappa\left(x, I', \tau\right)\right| \geq k+1\right\} \tag{B5}$$

To see this, we first observe that if y satisfies $x \in \mathcal{C}_\kappa(y, I')$ then y also satisfies $\mathcal{C}_\kappa(y, I') = \mathcal{C}_\kappa(x, I', I'(y))$. It follows from this observation that each element of the set $\{I'(y) \mid y \in \mathcal{S}, x \in \mathcal{C}_\kappa(y, I'), \text{ and } |\mathcal{C}_\kappa(y, I')| \geq k+1\}$ in (B4) belongs to the set $\{I'(y) \mid y \in \mathcal{S} \text{ and } |\mathcal{C}_\kappa(x, I', I'(y))| \geq k+1\}$ and therefore belongs to the set $\{\tau \mid |\mathcal{C}_\kappa(x, I', \tau)| \geq k+1\}$ in our claim (B5). So the right side of (B5) is no less than the right side of (B4); it remains to show that it is no greater.

For every $\tau \leq I'(x)$, let $y(\tau, x)$ be any spel in $\arg\min_{s \in \mathcal{C}_\kappa(x, I', \tau)} I'(s)$, so that $I'(y(\tau, x)) \geq \tau$, and it is easy to see that

$$\mathcal{C}_\kappa\left(y(\tau, x), I'\right) = \mathcal{C}_\kappa\left(x, I', \tau\right) \tag{B6}$$

since $I' \geq I'(y(\tau, x))$ at every spel in $\mathcal{C}_\kappa(x, I', \tau)$. Now if τ_0 is any element of the set $\{\tau \mid |\mathcal{C}_\kappa(x, I', \tau)| \geq k+1\}$, then we have that $I'(y(\tau_0, x)) \geq \tau_0$ and we see from

(B6) that $|\mathcal{C}_\kappa(y(\tau_0, x), I')| \geq k+1$ and $x \in \mathcal{C}_\kappa(y(\tau_0, x), I')$, so that $I'(y(\tau_0, x))$ is an element of $\{I'(y) \mid y \in \mathcal{S}, x \in \mathcal{C}_\kappa(y, I'),$ and $|\mathcal{C}_\kappa(y, I')| \geq k+1\}$ that is no less than τ_0. This shows that the right side of (B4) is no less than the right side of (B5). Hence the right sides of (B4) and (B5) are equal, and so our claim (B5) follows from (B4).

Next, we establish the following properties of I_1:

F. I_1 is an ε-perturbation of $I_{\mathbf{good}}$, and if $(I_a, I_b) = (I_1, I_{\mathbf{good}})$ or $(I_{\mathbf{good}}, I_1)$ then for any $\tau, \delta \in \mathbb{R}$ and any spels $s, t, u \in \mathcal{S}$ we have that:
   (i) If $s \Leftarrow_{I_a \geq \tau} \Rightarrow t$ then $s \Leftarrow_{I_b \geq \tau - \varepsilon} \Rightarrow t$.
   (ii) If $s \Leftarrow_{I_a \geq I_a(u) - \delta} \Rightarrow t$ then $s \Leftarrow_{I_b \geq I_b(u) - \delta - 2\varepsilon} \Rightarrow t$.

To see that I_1 has these properties, let x be any spel in $\mathcal{S}$ and note that $\mathcal{C}_\kappa(x, I_{\mathbf{good}}, \tau) \subseteq \mathcal{C}_\kappa(x, I', \tau - \varepsilon)$ for every $\tau \in \mathbb{R}$ since $\|I' - I_{\mathbf{good}}\|_\infty \leq \varepsilon$. On putting $\tau = I_{\mathbf{good}}(x)$, we deduce that $\mathcal{C}_\kappa(x, I', I_{\mathbf{good}}(x) - \varepsilon) \supseteq \mathcal{C}_\kappa(x, I_{\mathbf{good}}, I_{\mathbf{good}}(x)) = \mathcal{C}_\kappa(x, I_{\mathbf{good}})$, whence $|\mathcal{C}_\kappa(x, I', I_{\mathbf{good}}(x) - \varepsilon)| \geq |\mathcal{C}_\kappa(x, I_{\mathbf{good}})| \geq k+1$ (as $K_\kappa(I_{\mathbf{good}}) > k$). It follows from this and (B5) that $I_1(x) \geq I_{\mathbf{good}}(x) - \varepsilon$. On the other hand, whenever $\tau > I_{\mathbf{good}}(x) + \varepsilon$ we have that $I'(x) < \tau$ (as $\|I' - I_{\mathbf{good}}\|_\infty \leq \varepsilon$), which implies that $|\mathcal{C}_\kappa(x, I', \tau)| = 0$ and hence (by (B5)) that $I_1(x) < \tau$. From this it follows that $I_1(x) \leq I_{\mathbf{good}}(x) + \varepsilon$. This shows that I_1 is an ε-perturbation of $I_{\mathbf{good}}$, as F asserts. Now (i) follows immediately, and (ii) can be deduced from (i) by putting $\tau = I_a(u) - \delta$, since the fact that I_a is an ε-perturbation of I_b implies that $I_a(u) - \delta \geq I_b(u) - \delta - \varepsilon$ for every $u \in \mathcal{S}$.

B.3 Step 2 of the Proof of the Fundamental Lemma

The main goals of this step are to show that the mapping ϕ defined in step 1 of the proof is 1-to-1 and that the range of ϕ is exactly the subset **Leaves**$(\mathcal{T}_2)$ of **Leaves**$(\mathcal{T}_1)$. This will allow us to regard ϕ as a bijection ϕ : **Leaves**$(\mathcal{T}_{\mathbf{good}}) \rightarrow$ **Leaves**$(\mathcal{T}_2)$.

We first state and prove the following easy lemma:

Lemma B1 *Let* $\mathcal{C}_\kappa(v, I_{\mathbf{good}})$ *be any leaf of* $\mathcal{T}_{\mathbf{good}}$, *let* x *be any spel in* $\mathcal{S}$ *that satisfies* $x \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v) - 2\varepsilon} \Rightarrow v$, *and let* $\mathbf{s}$ *be any leaf of* $\mathcal{T}_1$ *such that* $\mathbf{s} \succeq_{\mathcal{T}_1} \mathcal{C}_\kappa(x, I_1)$. *Then* $\mathbf{s} \in \mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$.

Proof Let z be a spel that satisfies (B1) with respect to v. Then (B2) implies that $x \in \mathcal{C}_\kappa(z, I_1)$ and hence that $\mathcal{C}_\kappa(x, I_1) \succeq_{\mathcal{T}_1} \mathcal{C}_\kappa(z, I_1)$. This and (B3) imply $\mathbf{s} \in \mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$. □

Next, we establish the following properties of $\mathcal{M}$ and the mapping ϕ:

G. The following are true for any leaf $\mathcal{C}_\kappa(v, I_{\mathbf{good}})$ of $\mathcal{T}_{\mathbf{good}}$:
   (a) If $\mathcal{C}_\kappa(y, I_1) \in \mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$, then:

(i) $y \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v)-4\varepsilon} \Rightarrow v$
(ii) $y \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(y)} \Rightarrow v$
(iii) $y \Leftarrow_{I_1 \geq I_1(y)-2\varepsilon} \Rightarrow v$

(b) If $\mathcal{C}_\kappa(y, I_1) = \phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$, then:
(i) $I_{\mathbf{good}}(v) + \varepsilon \geq I_1(y) \geq I_1(v) \geq I_{\mathbf{good}}(v) - \varepsilon$
(ii) $y \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v)-2\varepsilon} \Rightarrow v$
(iii) $\mathcal{C}_\kappa(y, I_1) \in \mathbf{Leaves}(\mathscr{T}_2)$

To establish (a), let $\mathcal{C}_\kappa(v, I_{\mathbf{good}})$ be any leaf of $\mathscr{T}_{\mathbf{good}}$ and let $\mathcal{C}_\kappa(y, I_1)$ be an arbitrary element of $\mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$. Then it follows from the definition of the set $\mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$ that $\mathcal{C}_\kappa(y, I_1) \subseteq \mathcal{C}_\kappa(z, I_1)$ for some spel z that satisfies the condition $v \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v)-2\varepsilon} \Rightarrow z$ (which implies $I_{\mathbf{good}}(z) \geq I_{\mathbf{good}}(v) - 2\varepsilon$). Since $\mathcal{C}_\kappa(y, I_1) \subseteq \mathcal{C}_\kappa(z, I_1)$, we have that $z \Leftarrow_{I_1 \geq I_1(z)} \Rightarrow y$. This implies $z \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(z)-2\varepsilon} \Rightarrow y$ (in view of assertion (ii) of F), which implies $z \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v)-4\varepsilon} \Rightarrow y$ (as $I_{\mathbf{good}}(z) \geq I_{\mathbf{good}}(v) - 2\varepsilon$).

Combining $z \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v)-4\varepsilon} \Rightarrow y$ with $v \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v)-2\varepsilon} \Rightarrow z$, we deduce assertion (i) of (a). Now (ii) follows from (i) and D because $\mathcal{C}_\kappa(v, I_{\mathbf{good}}) \in \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$, and (iii) follows from (ii) and F.

Now we establish (b). Suppose $\mathcal{C}_\kappa(y, I_1) = \phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$. Consider the node $\mathcal{C}_\kappa(v, I_1)$ of $\mathscr{T}_1$. Let $\mathbf{s}$ be a leaf of $\mathscr{T}_1$ such that $\mathbf{s} \succeq_{\mathscr{T}_1} \mathcal{C}_\kappa(v, I_1)$. Then we have that $\mathbf{s} \in \mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$, by Lemma B1. Hence $\ell_1(\mathcal{C}_\kappa(y, I_1)) \geq \ell_1(\mathbf{s})$ (as $\mathbf{s}$ cannot occur later in the ℓ_1-increasing leaf enumeration that is used in pruning $(\mathscr{T}_1, \ell_1)$ than $\phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}})) = \mathcal{C}_\kappa(y, I_1)$, by the definition of $\phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$). Therefore

$$I_1(y) = \ell_1\big(\mathcal{C}_\kappa(y, I_1)\big) \geq \ell_1(\mathbf{s}) \geq \ell_1\big(\mathcal{C}_\kappa(v, I_1)\big) = I_1(v) \tag{B7}$$

which establishes the second inequality of assertion (i) of (b). The third inequality of (i) follows from F. Now $I_{\mathbf{good}}(v) \geq I_{\mathbf{good}}(y)$ (by assertion (ii) of (a)). This implies $I_{\mathbf{good}}(v) \geq I_1(y) - \varepsilon$ (by F), which is equivalent to the first inequality of assertion (i) of (b). This establishes assertion (i) of (b). It follows from F and assertion (i) of (b) that $I_{\mathbf{good}}(y) \geq I_{\mathbf{good}}(v) - 2\varepsilon$. Assertion (ii) of (b) follows from this and assertion (ii) of (a).

To see that assertion (iii) of (b) holds, let $\mathcal{C}_\kappa(w, I_1)$ be any leaf of $\mathscr{T}_1$ that occurs later in the ℓ_1-increasing leaf enumeration that is used in pruning $(\mathscr{T}_1, \ell_1)$ than $\phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}})) = \mathcal{C}_\kappa(y, I_1)$. Then it follows from the definitions of $\phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$ and of an ℓ_1-increasing leaf enumeration that:

- $\mathcal{C}_\kappa(w, I_1) \notin \mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$
- $I_1(w) = \ell_1(\mathcal{C}_\kappa(w, I_1)) \geq \ell_1(\mathcal{C}_\kappa(y, I_1)) = I_1(y)$

As $I_1(w) \geq I_1(y)$, (B7) implies that $I_1(w) \geq I_1(v)$, and now it follows from F that $I_{\mathbf{good}}(w) \geq I_{\mathbf{good}}(v) - 2\varepsilon$. So $\mathcal{C}_\kappa(v, I_{\mathbf{good}}) \not\succeq_{\mathscr{T}_{\mathbf{good}}} \mathcal{C}_\kappa(w, I_{\mathbf{good}})$; otherwise the spel w would satisfy $w \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(w)} \Rightarrow v$, which would imply that $w \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v)-2\varepsilon} \Rightarrow v$ (since $I_{\mathbf{good}}(w) \geq I_{\mathbf{good}}(v) - 2\varepsilon$), which would in turn imply that $\mathcal{C}_\kappa(w, I_1)$ is an element of $\mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$ (by Lemma B1), which is false as we saw above.

Since $\mathcal{C}_\kappa(v, I_{\mathbf{good}}) \not\succeq_{\mathscr{T}_{\mathbf{good}}} \mathcal{C}_\kappa(w, I_{\mathbf{good}})$, it follows from C and A that w does *not* satisfy $w \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v)-4\varepsilon} \Rightarrow v$. This and assertion (ii) of F imply that w does *not* satisfy $w \Leftarrow_{I_1 \geq I_1(v)-2\varepsilon} \Rightarrow v$, and so (since $I_1(y) \geq I_1(v)$, by (B7)) w does *not* satisfy

$w \Leftarrow_{I_1 \geq I_1(y)-2\varepsilon} \Rightarrow v$. But we know from assertion (iii) of (a) that $y \Leftarrow_{I_1 \geq I_1(y)-2\varepsilon} \Rightarrow v$, so w also does *not* satisfy $w \Leftarrow_{I_1 \geq I_1(y)-2\varepsilon} \Rightarrow y$. As $\mathcal{C}_\kappa(w, I_1)$ is an arbitrary leaf of $\mathscr{T}_1$ that occurs later in the ℓ_1-increasing leaf enumeration used in pruning $(\mathscr{T}_1, \ell_1)$ than the leaf $\phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$, we see from E that $\phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}})) \in \mathbf{Leaves}(\mathscr{T}_2)$—i.e., assertion (iii) of (b) holds.

Since $\phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}})) \in \mathbf{Leaves}(\mathscr{T}_2)$ for every leaf $\mathcal{C}_\kappa(v, I_{\mathbf{good}})$ of $\mathscr{T}_{\mathbf{good}}$, we can regard ϕ as a mapping $\phi : \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}}) \to \mathbf{Leaves}(\mathscr{T}_2)$, and we will do this from now on.

We next show that $\phi : \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}}) \to \mathbf{Leaves}(\mathscr{T}_2)$ is 1-to-1:

H. $\phi(\mathbf{v}) \neq \phi(\mathbf{v}')$ whenever $\mathbf{v}$ and $\mathbf{v}'$ are distinct leaves of $\mathscr{T}_{\mathbf{good}}$.

Indeed, let $\mathcal{C}_\kappa(v_\mathrm{a}, I_{\mathbf{good}})$ and $\mathcal{C}_\kappa(v_\mathrm{b}, I_{\mathbf{good}})$ be any two distinct leaves of $\mathscr{T}_{\mathbf{good}}$. To establish H, it is enough to show that $\mathcal{M}(\mathcal{C}_\kappa(v_\mathrm{a}, I_{\mathbf{good}}))$ and $\mathcal{M}(\mathcal{C}_\kappa(v_\mathrm{b}, I_{\mathbf{good}}))$ are disjoint. Suppose this is not the case. Then there is a leaf $\mathcal{C}_\kappa(x, I_1)$ of $\mathscr{T}_1$ such that $\mathcal{C}_\kappa(x, I_1) \in \mathcal{M}(\mathcal{C}_\kappa(v_\mathrm{a}, I_{\mathbf{good}}))$ and $\mathcal{C}_\kappa(x, I_1) \in \mathcal{M}(\mathcal{C}_\kappa(v_\mathrm{b}, I_{\mathbf{good}}))$. Now assertion (i) of part (a) of G implies that $v_\mathrm{a} \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v_\mathrm{a})-4\varepsilon} \Rightarrow x$ and that $v_\mathrm{b} \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v_\mathrm{b})-4\varepsilon} \Rightarrow x$.

Assuming without loss of generality that $I_{\mathbf{good}}(v_\mathrm{a}) \leq I_{\mathbf{good}}(v_\mathrm{b})$, these two properties imply that $v_\mathrm{a} \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v_\mathrm{a})-4\varepsilon} \Rightarrow v_\mathrm{b}$, which is impossible in view of C and A. This contradiction establishes H and shows that ϕ is 1-to-1.

Next, we show that:

I. $\mathbf{Leaves}(\mathscr{T}_2) \setminus \phi[\mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})] = \emptyset$

To justify I, let $\mathcal{C}_\kappa(x, I_1)$ be any element of $\mathbf{Leaves}(\mathscr{T}_1) \setminus \phi[\mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})]$. Then what we need to show is that $\mathcal{C}_\kappa(x, I_1) \notin \mathbf{Leaves}(\mathscr{T}_2)$.

Let $\mathcal{C}_\kappa(v, I_{\mathbf{good}})$ be a leaf of $\mathscr{T}_{\mathbf{good}}$ such that $\mathcal{C}_\kappa(x, I_{\mathbf{good}}) \supseteq \mathcal{C}_\kappa(v, I_{\mathbf{good}})$. Then $x \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(x)} \Rightarrow v$ and so it follows from F that $x \Leftarrow_{I_1 \geq I_1(x)-2\varepsilon} \Rightarrow v$. Let $\mathcal{C}_\kappa(y, I_1) = \phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$. We now claim that:

- $\mathcal{C}_\kappa(y, I_1)$ occurs later in the ℓ_1-increasing leaf enumeration that is used in pruning $(\mathscr{T}_1, \ell_1)$ than $\mathcal{C}_\kappa(x, I_1)$.

Now we justify this claim. Just one of the following is true:

(a) $I_{\mathbf{good}}(v) - 2\varepsilon > I_{\mathbf{good}}(x)$
(b) $I_{\mathbf{good}}(x) \geq I_{\mathbf{good}}(v) - 2\varepsilon$

In case (a) it follows from F that $I_1(v) > I_1(x)$, and so $I_1(y) > I_1(x)$ (since $I_1(y) \geq I_1(v)$, by assertion (i) of part (b) of G); thus our claim is valid.

In case (b), we first observe that, since $x \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(x)} \Rightarrow v$, (b) implies that $x \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v)-2\varepsilon} \Rightarrow v$, so that $\mathcal{C}_\kappa(x, I_1) \in \mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$ (by Lemma B1). Therefore $\mathcal{C}_\kappa(x, I_1) \in \mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}})) \setminus \{\phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))\}$, because $\mathcal{C}_\kappa(x, I_1)$ is an element of $\mathbf{Leaves}(\mathscr{T}_1) \setminus \phi[\mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})]$. As $\mathcal{C}_\kappa(y, I_1) = \phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$ and $\mathcal{C}_\kappa(x, I_1) \in \mathcal{M}(\mathcal{C}_\kappa(v, I_{\mathbf{good}})) \setminus \{\phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))\}$, it follows from the definition of ϕ that our claim is again valid.

In either case, we have that $x \Leftarrow_{I_1 \geq I_1(x)-2\varepsilon} \Rightarrow v$ (as we saw above), and the claim implies $I_1(y) \geq I_1(x)$. So, since we see from assertion (iii) of part (a) of G that $v \Leftarrow_{I_1 \geq I_1(y)-2\varepsilon} \Rightarrow y$, we also have that $x \Leftarrow_{I_1 \geq I_1(x)-2\varepsilon} \Rightarrow y$. From this, E, and the above claim, we deduce that $\mathcal{C}_\kappa(x, I_1) \notin \mathbf{Leaves}(\mathscr{T}_2)$. This justifies I.

It follows from H and I that $\phi : \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}}) \to \mathbf{Leaves}(\mathscr{T}_2)$ is a bijection. This completes step 2 of the proof of the Fundamental Lemma.

B.4 Step 3 of the Proof of the Fundamental Lemma

We now extend ϕ to a mapping $\varphi : \mathbf{Crit}(\mathscr{T}_{\mathbf{good}}) \to \mathbf{Crit}(\mathscr{T}_2)$ by defining $\varphi(\mathbf{u}) = \bigwedge_{\mathscr{T}_2} \phi[\mathbf{Leaves}(\mathscr{T}_{\mathbf{good}}[\mathbf{u}])]$. We will establish two properties of the mapping φ which together imply that φ is an essential isomorphism of $\mathfrak{F}_{\mathbf{good}}$ to $\mathfrak{F}_3$. The first property is that, for all $\mathbf{u}, \mathbf{u}' \in \mathbf{Crit}(\mathscr{T}_{\mathbf{good}})$, $\varphi(\mathbf{u}) \preceq_{\mathscr{T}_2} \varphi(\mathbf{u}')$ if and only if $\mathbf{u} \preceq_{\mathscr{T}_{\mathbf{good}}} \mathbf{u}'$ (so that φ is an order-preserving injection). The second property is that $\varphi[\mathbf{Crit}(\mathscr{T}_{\mathbf{good}})] = \mathbf{Crit}(\mathscr{T}_3)$. To establish these two properties, we first show that:

J. $|\ell_2(\bigwedge_{\mathscr{T}_2} \phi[\mathbf{L}]) - \ell_{\mathbf{good}}(\bigwedge_{\mathscr{T}_{\mathbf{good}}} \mathbf{L})| \le \varepsilon$ whenever $\emptyset \ne \mathbf{L} \subseteq \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$.

Indeed, suppose $\emptyset \ne \mathbf{L} \subseteq \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$. If $|\mathbf{L}| = 1$, then J is an immediate consequence of assertion (i) of part (b) of G, so we will assume $|\mathbf{L}| \ge 2$.

For brevity, we will write $\tau_{\mathbf{L}}$ for $\ell_{\mathbf{good}}(\bigwedge_{\mathscr{T}_{\mathbf{good}}} \mathbf{L})$ and $\tau_{\phi[\mathbf{L}]}$ for $\ell_2(\bigwedge_{\mathscr{T}_2} \phi[\mathbf{L}])$, so that J can be written as $|\tau_{\phi[\mathbf{L}]} - \tau_{\mathbf{L}}| \le \varepsilon$.

We first show that $\tau_{\phi[\mathbf{L}]} \ge \tau_{\mathbf{L}} - \varepsilon$. For this purpose, let $\mathcal{C}_\kappa(x, I_1)$ and $\mathcal{C}_\kappa(y, I_1)$ be any two distinct elements of $\phi[\mathbf{L}]$. Then $\mathcal{C}_\kappa(x, I_1) = \phi(\mathcal{C}_\kappa(u, I_{\mathbf{good}}))$ and $\mathcal{C}_\kappa(y, I_1) = \phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$, where $\mathcal{C}_\kappa(u, I_{\mathbf{good}})$ and $\mathcal{C}_\kappa(v, I_{\mathbf{good}})$ are two distinct elements of $\mathbf{L}$. From A and the definition of $\tau_{\mathbf{L}}$ we see that $u \Leftarrow\!\!I_{\mathbf{good}} \ge \tau_{\mathbf{L}} \Rrightarrow v$. This and F imply that $u \Leftarrow\!\!I_1 \ge \tau_{\mathbf{L}} - \varepsilon \Rrightarrow v$. We see from the definition of ϕ and assertion (iii) of part (a) of G that $x \Leftarrow\!\!I_1 \ge I_1(x) - 2\varepsilon \Rrightarrow u$ and $y \Leftarrow\!\!I_1 \ge I_1(y) - 2\varepsilon \Rrightarrow v$. Combining the last three observations, we deduce that:

$$x \Leftarrow\!\!I_1 \ge \min(\tau_{\mathbf{L}} - \varepsilon, I_1(x) - 2\varepsilon, I_1(y) - 2\varepsilon) \Rrightarrow y \tag{B8}$$

However, it follows from C and the definition of $\tau_{\mathbf{L}}$ that

$$\begin{aligned}
\tau_{\mathbf{L}} &\le \ell_{\mathbf{good}}\Big(\bigwedge_{\mathscr{T}_{\mathbf{good}}} \big\{\mathcal{C}_\kappa(u, I_{\mathbf{good}}), \mathcal{C}_\kappa(v, I_{\mathbf{good}})\big\}\Big) \\
&< \min\big(\ell_{\mathbf{good}}\big(\mathcal{C}_\kappa(u, I_{\mathbf{good}})\big) - 4\varepsilon, \ell_{\mathbf{good}}\big(\mathcal{C}_\kappa(v, I_{\mathbf{good}})\big) - 4\varepsilon\big) \\
&= \min\big(I_{\mathbf{good}}(u) - 4\varepsilon, I_{\mathbf{good}}(v) - 4\varepsilon\big)
\end{aligned}$$

which implies that $\tau_{\mathbf{L}} - \varepsilon < \min(I_{\mathbf{good}}(u) - 5\varepsilon, I_{\mathbf{good}}(v) - 5\varepsilon)$, which implies that $\tau_{\mathbf{L}} - \varepsilon < \min(I_1(u) - 4\varepsilon, I_1(v) - 4\varepsilon)$ (in view of F), which in turn implies that $\tau_{\mathbf{L}} - \varepsilon < \min(I_1(x) - 4\varepsilon, I_1(y) - 4\varepsilon)$ (by assertion (i) of part (b) of G). So (B8) can be simplified to $x \Leftarrow\!\!I_1 \ge \tau_{\mathbf{L}} - \varepsilon \Rrightarrow y$. It now follows from A that $\tau_{\phi[\mathbf{L}]} \ge \tau_{\mathbf{L}} - \varepsilon$ (since $\mathcal{C}_\kappa(x, I_1)$ and $\mathcal{C}_\kappa(y, I_1)$ are arbitrary distinct elements of $\phi[\mathbf{L}]$), as required.

To complete the proof of J, we show that $\tau_{\mathbf{L}} \ge \tau_{\phi[\mathbf{L}]} - \varepsilon$. This time we let $\mathcal{C}_\kappa(u, I_{\mathbf{good}})$ and $\mathcal{C}_\kappa(v, I_{\mathbf{good}})$ be any two distinct elements of $\mathbf{L}$, and then define $\mathcal{C}_\kappa(x, I_1) = \phi(\mathcal{C}_\kappa(u, I_{\mathbf{good}}))$ and $\mathcal{C}_\kappa(y, I_1) = \phi(\mathcal{C}_\kappa(v, I_{\mathbf{good}}))$, so that $\mathcal{C}_\kappa(x, I_1)$, $\mathcal{C}_\kappa(y, I_1) \in \phi[\mathbf{L}]$. From A and the definition of $\tau_{\phi[\mathbf{L}]}$ we see that $x \Leftarrow\!\!I_1 \ge \tau_{\phi[\mathbf{L}]} \Rrightarrow y$. This and F imply that $x \Leftarrow\!\!I_{\mathbf{good}} \ge \tau_{\phi[\mathbf{L}]} - \varepsilon \Rrightarrow y$. We see from assertion (ii) of part (b) of G that

$u \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(u)-2\varepsilon} \Rightarrow x$; we similarly have that $v \Leftarrow_{I_{\mathbf{good}} \geq I_{\mathbf{good}}(v)-2\varepsilon} \Rightarrow y$. Combining the last three observations, we see that:

$$u \Leftarrow_{I_{\mathbf{good}} \geq \min(\tau_{\phi[\mathbf{L}]}-\varepsilon, I_{\mathbf{good}}(u)-2\varepsilon, I_{\mathbf{good}}(v)-2\varepsilon)} \Rightarrow v \tag{B9}$$

However, it follows from the definition of $\tau_{\phi[\mathbf{L}]}$ and E′ that:

$$\begin{aligned}\tau_{\phi[\mathbf{L}]} &\leq \ell_2\Big(\bigwedge\nolimits_{\mathscr{T}_2}\big\{\phi\big(\mathcal{C}_\kappa(u, I_{\mathbf{good}})\big), \phi\big(\mathcal{C}_\kappa(v, I_{\mathbf{good}})\big)\big\}\Big)\\ &< \min\Big(\ell_2\big(\phi\big(\mathcal{C}_\kappa(u, I_{\mathbf{good}})\big)\big), \ell_2\big(\phi\big(\mathcal{C}_\kappa(v, I_{\mathbf{good}})\big)\big)\Big) - 2\varepsilon\\ &= \min\Big(\ell_2\big(\mathcal{C}_\kappa(x, I_1)\big), \ell_2\big(\mathcal{C}_\kappa(y, I_1)\big)\Big) - 2\varepsilon = \min\big(I_1(x), I_1(y)\big) - 2\varepsilon\end{aligned}$$

Hence $\tau_{\phi[\mathbf{L}]} - \varepsilon < \min(I_1(x) - 3\varepsilon, I_1(y) - 3\varepsilon)$, which (by assertion (i) of part (b) of G) implies $\tau_{\phi[\mathbf{L}]} - \varepsilon < \min(I_{\mathbf{good}}(u) - 2\varepsilon, I_{\mathbf{good}}(v) - 2\varepsilon)$. We now see from (B9) that $u \Leftarrow_{I_{\mathbf{good}} \geq \tau_{\phi[\mathbf{L}]}-\varepsilon} \Rightarrow v$. It follows from this and A that $\tau_{\mathbf{L}} \geq \tau_{\phi[\mathbf{L}]} - \varepsilon$ (since $\mathcal{C}_\kappa(u, I_{\mathbf{good}})$ and $\mathcal{C}_\kappa(v, I_{\mathbf{good}})$ are arbitrary distinct elements of $\mathbf{L}$), as required. Thus we have established J.

From B and J, we deduce:

K. Whenever $\emptyset \neq \mathbf{L} \subseteq \mathbf{L}' \subseteq \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$, $\bigwedge_{\mathscr{T}_{\mathbf{good}}} \mathbf{L}' = \bigwedge_{\mathscr{T}_{\mathbf{good}}} \mathbf{L}$ if and only if $\ell_2(\bigwedge_{\mathscr{T}_2} \phi[\mathbf{L}]) - \ell_2(\bigwedge_{\mathscr{T}_2} \phi[\mathbf{L}']) \leq 2\varepsilon$.

As we show in Appendix C, it is not difficult to deduce from K that:

L. For all $\mathbf{u} \in \mathbf{Crit}(\mathscr{T}_{\mathbf{good}})$, $\mathbf{Leaves}(\mathscr{T}_2[\varphi(\mathbf{u})]) = \varphi[\mathbf{Leaves}(\mathscr{T}_{\mathbf{good}}[\mathbf{u}])]$.
M. For all $\mathbf{x} \in \varphi[\mathbf{Crit}(\mathscr{T}_{\mathbf{good}})]$, there is no $\mathbf{y} \in \mathbf{x}{\downarrow_{\mathscr{T}_2}} \cap \mathbf{Crit}(\mathscr{T}_2)$ that satisfies the condition $\ell_2(\mathbf{x}) - \ell_2(\mathbf{y}) \leq 2\varepsilon$.
N. For all $\mathbf{x} \in \mathbf{Crit}(\mathscr{T}_2)$, some $\mathbf{z} \in \mathbf{x}{\Downarrow_{\mathscr{T}_2}} \cap \varphi[\mathbf{Crit}(\mathscr{T}_{\mathbf{good}})]$ satisfies the condition $\ell_2(\mathbf{x}) - \ell_2(\mathbf{z}) \leq 2\varepsilon$.

We mention here that N is proved by showing that for every $\mathbf{x} \in \mathbf{Crit}(\mathscr{T}_2)$ the node $\mathbf{z} = \varphi(\bigwedge_{\mathscr{T}_{\mathbf{good}}} \varphi^{-1}[\mathbf{Leaves}(\mathscr{T}_2[\mathbf{x}])])$ has the stated property.

Using L, it is quite easy to show that:

O. For all $\mathbf{u}, \mathbf{u}' \in \mathbf{Crit}(\mathscr{T}_{\mathbf{good}})$, $\varphi(\mathbf{u}) \preceq_{\mathscr{T}_2} \varphi(\mathbf{u}')$ if and only if $\mathbf{u} \preceq_{\mathscr{T}_{\mathbf{good}}} \mathbf{u}'$.

Details of the proof of O are given in Appendix C. It follows from O that φ is an order-preserving injection.

As $\mathfrak{F}_3 = (\mathscr{T}_3, \ell_3)$ is the result of eliminating internal edges of length $\leq 2\varepsilon$ from $\mathfrak{F}_2^{\mathbf{crit}}$, it follows from M and property E8 of simplification step 3 that φ must satisfy $\varphi[\mathbf{Crit}(\mathscr{T}_{\mathbf{good}})] \subseteq \mathbf{Crit}(\mathscr{T}_2) \cap \mathbf{Nodes}(\mathscr{T}_3) = \mathbf{Crit}(\mathscr{T}_3)$. Moreover, N implies that, for all $\mathbf{x} \in \mathbf{Crit}(\mathscr{T}_2) \setminus \varphi[\mathbf{Crit}(\mathscr{T}_{\mathbf{good}})]$, some $\mathbf{z} \in \mathbf{x}{\downarrow_{\mathscr{T}_2}} \cap \varphi[\mathbf{Crit}(\mathscr{T}_{\mathbf{good}})]$ satisfies $\ell_2(\mathbf{x}) - \ell_2(\mathbf{z}) \leq 2\varepsilon$. We therefore have that:

- For all $\mathbf{x} \in \mathbf{Crit}(\mathscr{T}_2) \setminus \varphi[\mathbf{Crit}(\mathscr{T}_{\mathbf{good}})]$, some $\mathbf{z} \in \mathbf{x}{\downarrow_{\mathscr{T}_2}} \cap \mathbf{Crit}(\mathscr{T}_3)$ satisfies the condition $\ell_2(\mathbf{x}) - \ell_2(\mathbf{z}) \leq 2\varepsilon$.

From this and property E9 of simplification step 3 we deduce that φ satisfies $(\mathbf{Crit}(\mathscr{T}_2) \setminus \varphi[\mathbf{Crit}(\mathscr{T}_{\mathbf{good}})]) \cap \mathbf{Nodes}(\mathscr{T}_3) = \emptyset$. Equivalently, φ satisfies the condition $\mathbf{Crit}(\mathscr{T}_3) \setminus \varphi[\mathbf{Crit}(\mathscr{T}_{\mathbf{good}})] = \emptyset$. Thus $\varphi[\mathbf{Crit}(\mathscr{T}_{\mathbf{good}})] = \mathbf{Crit}(\mathscr{T}_3)$. So the

order-preserving injection φ can be regarded as a bijection $\varphi : \mathbf{Crit}(\mathscr{T}_{\mathbf{good}}) \to \mathbf{Crit}(\mathscr{T}_3)$. When so regarded, φ is an essential isomorphism of $\mathfrak{F}_{\mathbf{good}}$ to $\mathfrak{F}_3$. Finally, φ is level-preserving to within ε because, for any node $\mathbf{u} \in \mathbf{Crit}(\mathscr{T}_{\mathbf{good}})$, we deduce from J (on putting $\mathbf{L} = \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}}[\mathbf{u}])$, so that $\bigwedge_{\mathscr{T}_{\mathbf{good}}} \mathbf{L} = \mathbf{u}$) that $|\ell_3(\varphi(\mathbf{u})) - \ell_{\mathbf{good}}(\mathbf{u})| \le \varepsilon$.

This completes the proof of the Fundamental Lemma.

Appendix C: Justification of Assertions L, M, N, and O in Step 3 of the Proof of the Fundamental Lemma

For any rooted tree $\mathscr{T}$ and any $\mathbf{u} \in \mathbf{Crit}(\mathscr{T})$, we write $\mathcal{L}_{\mathscr{T}}\mathbf{u}$ to denote the set $\mathbf{Leaves}(\mathscr{T}[\mathbf{u}]) = \{\mathbf{v} \in \mathbf{Leaves}(\mathscr{T}) \mid \mathbf{u} \preceq_{\mathscr{T}} \mathbf{v}\}$. It is readily confirmed that the following are true in any rooted tree $\mathscr{T}$:

$$\text{If } \emptyset \neq \mathbf{L} \subseteq \mathbf{L}' \subseteq \mathbf{Leaves}(\mathscr{T}), \text{ then: } \quad \bigwedge_{\mathscr{T}} \mathbf{L}' \preceq_{\mathscr{T}} \bigwedge_{\mathscr{T}} \mathbf{L} \tag{C1}$$

$$\text{If } \emptyset \neq \mathbf{L} \subseteq \mathbf{Leaves}(\mathscr{T}), \text{ then: } \quad \mathcal{L}_{\mathscr{T}} \bigwedge_{\mathscr{T}} \mathbf{L} \supseteq \mathbf{L} \tag{C2}$$

$$\text{If } \mathbf{u} \in \mathbf{Crit}(\mathscr{T}), \text{ then: } \quad \bigwedge_{\mathscr{T}} \mathcal{L}_{\mathscr{T}}\mathbf{u} = \mathbf{u} \tag{C3}$$

$$\text{If } \mathbf{u} \in \mathbf{Crit}(\mathscr{T}) \text{ and } \mathbf{L} \supsetneq \mathcal{L}_{\mathscr{T}}\mathbf{u}, \text{ then: } \quad \bigwedge_{\mathscr{T}} \mathbf{L} \prec_{\mathscr{T}} \mathbf{u} = \bigwedge_{\mathscr{T}} \mathcal{L}_{\mathscr{T}}\mathbf{u} \tag{C4}$$

$$\text{If } \mathbf{u}, \mathbf{v} \in \mathbf{Crit}(\mathscr{T}), \text{ then: } \quad \mathcal{L}_{\mathscr{T}}\mathbf{v} = \mathcal{L}_{\mathscr{T}}\mathbf{u} \text{ if and only if } \mathbf{v} = \mathbf{u} \tag{C5}$$

$$\text{If } \mathbf{u}, \mathbf{v} \in \mathbf{Crit}(\mathscr{T}), \text{ then: } \quad \mathcal{L}_{\mathscr{T}}\mathbf{v} \supsetneq \mathcal{L}_{\mathscr{T}}\mathbf{u} \text{ if and only if } \mathbf{v} \prec_{\mathscr{T}} \mathbf{u} \tag{C6}$$

For all $\mathbf{L} \subseteq \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$ and all $\mathbf{L}' \subseteq \mathbf{Leaves}(\mathscr{T}_2)$, we write $\phi\mathbf{L}$ to mean $\phi[\mathbf{L}]$ and we write $\phi^{-1}\mathbf{L}$ to mean $\phi^{-1}[\mathbf{L}]$.

If $\mathbf{x} \preceq_{\mathscr{T}_2} \mathbf{y}$ or $\mathbf{y} \preceq_{\mathscr{T}_2} \mathbf{x}$, and λ is any positive value, then we write $\mathbf{x} \approx_\lambda \mathbf{y}$ to mean that $|\ell_2(\mathbf{y}) - \ell_2(\mathbf{x})| \le \lambda$, and write $\mathbf{x} \prec_\lambda \mathbf{y}$ to mean that $\ell_2(\mathbf{y}) - \ell_2(\mathbf{x}) > \lambda$; in the latter case we must have that $\mathbf{x} \prec_{\mathscr{T}_2} \mathbf{y}$. For brevity, we will write $\bigwedge_{\mathbf{good}}$ and $\bigwedge_2$ to mean $\bigwedge_{\mathscr{T}_{\mathbf{good}}}$ and $\bigwedge_{\mathscr{T}_2}$, and write $\mathcal{L}_{\mathbf{good}}$ and $\mathcal{L}_2$ to mean $\mathcal{L}_{\mathscr{T}_{\mathbf{good}}}$ and $\mathcal{L}_{\mathscr{T}_2}$. Note that the definition of the mapping φ can be rewritten in terms of ϕ and $\mathcal{L}_{\mathbf{good}}$ as follows:

$$\varphi(\mathbf{u}) \stackrel{\text{def}}{=} \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \tag{C7}$$

If $\emptyset \neq \mathbf{L} \subseteq \mathbf{L}' \subseteq \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$, then $\emptyset \neq \phi\mathbf{L} \subseteq \phi\mathbf{L}' \subseteq \mathbf{Leaves}(\mathscr{T}_2)$ and so $\bigwedge_2 \phi\mathbf{L}' \preceq_{\mathscr{T}_2} \bigwedge_2 \phi\mathbf{L}$ (by (C1)). Hence assertion K can be restated as follows (for all nonempty sets $\mathbf{L} \subseteq \mathbf{L}' \subseteq \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$):

$$\bigwedge_2 \phi\mathbf{L}' \approx_{2\varepsilon} \bigwedge_2 \phi\mathbf{L} \quad \text{if and only if} \quad \bigwedge_{\mathbf{good}} \mathbf{L}' = \bigwedge_{\mathbf{good}} \mathbf{L} \tag{C8}$$

When $\emptyset \neq \mathbf{L} \subseteq \mathbf{L}' \subseteq \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$, the negations of $\bigwedge_2 \phi\mathbf{L} \approx_{2\varepsilon} \bigwedge_2 \phi\mathbf{L}'$ and $\bigwedge_{\mathbf{good}} \mathbf{L}' = \bigwedge_{\mathbf{good}} \mathbf{L}$ are $\bigwedge_2 \phi\mathbf{L}' \prec_{2\varepsilon} \bigwedge_2 \phi\mathbf{L}$ and $\bigwedge_{\mathbf{good}} \mathbf{L}' \prec_{\mathscr{T}_{\mathbf{good}}} \bigwedge_{\mathbf{good}} \mathbf{L}$ respectively (since $\bigwedge_{\mathbf{good}} \mathbf{L}' \preceq_{\mathscr{T}_{\mathbf{good}}} \bigwedge_{\mathbf{good}} \mathbf{L}$ and $\bigwedge_2 \phi\mathbf{L}' \preceq_{\mathscr{T}_2} \bigwedge_2 \phi\mathbf{L}$), so (C8) can also be stated as follows (for all nonempty sets $\mathbf{L} \subseteq \mathbf{L}' \subseteq \mathbf{Leaves}(\mathscr{T}_{\mathbf{good}})$):

$$\bigwedge_2 \phi\mathbf{L}' \prec_{2\varepsilon} \bigwedge_2 \phi\mathbf{L} \quad \text{if and only if} \quad \bigwedge_{\mathbf{good}} \mathbf{L}' \prec_{\mathscr{T}_{\mathbf{good}}} \bigwedge_{\mathbf{good}} \mathbf{L} \tag{C9}$$

C.1 Proof of Assertion L

In view of (C7), L can be restated as follows:

- For all $\mathbf{u} \in \mathbf{Crit}(\mathscr{T}_{\mathbf{good}})$, we have that $\mathcal{L}_2 \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} = \phi\mathcal{L}_{\mathbf{good}}\mathbf{u}$. Equivalently, $\phi^{-1}\mathcal{L}_2 \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} = \mathcal{L}_{\mathbf{good}}\mathbf{u}$.

To prove this, let $\mathbf{u} \in \mathbf{Crit}(\mathscr{T}_{\mathbf{good}})$. Then we successively deduce:

$$\begin{aligned}
\mathcal{L}_2 \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} &\supseteq \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \quad \text{[by (C2)]} \\
\phi^{-1}\mathcal{L}_2 \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} &\supseteq \phi^{-1}\phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \\
\phi^{-1}\mathcal{L}_2 \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} &\supseteq \mathcal{L}_{\mathbf{good}}\mathbf{u} \qquad \text{(C10)}
\end{aligned}$$

The result will follow from (C10) if we can show that the following is *not* true:

$$\phi^{-1}\mathcal{L}_2 \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \supsetneq \mathcal{L}_{\mathbf{good}}\mathbf{u} \qquad \text{(C11)}$$

To do this, we derive a contradiction from (C11) as follows:

$$\begin{aligned}
\bigwedge_{\mathbf{good}} \phi^{-1}\mathcal{L}_2 \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} &\prec_{\mathscr{T}_{\mathbf{good}}} \bigwedge_{\mathbf{good}} \mathcal{L}_{\mathbf{good}}\mathbf{u} \quad \text{[by (C11) and (C4)]} \\
\bigwedge_2 \phi\phi^{-1}\mathcal{L}_2 \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} &\prec_{2\varepsilon} \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \quad \text{[by (C9) and (C10)]} \\
\bigwedge_2 \mathcal{L}_2 \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} &\prec_{2\varepsilon} \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \\
\bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} &\prec_{2\varepsilon} \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \quad \text{[by (C3)]}
\end{aligned}$$

C.2 Proof of Assertion M

In view of (C7), M is equivalent to:

- If $\mathbf{x} = \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u}$ for some $\mathbf{u} \in \mathbf{Crit}(\mathscr{T}_{\mathbf{good}})$, and if $\mathbf{y} \in \mathbf{Crit}(\mathscr{T}_2)$ satisfies $\mathbf{y} \prec_{\mathscr{T}_2} \mathbf{x}$, then $\mathbf{y} \prec_{2\varepsilon} \mathbf{x}$.

To prove this, suppose $\mathbf{x} = \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u}$ for some $\mathbf{u} \in \mathbf{Crit}(\mathscr{T}_{\mathbf{good}})$, and $\mathbf{y} \in \mathbf{Crit}(\mathscr{T}_2)$ satisfies $\mathbf{y} \prec_{\mathscr{T}_2} \mathbf{x}$. Then we can successively deduce:

$$\begin{aligned}
\mathbf{y} &\prec_{\mathscr{T}_2} \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \quad \text{[because } \mathbf{y} \prec_{\mathscr{T}_2} \mathbf{x}\text{]} \\
\mathcal{L}_2\mathbf{y} &\supsetneq \mathcal{L}_2 \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \quad \text{[by (C6)]} \\
\mathcal{L}_2\mathbf{y} &\supsetneq \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \quad \text{[by (C2)]} \\
\phi^{-1}\mathcal{L}_2\mathbf{y} &\supsetneq \mathcal{L}_{\mathbf{good}}\mathbf{u} \qquad \text{(C12)} \\
\bigwedge_{\mathbf{good}} \phi^{-1}\mathcal{L}_2\mathbf{y} &\prec_{\mathscr{T}_{\mathbf{good}}} \bigwedge_{\mathbf{good}} \mathcal{L}_{\mathbf{good}}\mathbf{u} \quad \text{[by (C4)]} \\
\bigwedge_2 \phi\phi^{-1}\mathcal{L}_2\mathbf{y} &\prec_{2\varepsilon} \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \quad \text{[by (C9) and (C12)]} \\
\bigwedge_2 \mathcal{L}_2\mathbf{y} &\prec_{2\varepsilon} \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \\
\mathbf{y} &\prec_{2\varepsilon} \bigwedge_2 \phi\mathcal{L}_{\mathbf{good}}\mathbf{u} \quad \text{[by (C3)]}
\end{aligned}$$

This proves that $\mathbf{y} \prec_{2\varepsilon} \mathbf{x}$.

C.3 Proof of Assertion N

In view of (C7), $\bigwedge_2 \phi \mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x} \in \varphi[\mathbf{Crit}(\mathscr{T}_{\mathbf{good}})]$ for every node $\mathbf{x}$ of $\mathscr{T}_2$. So N can be proved by establishing that:

- For all $\mathbf{x} \in \mathbf{Crit}(\mathscr{T}_2)$, the node $\mathbf{z} = \bigwedge_2 \phi \mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x}$ satisfies $\mathbf{z} \preceq_{\mathscr{T}_2} \mathbf{x}$ and $\mathbf{x} \approx_{2\varepsilon} \mathbf{z}$.

To prove this, let $\mathbf{x} \in \mathbf{Crit}(\mathscr{T}_2)$ and let $\mathbf{z} = \bigwedge_2 \phi \mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x}$. Then we successively deduce:

$$\begin{aligned}
\mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x} &\supseteq \phi^{-1} \mathcal{L}_2 \mathbf{x} \quad \big[\text{by (C2)}\big] \\
V \phi \mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x} &\supseteq \phi \phi^{-1} \mathcal{L}_2 \mathbf{x} \\
\phi \mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x} &\supseteq \mathcal{L}_2 \mathbf{x} \\
\bigwedge_2 \phi \mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x} &\preceq_{\mathscr{T}_2} \bigwedge_2 \mathcal{L}_2 \mathbf{x} \quad \big[\text{by (C1)}\big] \\
\bigwedge_2 \phi \mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x} &\preceq_{\mathscr{T}_2} \mathbf{x} \quad \big[\text{by (C3)}\big]
\end{aligned} \tag{C13}$$

This proves that $\mathbf{z} \preceq_{\mathscr{T}_2} \mathbf{x}$. We can also successively deduce:

$$\begin{aligned}
\bigwedge_{\mathbf{good}} \mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x} &= \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x} \quad \big[\text{by (C3)}\big] \\
\bigwedge_2 \phi \mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x} &\approx_{2\varepsilon} \bigwedge_2 \phi \phi^{-1} \mathcal{L}_2 \mathbf{x} \quad \big[\text{by (C8) and (C13)}\big] \\
\bigwedge_2 \phi \mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x} &\approx_{2\varepsilon} \bigwedge_2 \mathcal{L}_2 \mathbf{x} \\
\bigwedge_2 \phi \mathcal{L}_{\mathbf{good}} \bigwedge_{\mathbf{good}} \phi^{-1} \mathcal{L}_2 \mathbf{x} &\approx_{2\varepsilon} \mathbf{x} \quad \big[\text{by (C3)}\big]
\end{aligned}$$

This proves that $\mathbf{z} \approx_{2\varepsilon} \mathbf{x}$.

C.4 Proof of Assertion O

Let $\mathbf{u}, \mathbf{u}' \in \mathbf{Crit}(\mathscr{T}_{\mathbf{good}})$. Then:

$$\begin{aligned}
\mathbf{u} \preceq_{\mathscr{T}_{\mathbf{good}}} \mathbf{u}' \quad & \text{just if} \quad \mathcal{L}_{\mathbf{good}} \mathbf{u} \supseteq \mathcal{L}_{\mathbf{good}} \mathbf{u}' && \big[\text{by (C5) and (C6)}\big] \\
& \text{just if} \quad \phi \mathcal{L}_{\mathbf{good}} \mathbf{u} \supseteq \phi \mathcal{L}_{\mathbf{good}} \mathbf{u}' \\
& \text{just if} \quad \mathcal{L}_2 \varphi(\mathbf{u}) \supseteq \mathcal{L}_2 \varphi\big(\mathbf{u}'\big) && \text{[by assertion L]} \\
& \text{just if} \quad \varphi(\mathbf{u}) \preceq_{\mathscr{T}_2} \varphi\big(\mathbf{u}'\big) && \big[\text{by (C5) and (C6)}\big]
\end{aligned}$$

References

1. Carr, H., Snoeyink, J., Axen, U.: Computing contour trees in all dimensions. Comput. Geom. **24**, 75–94 (2003)
2. Cohen-Steiner, D., Edelsbrunner, H., Harer, J.: Stability of persistence diagrams. Discrete Comput. Geom. **37**, 103–120 (2007)

3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
4. Edelsbrunner, H., Harer, J.: Computational Topology: An Introduction. Am. Math. Soc., Providence (2010)
5. Herman, G.T.: Geometry of Digital Spaces. Birkhäuser Boston, Boston (1998)
6. Matas, J., Chum, O., Urban, M., Pajdla, T.: Robust wide baseline stereo from maximally stable extremal regions. In: Rosin, P.L., Marshall, D. (eds.) Proceedings of the British Machine Vision Conference, BMVC 2002, pp. 384–393. British Machine Vision Association, Malvern (2002).
7. Najman, L., Couprie, M.: Building the component tree in quasi-linear time. IEEE Trans. Image Process. **15**, 3531–3539 (2006)
8. Russell, W.C.: Update on adenovirus and its vectors. J. Gen. Virol. **81**, 2573–2604 (2000)
9. San Martín, C., Glasgow, J.N., Borovjagin, A., Beatty, M.S., Kashentseva, E.A., Curiel, D.T., Marabini, R., Dmitriev, I.P.: Localization of the N-terminus of minor coat protein IIIa in the adenovirus capsid. J. Mol. Biol. **383**, 923–934 (2008)
10. Sarioz, D., Kong, T.Y., Herman, G.T.: History trees as descriptors of macromolecular structures. In: Bebis, G., et al. (eds.) Advances in Visual Computing: Second International Symposium, ISVC 2006, Proceedings, Part I, pp. 263–272. Springer, Berlin (2006)
11. Tarjan, R.E.: Efficiency of a good but not linear set union algorithm. J. Assoc. Comput. Mach. **22**, 215–225 (1975)
12. Weber, G.H., Dillard, S.E., Carr, H., Pascucci, V., Hamann, B.: Topology controlled volume rendering. IEEE Trans. Vis. Comput. Graph. **13**, 330–341 (2007)

Chapter 3
Discrete Topological Transformations for Image Processing

Michel Couprie and Gilles Bertrand

Abstract Topology-based image processing operators usually aim at transforming an image while preserving its topological characteristics. This chapter reviews some approaches which lead to efficient and exact algorithms for topological transformations in 2D, 3D and grayscale images. Some transformations that modify topology in a controlled manner are also described. Finally, based on the framework of critical kernels, we show how to design a topologically sound parallel thinning algorithm guided by a priority function.

3.1 Introduction

Topology-preserving operators, such as homotopic thinning and skeletonization, are used in many applications of image analysis to transform an object while leaving unchanged its topological characteristics. In particular, skeletons are often used as a simplification of the original data, which facilitates shape recognition, registration or animation.

In this chapter, we will see how to define and efficiently implement such operators, on the basis of elementary topology-preserving transformations. We will also discuss some geometrical aspects of skeletons, as well as the need for filtering them. Besides, we will see that it is sometimes interesting to be able to selectively modify topology: we will present, in particular, a method that suppresses holes (or tunnels) in 3D images, depending on a "size" criterion.

These transformations are usually defined for acting on binary images (*i.e.*, pixel or voxel sets). In Sect. 3.2, we will extend them to the case of grayscale images (*i.e.*, functions), and present some applications to image filtering, segmentation and restoration.

M. Couprie (✉) · G. Bertrand
Laboratoire d'Informatique Gaspard-Monge, Équipe A3SI, Université Paris-Est, ESIEE Paris, Marne-la-Vallée, France
e-mail: m.couprie@esiee.fr

G. Bertrand
e-mail: g.bertrand@esiee.fr

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_3,

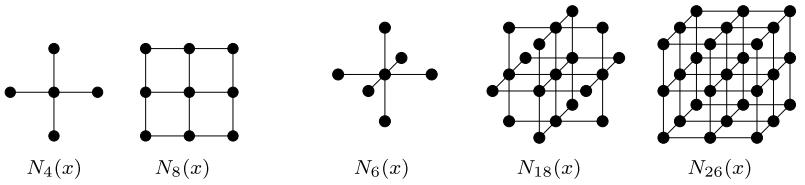


Fig. 3.1 Different neighborhoods of a point x (the *central point*) in 2D and in 3D

There are two main kinds of thinning procedures: the sequential ones, that take a single point in consideration at each step, and the parallel ones, that attempt at removing a whole set of points at each iteration. In the first case, the result most often depends on the order in which the points are considered, while the latter kind permits to provide a well-defined result, which is generally more robust than the former to noise and orientation changes. The third part of this chapter deals with parallel thinning: we present the framework of critical kernels, that provides a mean to guarantee the topological soundness of parallel homotopic transformations. We introduce in this framework a new algorithm that builds at once a well-defined family of filtered Euclidean skeletons.

3.2 Topological Transformations of Binary Images

3.2.1 Neighborhoods, Connectedness

First of all, let us recall the basic definitions of digital topology [29, 34] that will be used in this chapter.

A point $x \in \mathbb{Z}^D$ $(D = 2, 3)$ is defined by $(x_1, \ldots, x_D)$ with $x_i \in \mathbb{Z}$. We consider the neighborhood relations N_4 and N_8 defined for any point $x \in \mathbb{Z}^2$ by:

$$N_4(x) = \big\{y \in \mathbb{Z}^2; |y_1 - x_1| + |y_2 - x_2| \leqslant 1\big\},$$
$$N_8(x) = \big\{y \in \mathbb{Z}^2; \max\big(|y_1 - x_1|, |y_2 - x_2|\big) \leqslant 1\big\},$$

and the neighborhood relations N_6, N_{26} and N_{18} defined for any point $x \in \mathbb{Z}^3$ by:

$$N_6(x) = \big\{y \in \mathbb{Z}^3; |y_1 - x_1| + |y_2 - x_2| + |y_3 - x_3| \leqslant 1\big\},$$
$$N_{26}(x) = \big\{y \in \mathbb{Z}^3; \max\big(|y_1 - x_1|, |y_2 - x_2|, |y_3 - x_3|\big) \leqslant 1\big\},$$
$$N_{18}(x) = \big\{y \in N_{26}(x); |y_1 - x_1| + |y_2 - x_2| + |y_3 - x_3| \leqslant 2\big\}.$$

These neighborhoods are illustrated in Fig. 3.1.

We denote by E the set $\mathbb{Z}^2$ or $\mathbb{Z}^3$. In the sequel, we denote by n a number such that $n \in \{4, 8, 6, 26\}$. We define $N_n^*(x) = N_n(x) \setminus \{x\}$. The point $y \in E$ is said to be *n-adjacent* to the point $x \in E$ if $y \in N_n^*(x)$. An *n-path* is an ordered sequence of points $x_0 \ldots x_k$ such that x_i is n-adjacent to x_{i-1} for any $i \in \{1, \ldots, k\}$.

Let $X \subseteq E$, we say that two points x, y of X are *n-connected in X* if there exists an n-path in X between those two points. This defines an equivalence relation on X.

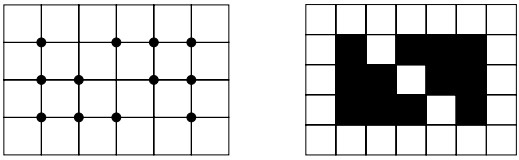


Fig. 3.2 The set of black points has two 4-connected components, and only one 8-connected component. This figure also illustrates two common representations of a binary digital image (points on the *left*, pixels on the *right*)

The equivalence classes for this relation are the *n-connected components of X* (see Fig. 3.2). A subset X of E is said to be *n-connected* if it is composed of exactly one n-connected component.

The set composed of all n-connected components of X is denoted by $C_n(X)$. A subset Y of E is said to be *n-adjacent* to a point $x \in E$ if there exists a point $y \in Y$ that is n-adjacent to x. The set of all n-connected components of X that are n-adjacent to x is denoted by $C_n^x(X)$. Remark that $C_n(X)$ and $C_n^x(X)$ are sets of subsets of X, and not sets of points. Besides, if S is a finite set, we denote by $|S|$ the number of elements of S.

3.2.2 Connectivity Numbers

Intuitively, a point x of an object $X \subseteq E$ is said to be simple if it can be deleted from X while preserving the topological characteristics of X (see [19]). In the case of $\mathbb{Z}^2$, this implies preserving the number of connected components of both the object and its complementary set. In $\mathbb{Z}^3$, it is necessary to preserve also holes (or tunnels), a notion that may be formalized through the fundamental group (see *e.g.* [24]).

Note that the definition of a simple point relies on notions (connected components, tunnels) that can be classified as global, in the sense that they cannot be defined without taking the whole object into account. Nevertheless, we will see that in 2D and 3D, it is possible to characterize simple points on a local basis, thanks to the connectivity numbers introduced in this section. Such a local characterization is essential to get efficient algorithms for topological transformations.

Let X be a finite subset of $\mathbb{Z}^D$. We denote by $\overline{X}$ the complement set of X, *i.e.*, $\overline{X} = \mathbb{Z}^D \setminus X$. If we use a n-connectivity for X then we have to use a $\overline{n}$-connectivity for $\overline{X}$. For example in 2D the 4-connectivity for X is associated with the 8-connectivity for $\overline{X}$, and in 3D the 6-connectivity for X is associated with the 26-connectivity for $\overline{X}$. This is necessary to have a correspondence between topological characteristics of X and $\overline{X}$ (see *e.g.* [29]). To summarize, we have the following possibilities in 2D: $(n, \overline{n}) = (4, 8)$ or $(8, 4)$; and in 3D[1]: $(n, \overline{n}) = (6, 26)$ or $(26, 6)$.

[1]For the sake of simplicity we do not discuss here the case of the 18-connectivity, see [3, 10, 32] for more information.

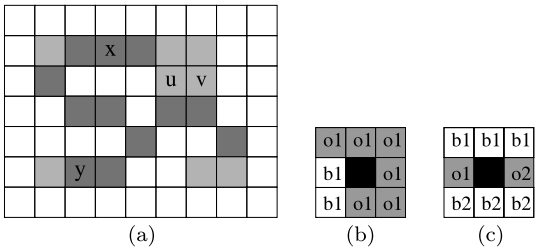


Fig. 3.3 We set $(n, \overline{n}) = (8, 4)$. (**a**): An object X (*light gray* and *dark gray* pixels). (**b**): The eight neighbors of pixel u. The unique 8-connected component of $N_8^*(u) \cap X$ is labeled with $o1$, and the unique 4-connected component of $N_8^*(u) \cap \overline{X}$ is labeled with $b1$. (**c**): Depicts the eight neighbors of pixel x or pixel y. The 8-connected components of $N_8^*(x) \cap X$ are labeled with $o1$, $o2$, and the 4-connected components of $N_8^*(x) \cap \overline{X}$ are labeled with $b1$, $b2$

Now, we can define the connectivity numbers in 2D and in 3D [3]. Intuitively, the connectivity number of a point x relative to a set X, counts the number of connected components of $X \setminus \{x\}$, which are in the neighborhood of x, and which are adjacent to x.

Definition 1 Let $X \subseteq \mathbb{Z}^2$ and $x \in \mathbb{Z}^2$. Let $n \in \{4, 8\}$. The (2D) *connectivity numbers* are defined as follows:

$$T_4(x, X) = \left|C_4^x\left[N_8^*(x) \cap X\right]\right|,$$
$$T_8(x, X) = \left|C_8^x\left[N_8^*(x) \cap X\right]\right|.$$

In Fig. 3.3, we illustrate some connectivity numbers in 2D. Figure 3.3b shows the neighborhood of point u, we can verify that $T_8(u, X) = 1$ and $T_4(u, \overline{X}) = 1$. Similarly, the reader can check that $T_8(v, X) = T_4(v, \overline{X}) = 1$. For pixel x, we have $T_8(x, X) = T_4(x, \overline{X}) = 2$ (see Fig. 3.3c). The same holds for pixel y.

Definition 2 Let $X \subseteq \mathbb{Z}^3$ and $x \in \mathbb{Z}^3$. The (3D) *connectivity numbers* are defined as follows:

$$T_6(x, X) = \left|C_6^x\left[N_{18}^*(x) \cap X\right]\right|,$$
$$T_{26}(x, X) = \left|C_{26}^x\left[N_{26}^*(x) \cap X\right]\right|.$$

Figure 3.4 shows some examples that illustrate this definition. Note that components that are not adjacent to the central point, according to the chosen adjacency relation, are not taken into account: this is illustrated in Fig. 3.4b.

3.2.3 Topological Classification of Object Points

If we use the n-connectivity for X and the $\overline{n}$-connectivity for $\overline{X}$, the numbers $T_n(x, X)$ and $T_{\overline{n}}(x, \overline{X})$ give us topological characteristics of the point x in the ob-

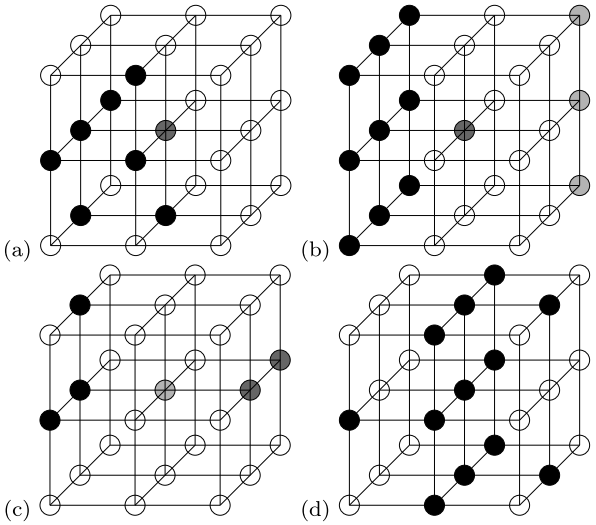


Fig. 3.4 (**a**): The central point x is a 6-simple point ($T_6(x, X) = T_{26}(x, \overline{X}) = 1$): the unique "object" component in its neighborhood is in *black*, and the unique "background" component is in *white*. We have also $T_{26}(x, X) = T_6(x, \overline{X}) = 1$, hence x is 26-simple. (**b**): The central point x is a 6-simple point ($T_6(x, X) = T_{26}(x, \overline{X}) = 1$): there are two "object" components in its neighborhood, but only the one in *black* is 6-adjacent to x. However, x is not 26-simple, for $T_{26}(x, X) = 2$. (**c**): The central point x is such that $T_6(x, X) = 2$ and $T_{26}(x, \overline{X}) = 1$; the two "object" components are in *black* and *dark gray*. (**d**): The central point x is such that $T_6(x, X) = 1$ and $T_{26}(x, \overline{X}) = 2$

ject X. In particular, the connectivity numbers allow us to detect whether a point is simple or not [3, 10], both in 2D and in 3D:

Theorem 1 *Let $X \subseteq E$ and $x \in X$. The point x is n-simple if and only if $T_n(x, X) = 1$ and $T_{\overline{n}}(x, \overline{X}) = 1$.*

Intuitively, this characterization states that a point is simple if and only if there is, in its neighborhood, exactly one "object" component and one "background" component. For example, in Fig. 3.3a, we conclude from the computation of connectivity numbers that points u, v are both simple, whereas x, y are both non-simple points. In this figure, all simple points are in lighter gray.

Note that the neighborhoods of points x and y are the same (Fig. 3.3c), hence also the connectivity numbers, but different events occur whenever x or y is deleted from X. In the case of x, two background components are merged; whereas if y disappears, X is split into two components. Note also that any simple point may be removed from X without altering topology, but removing simultaneously u and v for instance would change topological characteristics of the image (here, the number of background components). We will see in Sect. 3.4.5 how to perform parallel thinning with topological guarantees.

The characterization of Theorem 1 also holds in the 3D case, see the examples of Fig. 3.4.

The fact that an intrinsically global notion—the one of simple point—admits a local characterization is indeed a quite remarkable property. It will allow us to efficiently implement topological transformations.

The connectivity numbers are also useful to detect other kinds of points of particular interest. A point x such that $T_n(x, X) = 0$ is an *isolated point*. If $T_{\overline{n}}(x, \overline{X}) = 0$, then we have an *interior point*. The *border points* are characterized by $T_{\overline{n}}(x, \overline{X}) \neq 0$.

Let us consider the case where $E = \mathbb{Z}^3$, and take a point x such that $T_n(x, X) \geqslant 2$. If we delete x from X, we locally disconnect the object X (see Fig. 3.4b). We say that such a point is a *1D isthmus*.

Consider the simplest case where $T_n(x, X) = 2$ (see an example in Fig. 3.4c). Two situations may occur whenever x is deleted. In the first case, the two local components involved in the definition of $T_n(x, X)$ are in fact connected together by a path in X outside the neighborhood of x, and the deletion of the latter suppresses a tunnel from the object (this situation is similar to the one of point x in Fig. 3.3a, in 2D). In the second case, the two local components are not connected and the deletion of x indeed disconnects the object (see y in Fig. 3.3a for a similar 2D situation). In both cases, topology is not preserved, in other words the point x is not simple.

In the same way, a point x such that $T_{\overline{n}}(x, \overline{X}) \geqslant 2$ is called a *2D isthmus*; its deletion causes the merging of connected components of the neighborhood of x in $\overline{X}$ (see Fig. 3.4d). If these components are connected together in $\overline{X}$, the deletion of x creates a new tunnel for the object, and if they are not, the deletion of x causes decrease of the number of cavities. Also here, the point x is non-simple.

3.2.4 Topology-Preserving Transformations

Deleting a simple point from an object X yields an object Y included in X, which is "topologically equivalent" to X. If we iterate this elementary operation, we can obtain a family of nested sets that are all topologically equivalent to X. More formally, we say that Y *is an elementary homotopic thinning of* X, and we write $X \xrightarrow{e} Y$, if there exists a simple point x for X such that $Y = X \setminus \{x\}$. We say that Y *is a homotopic thinning of* X if $Y = X$ or if there exists a sequence $\langle Y_0, \ldots, Y_k \rangle$ such that $Y_0 = X$, $Y_k = Y$ and $Y_0 \xrightarrow{e} \ldots \xrightarrow{e} Y_k$. If, furthermore, no point in Y is simple, we say that Y *is an ultimate homotopic thinning of* X.

When transforming an object X through an homotopic thinning, it is often needed to preserve from deletion a given subset K of X. Such a subset is called a *constraint set*, and we say that Y *is an homotopic thinning of* X *constrained by* K if Y is an homotopic thinning of X such that $K \subseteq Y$. If, furthermore, no point of $Y \setminus K$ is simple, we say that Y *is an ultimate homotopic thinning of* X *constrained by* K.

In order to thicken an object X in a topology-preserving manner, it is sufficient to compute an homotopic thinning of the complementary set of X (for the dual connectivity), and to take the complementary of the result.

3.2.5 Transformations Guided by a Priority Function

The order in which points are considered during a thinning process plays, of course, an important role with respect to the geometrical aspect of the result. This order can be specified by means of a numerical function, called priority function.

With each point x of X, a priority function associates an integer or real number $P(x)$, which represents the priority of point x. The points of X will be treated during the thinning process following the increasing values of P. To certain points x, a value $P(x) = +\infty$ may be given, meaning that these points must be preserved; in other words, the points with infinite priority constitute the constraint set.

This strategy is realized by the Algorithm *GuidedThinning* (Algorithm 1). The complexity of this algorithm is determined by the choice of the data structure used to represent the function P. For example, a balanced search tree allows for reaching a global time complexity in $O(n \log n)$, where n is the number of image points. In certain particular cases, including the very common case where the function P is a distance map [35], it is possible to implement Algorithm *GuidedThinning* in linear time (see [1]).

Algorithm 1: GuidedThinning

Data : $X \subseteq E$, a function P from X in $\mathbb{Z} \cup \{+\infty\}$ or $\mathbb{R} \cup \{+\infty\}$
Result : X
repeat
  Let x be a point in X such that x is simple for X, $P(x) < +\infty$, and $P(x)$ is minimal;
  $X = X \setminus \{x\}$;
until *stability* ;

If one wants to use Algorithm *GuidedThinning* for skeletonization purposes, a natural choice for the priority function is a distance map relative to the background. In other words, the points with highest priority (*i.e.*, smallest value) are those closest to the background, and the points that "survive" are well centered in the object, in the sense that their distance to the background is, roughly speaking, maximal. Note that any distance may be chosen: discrete distances [35], chamfer distances [13], Euclidean distance [23], etc. The choice of the Euclidean distance permits to obtain the lowest sensibility to rotations.

However, choosing the exact Euclidean distance map as a priority function for removing simple points from the object may lead to geometric distortions [39]. To illustrate this point, let us consider the object X depicted in white in Fig. 3.5a. In Fig. 3.5b, we show in black superimposed to X, all the centers of maximal included Euclidean balls (that is, balls that are included in X but that are not subsets of any other ball included in X). This is one of the possible definitions for the *medial axis* of X (see also Sect. 3.2.6). It is usual to take only a subset of the medial axis as constraint set for computing centered skeletons, since the full medial axis often contains

Fig. 3.5 (**a**): The original object X (in *white*). (**b**): The Euclidean medial axis of X (centers of maximal balls, see text), superimposed to X. (**c**): A subset Y of the medial axis. (**d**): Result of the skeletonization using the Euclidean distance map as a priority function, and Y as constraint set

spurious points. Such a constraint set, let us call it Y, is depicted in Fig. 3.5c, superimposed to X. We use as priority function the map P defined by

$$P(x) = \begin{cases} +\infty & \text{whenever } x \in Y; \\ d(x, \overline{X}) & \text{otherwise} \end{cases}$$

where $d(x, \overline{X}) = \min\{d(x, y) \mid y \in \overline{X}\}$, and $d(x, y)$ denotes the Euclidean distance between x and y. When $Y = \emptyset$, the function P is just the distance map relative to X.

Figure 3.5d depicts the result of Algorithm *GuidedThinning* in this case. Note that the obtained skeleton deviates from the medial axis points.

In the next section, we will study another priority function that gives better results than the Euclidean distance map, and is linked to a family of filtered medial axes.

3.2.6 Lambda-Medial Axis

The notion of medial axis has been introduced by Blum in the 60s [11, 12]. In the continuous Euclidean space, the following definition can be used to formalize this notion: let X be a bounded subset of $\mathbb{R}^D$, the medial axis of X consists of the points $x \in X$ that have more than one nearest points on the boundary of X.

A major difficulty when using the medial axis in applications (*e.g.*, shape recognition), is its sensitivity to small contour perturbations, in other words, its lack of stability. A recent survey [2] summarizes selected relevant studies dealing with this topic. Because of this problem, it is usually necessary to add a filtering step (or pruning step) to any method that aims at computing the medial axis.

In 2005, F. Chazal and A. Lieutier introduced the λ-medial axis [16], a particular class of filtered skeletons, and studied its properties, in particular those related to stability. A major outcome of [16] is the following property: informally, except for particular values of the filtering parameter, the λ-medial axis remains stable under perturbations of the shape that are small with regard to the Hausdorff distance.

The original definition of the λ-medial axis (see [16]) holds and makes sense in the (continuous) Euclidean D-dimensional space.

Let $x = (x_1, \ldots, x_D)$, $y = (y_1, \ldots, y_D) \in \mathbb{R}^D$, we denote by $d(x, y)$ the Euclidean distance between x and y, in other words, $d(x, y) = (\sum_{k=1}^{D} (y_k - x_k)^2)^{\frac{1}{2}}$. Let $S \subseteq \mathbb{R}^D$, we set $d(y, S) = \min_{x \in S}\{d(y, x)\}$.

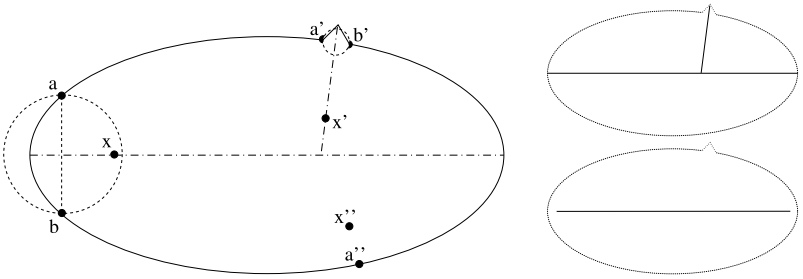


Fig. 3.6 Illustration of the λ-medial axis in $\mathbb{R}^2$. *Left*: Points x, x' and x'' and their respective closest boundary points. *Top right*: λ-medial axis with $\lambda = \epsilon$, a very small positive real number. *Bottom right*: λ-medial axis with $\lambda = d(a', b') + \epsilon$

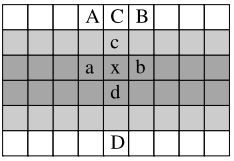


Fig. 3.7 We consider an object X in $\mathbb{Z}^2$ that is a horizontal ribbon of infinite length and width 4 (partially depicted here in *gray*). The projection of x on $\overline{X}$ is $\Pi_{\overline{X}}(x) = \{C\}$. The smallest ball that includes $\Pi_{\overline{X}}(x)$ is the one with center C and radius 0. The projections of a, b, c, d on $\overline{X}$ are respectively $\{A\}, \{B\}, \{C\}, \{D\}$. Hence, the extended projection of x on $\overline{X}$ is $\Pi^e_{\overline{X}}(x) = \{A, B, C, D\}$, and we have $PR_X(x) = R > 2$. The pixels in *darker gray* are in any λ-medial axis with $\lambda \leqslant R$, those in *lighter gray* are only in the 0-medial axis of X

Let $x \in \mathbb{R}^D$, $r \in \mathbb{R}$, $r \geqslant 0$, we denote by $B_r(x)$ the *ball of radius r centered on x*, defined by $B_r(x) = \{y \in \mathbb{R}^D \mid d(x, y) \leqslant r\}$.

Let S be a nonempty subset of $\mathbb{R}^D$, and let $x \in \mathbb{R}^D$. The *projection of x on S*, denoted by $\Pi_S(x)$, is the set of points y of S that are at a minimal distance from x; more precisely,

$$\Pi_S(x) = \left\{ y \in S \;\middle|\; \forall z \in S, d(y, x) \leqslant d(z, x) \right\}.$$

The *λ-medial axis of X* is the set of points x in X such that the radius of the smallest ball that includes $\Pi_{\overline{X}}(x)$ is not less than λ. For example in Fig. 3.6, we show a shape that is an ellipsis with a small "bump", and we consider the interior X of this shape. Two different λ-medial axes of X are displayed on the right.

Now, let us consider the discrete case. For each point $x \in \mathbb{Z}^D$, we define the *direct neighborhood of x* as $N(x) = \{y \in \mathbb{Z}^D \mid d(x, y) \leqslant 1\}$. Thus, $N(x) = N_4(x)$ (resp. $N_6(x)$) whenever $D = 2$ (resp. $D = 3$).

Transposing directly the definition of the λ-medial axis to the discrete grid $\mathbb{Z}^D$ would yield unsatisfactory results. For instance, consider a horizontal ribbon in $\mathbb{Z}^2$ with constant, even width and infinite length (see Fig. 3.7). Clearly, the projection of any point of this set on its complementary set is reduced to a singleton. If we keep

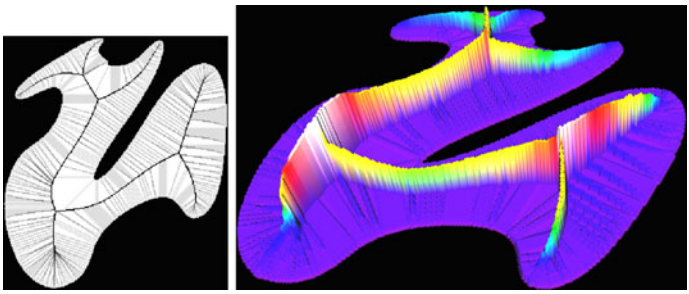


Fig. 3.8 (**a**): The function PR_X superimposed to the shape X. *Darkest gray* levels represent highest values of $PR_X(x)$. (**b**): A 3D representation of the function PR_X

the same definition as above, any λ-medial axis of this object with $\lambda > 0$ would be empty.

This is why we need the following notion. Let $X \subseteq \mathbb{Z}^D$, and let $x \in X$. The *extended projection of x on $\overline{X}$* (where $\overline{X} = \mathbb{Z}^D \setminus X$), denoted by $\Pi^e_{\overline{X}}(x)$, is the union of the sets $\Pi_{\overline{X}}(y)$, for all y in $N(x)$ such that $d(y, \overline{X}) \leqslant d(x, \overline{X})$. Figure 3.7 illustrates this notion and the following ones.

Let X be a finite subset of $\mathbb{Z}^D$, and let $\lambda \in \mathbb{R}$, $\lambda \geqslant 0$. We define the function PR_X that associates, to each point x of X, the value $PR_X(x)$ that is the radius of the smallest ball enclosing all the points of the extended projection of x on $\overline{X}$. In other terms, $PR_X(x) = \min\{r \in \mathbb{R}, r \geqslant 0 \mid \exists y \in \mathbb{R}^D, \Pi^e_{\overline{X}}(x) \subseteq B_r(y)\}$, and we call $PR_X(x)$ the *projection radius of x (for X)*.

The following definition was introduced in [15], together with an experimental evaluation of the stability and rotation invariance of the discrete λ-medial axis.

Definition 3 ([15]) The *discrete λ-medial axis of X*, denoted by $DLMA(X, \lambda)$, is the set of points x in X such that $PR_X(x) \geqslant \lambda$.

Note that the function PR_X can be computed once and stored as a grayscale image, and that any DLMA of X is a level set of this function at a particular value λ (see Fig. 3.8 and Fig. 3.9). For more details, illustrations and performance analysis, see [15].

The illustration in Fig. 3.9b is sufficient to demonstrate that a DLMA of a given shape X may have a homotopy type different from the one of X.

The Algorithm *GuidedThinning*, with PR_X as priority function and with a DLMA of X as constraint set, provides filtered skeletons that are homotopic to X and share the good geometric properties of the DLMAs (see Fig. 3.9c).

Another example is shown in Fig. 3.10, where a filtered Euclidean medial axis is used as a constraint set during skeletonization. As we have seen at the end of Sect. 3.2.5 (see also [39]), choosing the exact Euclidean distance map as a priority function for removing simple points from the object may lead to geometric distortions. In some cases, "extra branches" may even appear (see Fig. 3.5d). Choosing

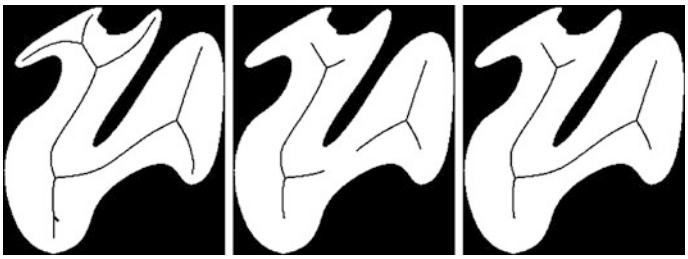


Fig. 3.9 Any DLMA of X is a threshold of PR_X at a particular value λ. (**a**): Discrete 7-medial axis. (**b**): Discrete 25-medial axis of X. (**c**): Guided homotopic thinning of X, with PR_X as priority function and with (**b**) as constraint set

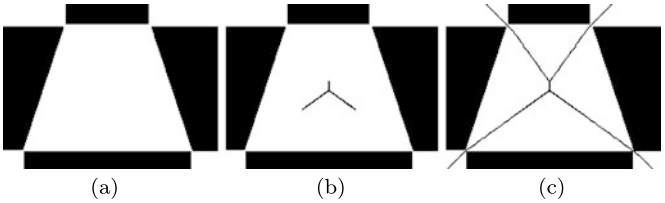


Fig. 3.10 (**a**): The original object X (in *white*, the same as Fig. 3.5a). (**b**): A constraint set Y: a filtered DLMA, that is also a set of centers of maximal included balls (see Fig. 3.5c). (**c**): Result of the skeletonization using PR_X as a priority function, and Y as constraint set

the map PR_X as priority function yields more satisfying results (see Fig. 3.10c), as it guides the thinning process towards elements that belong to the different nested discrete λ-medial axes.

3.2.7 Other Applications of Guided Thinning

For certain applications, it may be relevant to take as priority function the gray levels of an image. This makes sense when these gray levels can be interpreted as a measure of the likelihood, for a pixel, to belong to a certain class or region.

To illustrate this, suppose that we want to extract from a 3D magnetic resonance image (MRI) of the head, the white matter of the brain (see Fig. 3.11a). From the knowledge of human anatomy and the parameters of the imagery device, we know that a volume element x situated in the white matter produces a response that is coded by a value $F(x)$ for the corresponding voxel, which lies between two limits $\mu_1 < \mu_2$. Assuming a Gaussian model, the voxels with value $\frac{\mu_1+\mu_2}{2}$ are those with highest probability to belong to the white matter. Furthermore, we know from anatomical data that the white matter of the brain constitutes a simply connected volume, in other words, it is topologically equivalent to a ball. In order to guarantee a result having the wanted topological characteristics, we use the following scheme:

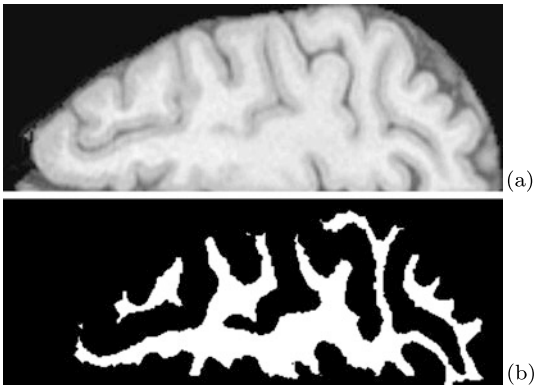


Fig. 3.11 (**a**): Detail of a 2D plane from a 3D MRI of the brain. (**b**): Result of the method described in the text. Note that the result is connected in 3D, although the shown 2D cross-section is not connected

start with an object $X = \{x_0\}$, where x_0 is any point situated within the white matter; then perform an homotopic thinning of $\overline{X}$ (*i.e.* an homotopic thickening of X) guided with the priority function P defined by:

$$P(x) = \begin{cases} |F(x) - \mu|, \text{ where } \mu = \frac{\mu_1+\mu_2}{2} & \text{if } \mu_1 \leqslant F(x) \leqslant \mu_2, \\ +\infty & \text{otherwise.} \end{cases}$$

The values $+\infty$ ensure that all the resulting points have, in the image F, values that lie in the correct range (see Fig. 3.11b). This method has been successfully exploited to segment the white matter, as well as the cortex, from 3D MRI with topological guarantees [21, 22, 36].

In this kind of application, it is useful to be able to apply morphological filtering operators (openings, closings, alternate sequential filters) on an object while guaranteeing topology preservation. See [18] for the definition of such filtering operators.

3.2.8 Hole Closing

We have seen that it is possible, thanks to the notion of simple point, to design operators that transform an object while preserving its topological characteristics. However, controlled topology modifications are needed in some applications. This topic is seldom addressed in the literature. In this section, we present a method [1] that is, to our knowledge, the first one that permits to close holes in a 3D object.

In our approach, we consider the notion of hole from a topological point of view. From this viewpoint, it is important to distinguish between holes, cavities and concavities. A *concavity* is a concave part of the contour of an object, it is not a topological feature. A *cavity* is a bounded connected component of the background, that forms a "hollow" inside the object (see Fig. 3.12a).

A *hole* is much more complicated to define. Intuitively, the presence of a hole (or tunnel in 3D) in an object can be characterized by the existence of a closed path

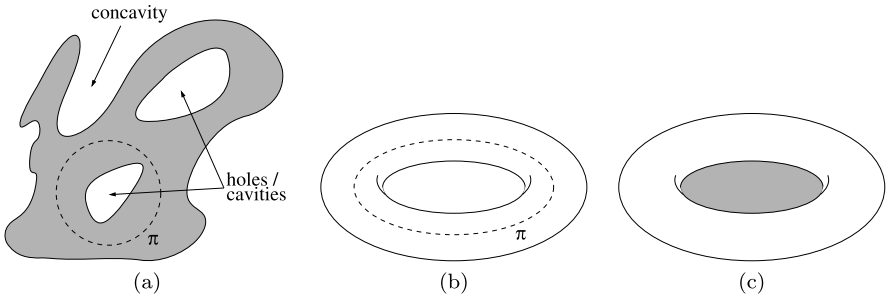


Fig. 3.12 (**a**): A 2D objects with two holes. (**b**): A solid torus. This object has one hole (tunnel), which is detected by the existence of path π. (**c**): The hole of the torus has been closed by a surface patch

in the object that cannot be continuously deformed, inside the object, into a single point. For example in 3D, a solid torus like the one depicted in Fig. 3.12b has one hole.

In 2D, the notions of hole and cavity coincide, thus closing holes in 2D may be simply done by using algorithms for connected component extraction. But closing holes in 3D objects is by no means a trivial problem, because 3D holes are not, like in 2D, delimited regions of space.

Based on connectivity numbers (Sect. 3.2.2) and the strategy of guided thinning (Sect. 3.2.5), the method that we present here closes holes in any 3D object (see Fig. 3.13). In addition, this method allows for controlling the "size" of the holes that are to be closed (Fig. 3.13b_2,b_3). It can be implemented by a linear-time algorithm.

The basic idea of this method consists of embedding the object X, in which we want to close holes, into another object Y that is connected, without any hole and without any cavity, such as a solid cuboid for example. Then, we iteratively shrink Y by deleting points that do not belong to X, and ensuring thanks to the analysis of connectivity numbers that each point deletion does not create any hole or cavity. This method has been introduced and formalized in [1], we recall here its main notions and properties.

Definition 4 ([1]) Let X, Y be such that $X \subseteq Y \subseteq \mathbb{Z}^3$. We say that Y *is a topological hull of* X if Y has no hole and no cavity, and if, for all $x \in Y \setminus X$, the set $Y \setminus \{x\}$ has a hole or a cavity.

For example in Fig. 3.13, $Y = (a_2)$ is a topological hull of $X = (a_1)$. The set $Y \setminus X$ (depicted by gray voxels in a_2) corresponds to "surface patches" that close the holes.

The following theorem allows for a local characterization of the class of sets that are topological hulls, relatively to the class of sets that have no cavity and no hole.

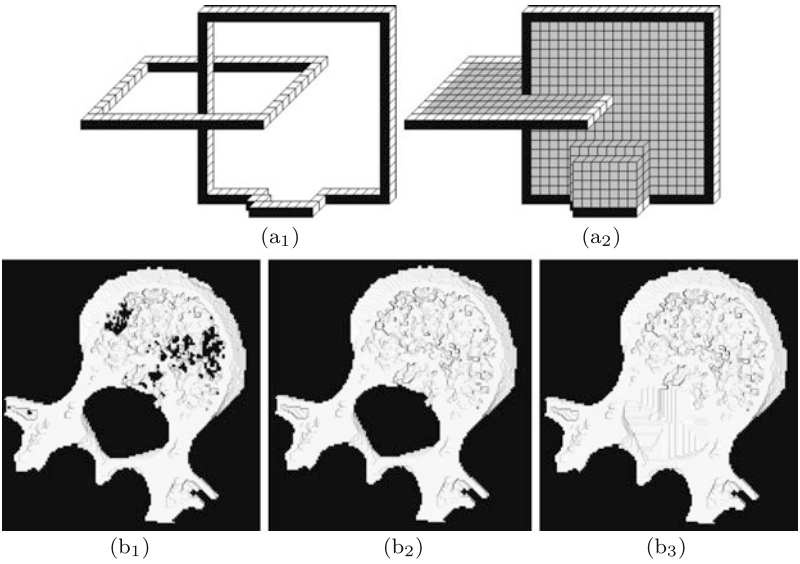


Fig. 3.13 Illustration of a hole closing algorithm for 3D objects. ($\mathbf{a_1}$, $\mathbf{a_2}$): The use of a distance map leads to a good centering of the surface patch that closes the hole. ($\mathbf{b_1}$, $\mathbf{b_2}$, $\mathbf{b_3}$): A parameter controls the "size" of the holes to be closed

Theorem 2 ([1]) *Let X, Y be such that $X \subseteq Y \subseteq \mathbb{Z}^3$. Suppose that Y has no cavity and no hole. Then, Y is a topological hull of X if and only if, for each point x of $Y \setminus X$, x is an interior point or a 2D isthmus for Y.*

Corollary 1 *Let X, Y, Z be such that $X \subseteq Y \subseteq Z \subseteq \mathbb{Z}^3$, and such that Z has no cavity and no hole. If Y can be obtained from Z by iterating the following two steps until stability*:

- *choose a point x in $Z \setminus X$ such that $T_{\overline{n}}(x, \overline{Z}) = 1$;*
- *set $Z = Z \setminus \{x\}$*

then Y is a topological hull of X.

In order to get a result that is well-centered with respect to the object X, we use a distance map to guide this process, in the manner of Algorithm *GuidedThinning*. More precisely, the points in the complement of X that are farthest from X are treated in the first place. We can also use a parameter s that allows for controlling the "size" of holes to be closed: if one also deletes, during the process, the candidate points x that are such that $T_{\overline{n}}(x, \overline{X}) > 1$, and having a distance map value greater than s, then the biggest holes (in this sense) will be let open. The Algorithm *HoleClosing* (Algorithm 2) formalizes this method. As for Algorithm *GuidedThinning*, with an adapted choice of data structure this algorithm may be implemented to run in linear time. Note that, whenever the parameter s is set to $+\infty$, Algorithm *HoleClosing* indeed computes a topological hull of X (in other words, it closes all holes).

Algorithm 2: HoleClosing

Data : $X \subseteq \mathbb{Z}^3$ (the object), $s \in \mathbb{R} \cup \{+\infty\}$ (the size parameter)
Result : Z
Let Z be a cuboid that includes X;
Let P be a distance map relative to X (*i.e.*, $P(x) = d(x, X)$ for any x);
repeat
  $Z' = \{z \in Z \setminus X \mid T_{\overline{n}}(z, \overline{Z}) = 1$ or $(T_{\overline{n}}(x, \overline{X}) > 1$ and $P(z) > s)\}$;
  Let x be a point in Z' such that $P(x)$ is maximal;
  $Z = Z \setminus \{x\}$;
until *stability* ;

3.3 Topological Transformations for Grayscale Images

In this section topological notions such as those of simple point, homotopic thinning, ultimate homotopic thinning, are extended to the case of grayscale images. Applications to image filtering, segmentation and restoration are presented.

A 2D grayscale image can be seen as a function F from $\mathbb{Z}^2$ into $\mathbb{Z}$. For each point x of $\mathbb{Z}^2$, $F(x)$ is the gray level, or the luminosity of x. We denote by $\mathcal{F}$ the set of all functions from $\mathbb{Z}^2$ into $\mathbb{Z}$.

Let $F \in \mathcal{F}$ and $k \in \mathbb{Z}$, the *cross-section (or threshold) of F at level k* is the set F_k composed of all points $x \in \mathbb{Z}^2$ such that $F(x) \geqslant k$. Observe that a cross-section is a set of points, *i.e.*, a binary image. As for the binary case, if we use the n-adjacency for the cross-sections F_k of F, we must use the $\overline{n}$-adjacency for the complementary sets $\overline{F_k}$, with $(n, \overline{n}) = (8, 4)$ or $(4, 8)$. Consider the function $-F$, that we call the *complementary function* of F (for each point x of $\mathbb{Z}^2$, $(-F)(x) = -F(x)$). Note that the complementary sets of the cross-sections of F are cross-sections of $-F$. In forthcoming examples and figures, we choose $n = 8$ for the cross-sections of F, thus we must use $\overline{n} = 4$ for the cross-sections of $-F$. A non-empty connected component X of a cross-section F_k of F is a *(regional) maximum for F* if $X \cap F_{k+1} = \emptyset$. A set $X \subseteq \mathbb{Z}^2$ is a *(regional) minimum for F* if it is a regional maximum for $-F$.

3.3.1 Cross-Section Topology

Intuitively, we say that a transformation of F preserves topology if the topology of all cross-sections of F is preserved. Hence, the "cross-section topology" of a function (*i.e.*, of a grayscale image) directly derives from the topology of binary images [9]. Based on this idea, the following notions generalize the notion of simple point to the case of functions.

Definition 5 Let $F \in \mathcal{F}$, the point $x \in \mathbb{Z}^2$ is *destructible (for F)* if x is simple for F_k, with $k = F(x)$. The point $x \in \mathbb{Z}^2$ is *constructible (for F)* if x is destructible for $-F$.

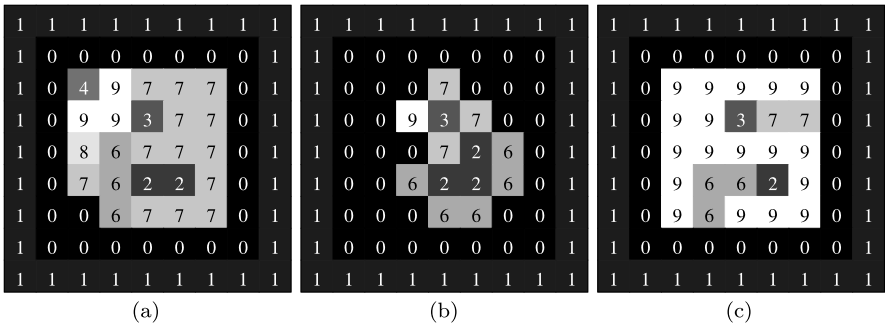


Fig. 3.14 (**a**): Original image. (**b**): An ultimate homotopic thinning of (**a**). (**c**): An ultimate homotopic thickening of (**a**)

We see that the gray level of a destructible (resp. constructible) point may be lowered (resp. raised) of one unit, while preserving the topology of F. For example in Fig. 3.14a, the point at level 8 is both destructible and constructible; the two points at level 2 are constructible, but only one of them may be raised, because after that, the other point would become non-constructible.

Let $F \in \mathcal{F}$ and $G \in \mathcal{F}$. We say that G *is an elementary homotopic thinning of* F, and we write $F \xrightarrow{e} G$, if there exists a point x that is destructible for F such that $G(x) = F(x) - 1$, and for each $y \neq x$, $G(y) = F(x)$. We say that G *is an homotopic thinning of* F if $G = F$ or if there exists a sequence $\langle G_0, \ldots, G_k \rangle$ such that $G_0 = F$, $G_k = G$ and $G_0 \xrightarrow{e} \ldots \xrightarrow{e} G_k$. Furthermore, if no point of G is destructible, we say that G *is an ultimate homotopic thinning of* F. We define in a dual manner the notions of *homotopic thickening* and *ultimate homotopic thickening*.

For example in Fig. 3.14, image (b) is an ultimate homotopic thinning of (a), and (c) is an ultimate homotopic thickening of (a).

3.3.2 Local Characterizations and Topological Classification of Points

Let $F \in \mathcal{F}$ and $x \in \mathbb{Z}^2$. For the sake of simplicity, we will omit to mention F unless necessary; for example, we will write $N^{++}(x)$ rather than $N^{++}(x, F)$. We define the four neighborhoods:

$$N^{++}(x) = \{y \in N_8^*(x);\ F(y) > F(x)\};$$
$$N^{+}(x) = \{y \in N_8^*(x);\ F(y) \geqslant F(x)\};$$
$$N^{--}(x) = \{y \in N_8^*(x);\ F(y) < F(x)\};$$
$$N^{-}(x) = \{y \in N_8^*(x);\ F(y) \leqslant F(x)\}.$$

We define also:

$$\eta^-(x) = \begin{cases} \max\{F(y);\, y \in N^{--}(x)\}, & \text{if } N^{--}(x) \neq \emptyset, \\ F(x) & \text{otherwise.} \end{cases}$$

It is easy to show that lowering a destructible point x down to the value $\eta^-(x)$ is a homotopic transformation. For example in Fig. 3.14a, the point at level 9 in the third row can be lowered down to 7, then to 4, and finally to 0 without changing the topology of cross-sections. This property, in addition to the local characterization of destructible and constructible points that we present next, allows for the design of efficient algorithms for computing transformations that preserve cross-section topology, on the model of *e.g.* Algorithm *GuidedThinning* (see [20]).

We define the four *connectivity numbers*:

$$T^{++}(x) = \left|C_n\left[x, N^{++}(x)\right]\right|; \qquad T^{+}(x) = \left|C_n\left[x, N^{+}(x)\right]\right|;$$
$$T^{--}(x) = \left|C_{\overline{n}}\left[x, N^{--}(x)\right]\right|; \qquad T^{-}(x) = \left|C_{\overline{n}}\left[x, N^{-}(x)\right]\right|.$$

The following property can be straightforwardly derived from the above definition and from the local characterization of simple points in binary images (see Theorem 1). It shows that connectivity numbers allow for a local characterization of destructible and constructible points.

Let $F \in \mathcal{F}$ and $x \in \mathbb{Z}^2$.

$$x \text{ is destructible for } F \quad \Leftrightarrow \quad T^{+}(x) = 1 \text{ and } T^{--}(x) = 1;$$
$$x \text{ is constructible for } F \quad \Leftrightarrow \quad T^{-}(x) = 1 \text{ and } T^{++}(x) = 1.$$

Furthermore, connectivity numbers allow for a classification of topological characteristics of a point:

x is a *peak* if $T^{+}(x) = 0$; x is *minimal* if $T^{--}(x) = 0$;
x is *k-divergent* if $T^{--}(x) = k$ with $k > 1$;
x is a *well* if $T^{-}(x) = 0$; x is *maximal* if $T^{++}(x) = 0$;
x is *k-convergent* if $T^{++}(x) = k$ with $k > 1$;
x is a *lower point* if it is not maximal; x is an *upper point* if it is not minimal;
x is an *interior point* if it is both minimal and maximal;
x is a *simple side* if it is both destructible and constructible;
x is a *saddle point* if it is both convergent and divergent.

By considering all the possible values of the four connectivity numbers, one proves [9] that the type of a point $x \in \mathbb{Z}^2$, whatever the function $F \in \mathcal{F}$, is necessarily one and only one of the following: 1) a peak; 2) a well; 3) an interior point; 4) a constructible minimal point; 5) a destructible maximal point; 6) a minimal convergent point; 7) a maximal divergent point; 8) a simple side; 9) a destructible convergent point; 10) a constructible divergent point; 11) a saddle point. Figure 3.15 shows examples of seven out of these eleven types; the four other types can be obtained by duality (for example a well is the dual of a peak, etc.).

The rest of this chapter is devoted to three applications of cross-section topology. In these applications, we combine homotopic transformations and transformations that modify topology in a controlled manner.

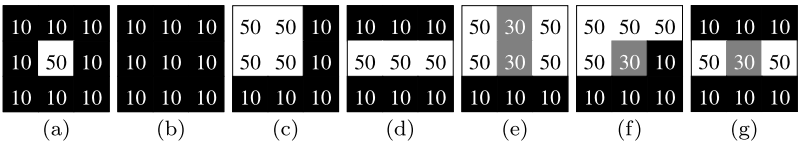


Fig. 3.15 Topological type. The central point has the following type: (**a**): peak; (**b**): interior; (**c**): destructible maximal; (**d**): maximal 2-divergent; (**e**): destructible 2-convergent; (**f**): simple side; (**g**): saddle

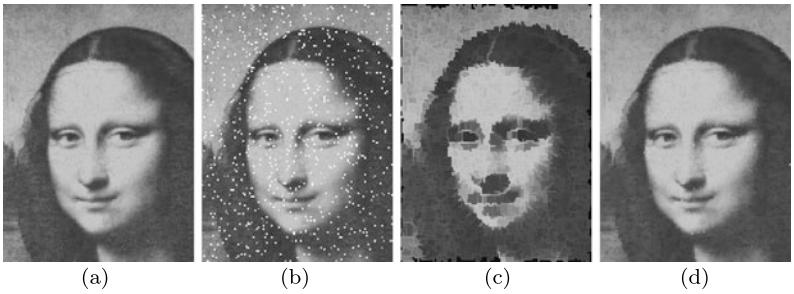


Fig. 3.16 Topological filtering. (**a**): Original image. (**b**): Original image with added impulse noise. (**c**): After 3 steps of homotopic thinning and peak lowering. (**d**): Homotopic reconstruction of (**c**) constrained by (**b**)

3.3.3 Topological Filtering

In the case of impulse noise, a positive impulse takes the form of a small group of pixels, having grayscale values higher than those of pixels in their neighborhood. We can detect a positive impulse made of an isolated pixel x by testing the topological type of x: it is a peak. One can "destroy" this peak by lowering x down to the value $\eta^-(x)$. For impulses formed by several adjacent pixels, this procedure is not sufficient. However, if we apply homotopic thinning to the image, an impulse formed by a few pixels may be reduced to a peak, allowing for its detection and deletion.

On the other hand, we do not want to lower bigger groups of pixels that may constitute significant objects in the image. This is why we need a notion of "thinning step" in order to control the spatial extent of the thinning (see [20] for more details).

In Fig. 3.16, we show in (c) the result of three steps of homotopic thinning applied to image (b), followed by the lowering of all peaks. The positive impulses have been eliminated, but some points outside these impulses have also been lowered. It is thus necessary to restore the initial values of these points. We use for this purpose a homotopic reconstruction operator, which is nothing else but a homotopic thickening constrained by the original image (that is, the final value of a point cannot be higher than the value of this point in the original image). Since only constructible points can be raised, the lowered peaks will not be restored at their original value. Figure 3.16d shows a homotopic reconstruction of (c) constrained

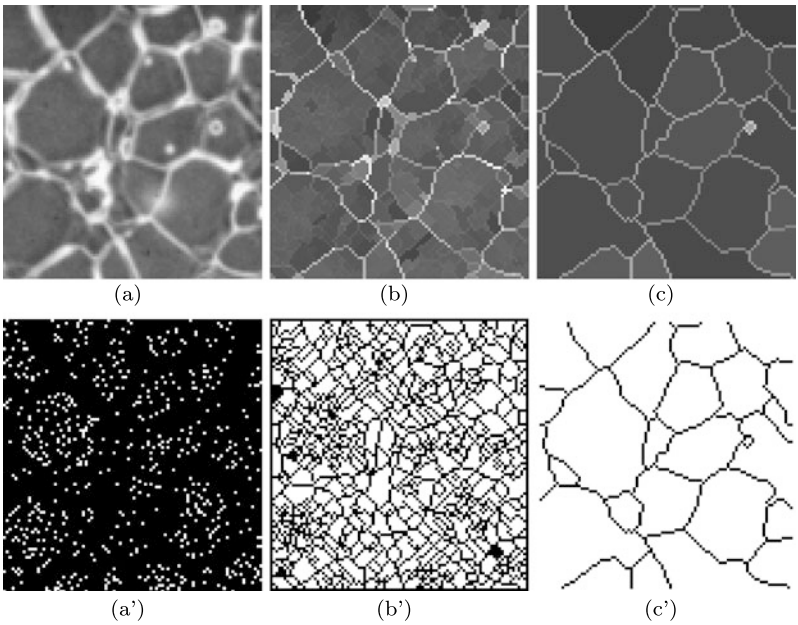


Fig. 3.17 Topological segmentation. (**a**): Original image. (**b**): Ultimate homotopic thinning. (**c**): Ultimate filtered thinning with $\kappa = 40$. (**a'**), (**b'**), (**c'**): In *white*, the minima of (**a**), (**b**), (**c**) respectively

by (b). Negative impulses can be filtered by the dual procedure. This topological filtering gives, for impulse noise, better results than a median filter or a filter based on morphological opening and reconstruction. In particular, it better preserves thin structures.

3.3.4 Topological Segmentation

Figure 3.17a shows an image in which one perceives dark cells separated by lighter borders. Due to noise, this image contains a lot of regional minima: they appear in white in (a'). An ultimate homotopic thinning (b) preserves, by construction, all these minima and extend them as much as possible (b'). Figure 3.18a shows a 1D profile extracted from such an ultimate homotopic thinning. In this profile, the points A, B and C correspond to divergent points that separate neighboring minima. Some of these divergent points (A, B) can be considered as "irregular points" [9]: we would like to lower them in order to eliminate, by merging, some minima having small depth.

To this aim, we introduce the notions of κ-destructible point and ultimate filtered thinning. Intuitively, a point κ-destructible x is either a destructible point, or a peak, or a divergent point that lies on a crest line that divides its neighborhood into several lower regions, such that at most one of these regions has a difference of altitude with

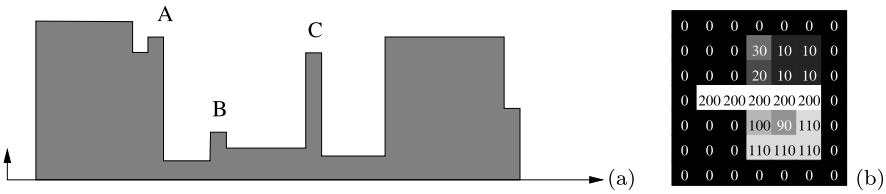


Fig. 3.18 Illustration of κ-destructible points. (**a**): A 1D profile of an ultimate homotopic thinning. (**b**): An image with two 10-destructible points (levels *20* and *100*) that are not destructible

respect to x that is greater than κ. Thus, the parameter κ corresponds to a notion of local contrast. For example, points at levels 20 and 100 in Fig. 3.18b are both 10-destructible, but are not destructible. An ultimate filtered thinning is defined in a similar manner as an ultimate homotopic thinning, by using "κ-destructible" instead of "destructible".

In Fig. 3.17c, we see an ultimate filtered thinning of (a) with $\kappa = 40$. A binary segmented image (c') is obtained by extracting regional minima of (c). Note that this segmentation method involves only one parameter (κ) relative to a notion of local contrast.

3.3.5 Crest Restoration Based on Topology

Segmentation methods that are based on minima extraction and region merging, as well as those based on contour detection, are sensitive to the quality of the crests that separate the regions of interest (see Fig. 3.17, Fig. 3.20), which may be altered by noise. In this section, we propose a procedure for detecting and eliminating narrow passes on the crests of a 2D function.

First of all, we apply some steps of filtered or homotopic thinning, in order to reduce crests to thin lines (see Fig. 3.19b). After this, we can detect points that belong to "thin crests", and that must be raised in order to eliminate passes.

Let $X \subseteq \mathbb{Z}^2$ and $x \in X$, x is a *separating point (for X)* if $\overline{T}(x) \geqslant 2$. Let $F \in \mathcal{F}$, a point $x \in \mathbb{Z}^2$ is called a *separating point (for F)* if there exists a level $k \in \mathbb{Z}$ such that x is a separating point for the set F_k. Note that, if x is a divergent point for F, then x is necessarily a separating point for F, but the converse is not true. For example, in Fig. 3.19e, the points at levels 15, 20 and 25 are separating points, whereas only the point at level 15 and the second point at level 20 (from the top) are divergent points.

We see in Fig. 3.19b that, in order to eliminate the pass at level 90, we can raise separating points that are constructible, until a saddle point appears. This saddle point can then be detected and raised. We also see in Fig. 3.19b that, if we iteratively raise constructible separating points without any restriction, we will also reinforce some low crest lines, like the one at level 60. Indeed, the point at level 60 circled in white is a constructible separating point. Furthermore, we cannot use the notions

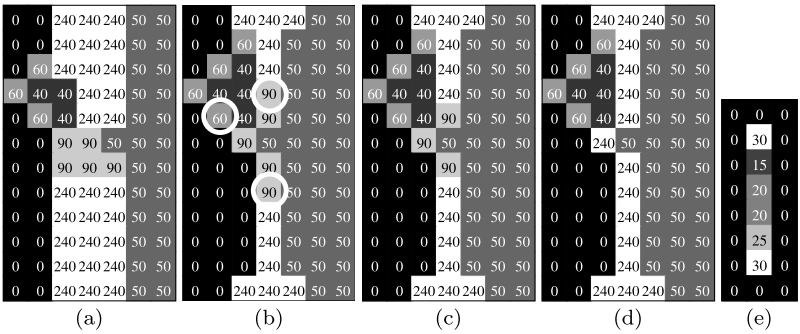


Fig. 3.19 Crest restoration. (**a**): The lowest value on the crest is the one of the pass (*90*). (**b**): After one step of homotopic thinning. (**c**, **d**): After 1 and 3 iterations of the crest restoration algorithm. (**e**): Points at levels *15*, *20* and *25* are separating points

of κ-destructible point and filtered thinning in this case, because we would take the risk of lowering those very passes that we want to raise.

Now, let us define a class of points that are "good candidates" for crest restoration. Intuitively, such a point may be characterized by the presence, in its neighborhood, of a point y that is a separating point for the section at level $k = F(x)$ but is not separating for higher sections. This is formalized through the notion of extensible point defined below.

Let $F \in \mathcal{F}$, a point $x \in \mathbb{Z}^2$ that is a separating point for F is called *extensible* if it is, either a constructible point, of a saddle point for F, and if x has at least one neighbor y that satisfies the following two conditions:

(i) y is a separating point (in the binary sense) for F_k, with $k = F(x)$, and
(ii) y is not a separating point (in the binary sense) for any cross-section F_j with $j > k$.

For example in Fig. 3.19b, we can check that the two circled constructible points at level 90 are extensible, because each of them has a neighbor at 240 that is separating for F_{90} but not for F_{91} and higher sections; whereas the circled constructible point at level 60 is not extensible. Indeed, the point at 90 adjacent to the latter point is separating both for F_{60} and for F_{61}.

The crest restoration method proceeds by iteratively detecting and raising extensible points. A more detailed description of the method can be found in [20]. In Fig. 3.19c, we see the result after applying one step of the method on (b). In particular, we see that two points at level 90 in (b) have been raised up to 240, and that points at level 60 have not been modified. In (d), we see the result after three iterations: the crest at 240 has been restored. Further iterations would not modify this result.

In Fig. 3.20, we illustrate this method on a gradient image (b). Image (b) is first thinned, giving (c). If we threshold this image, we see that either significant contour segments are lost (d), or we get too many details. Image (e) has been obtained from (c) by crest restoration until stability. The same threshold was applied on (c, e),

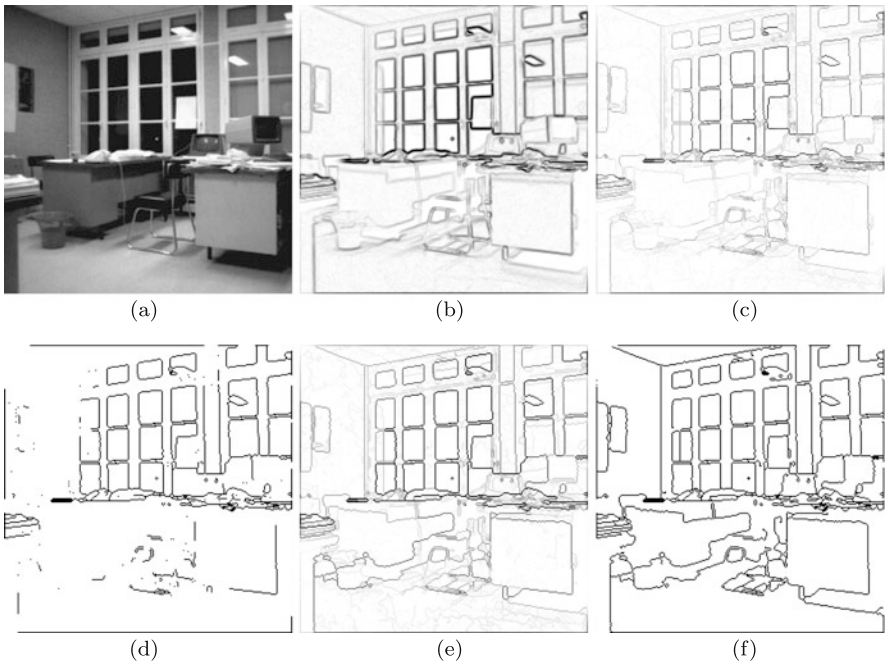


Fig. 3.20 Crest restoration. (**a**): Original image. (**b**): After applying a gradient modulus operator (the lowest values are in *white*). (**c**): After a filtered thinning. (**e**): After crest restoration, performed until stability. (**d**, **f**): Thresholds of (**c**, **e**) respectively, at the same level

giving (d, f) respectively. We see that many significant contour segments have been recovered, without introducing artefacts.

3.4 Parallel Thinning

In Sects. 3.2 and 3.3, we described transformations that are sequential by nature. By this, we mean that after each point modification, the result of this modification has to be taken into account in order to perform simplicity tests for other points. Consequently, depending on the order in which the points are examined, some arbitrary choices may be done, and different results may be obtained depending on these choices. Even when one uses a priority function to guide the thinning, it is not seldom that many points share the same priority value, and arbitrary decisions are still necessary.

Another strategy for thinning objects consists of removing some of its border points in parallel [37, 38]. However, parallel deletion of simple points does not, in general, guarantee topology preservation: see for example Fig. 3.3a, where removing both simple points u, v would merge two components of the background. In fact, such a guarantee is not obvious to obtain, even for the 2D case (see [17], where

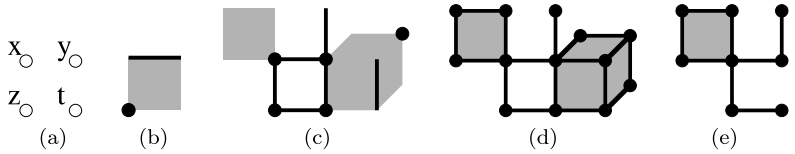


Fig. 3.21 (**a**) Four points in $\mathbb{Z}^2$: $x = (0, 1)$; $y = (1, 1)$; $z = (0, 0)$; $t = (1, 0)$. (**b**) A graphical representation of the set of faces $\{f_0, f_1, f_2\}$, where $f_0 = \{z\} = \{0\} \times \{0\}$ (a 0-face), $f_1 = \{x, y\} = \{0, 1\} \times \{1\}$ (a 1-face), and $f_2 = \{x, y, z, t\} = \{0, 1\} \times \{0, 1\}$ (a 2-face). (**b**, **c**) A set of faces that is not a complex. (**d**, **e**) A set of faces that is a complex

fifteen published parallel thinning algorithms are analyzed, and counter-examples are shown for five of them).

In order to study the conditions under which points may be removed simultaneously while preserving topology of 2D objects, C. Ronse introduced minimal non-simple sets [33]. This work leads to verification methods for the topological soundness of parallel thinning algorithms. Such methods have been proposed for 2D algorithms by C. Ronse [33] and R. Hall [25], they have been developed for the 3D case by T.Y. Kong [26, 27] and C.M. Ma [31]. For the 3D case, one of the authors [4] introduced the notion of P-simple point as a verification method but also as a methodology to design parallel thinning algorithms.

More recently, one of the authors introduced in [5] a general framework for studying parallel homotopic thinning in spaces of any dimension. This framework, called critical kernels, is developed in the context of abstract simplicial or cubical complexes, but it also permits to prove properties of algorithms acting in $\mathbb{Z}^D$. In particular, the notion of crucial point is introduced in [7] and [6], for the 2D and the 3D case respectively, together with the proof that any set of non-crucial points can be removed in parallel from any object in $\mathbb{Z}^D$ without changing its topological characteristics.

In Sects. 3.4.1–3.4.4, we present a minimal set of notions needed to survey the critical kernels framework. Section 3.4.5 is devoted to parallel thinning in $\mathbb{Z}^D$, where results about critical kernels are used only to prove topological correctness. The reader who prefers to quickly implement algorithms may jump directly to this latter section.

3.4.1 Cubical Complexes

Intuitively, a cubical complex may be thought of as a set of elements having various dimensions (*e.g.* cubes, squares, edges, vertices) glued together according to certain rules (see Fig. 3.21d).

Let $\mathbb{Z}$ be the set of integers. We consider the families of sets $\mathbb{F}_0^1$, $\mathbb{F}_1^1$, such that $\mathbb{F}_0^1 = \{\{a\} \mid a \in \mathbb{Z}\}$, $\mathbb{F}_1^1 = \{\{a, a + 1\} \mid a \in \mathbb{Z}\}$. A subset f of $\mathbb{Z}^D$, $D \geqslant 2$, which is the Cartesian product of exactly d elements of $\mathbb{F}_1^1$ and $(D - d)$ elements of $\mathbb{F}_0^1$ is

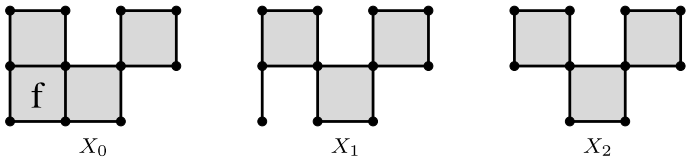


Fig. 3.22 X_0: a pure 2-complex. X_1: a complex such that X_0 collapses onto X_1; a free pair composed of a square and an edge has been removed. X_2: a complex such that X_1 collapses onto X_2; (a free pair composed of an edge and a vertex has been removed), hence X_0 collapses onto X_2

called a *face* or a *d-face* in $\mathbb{Z}^D$, d is the *dimension of* f, we write $\dim(f) = d$. See Fig. 3.21a,b for an illustration.

We denote by $\mathbb{F}^D$ the set composed of all faces in $\mathbb{Z}^D$. A d-face is called a *point* if $d = 0$, a *(unit) edge* if $d = 1$, a *(unit) square* if $d = 2$, a *(unit) cube* if $d = 3$. Observe that any non-empty intersection of faces is a face. For example, the intersection of two 2-faces A and B may be either a 2-face (if $A = B$), a 1-face, a 0-face, or the empty set.

Let f be a face in $\mathbb{F}^D$. We set $\hat{f} = \{g \in \mathbb{F}^D \mid g \subseteq f\}$ and $\hat{f}^* = \hat{f} \setminus \{f\}$; we call $\hat{f}^*$ the *boundary of* f. Any $g \in \hat{f}$ is called a *face of* f. If X is a finite set of faces in $\mathbb{F}^D$, we write $X^- = \bigcup\{\hat{f} \mid f \in X\}$, X^- is the *closure of* X. A finite set X of faces in $\mathbb{F}^D$ is a *complex (in* $\mathbb{F}^D$*)* if $X = X^-$. If $Y \subseteq X$ and Y is a complex, then we say that Y *is a subcomplex of* X. In the sequel, the symbol X will denote a complex in $\mathbb{F}^D$, and the symbol f will denote a face of X.

See in Fig. 3.21d,e two examples of complexes, and in Fig. 3.21b,c examples of sets of faces that are not complexes. The complex in Fig. 3.21d is the closure of the complex in Fig. 3.21c.

Let $d = \dim(f)$. We say that f is a *facet of* X or an *d-facet of* X if there is no face $g \in X$ such that $f \in \hat{g}^*$, in other words, if f is maximal for inclusion. We set $\dim(X) = \max\{\dim(f) \mid f \in X\}$. We say that X is an d-complex if $\dim(X) = d$. We say that X is *pure* if, for each facet f of X, we have $\dim(f) = \dim(X)$. For example in Fig. 3.22, X_0 and X_2 are pure 2-complexes, whereas X_1 is a 2-complex that is not pure.

The operation of detachment allows us to remove a subset from a complex, while guaranteeing that the result is still a complex.

Let $Y \subseteq X$. We set $\mathit{Detach}(Y, X) = (X \setminus Y)^-$. The set $\mathit{Detach}(Y, X)$ is a complex which is the *detachment* of Y from X. Figure 3.21e shows the detachment of $\hat{f}$ from X, where X is the complex of Fig. 3.21d and f is the 3-face of X.

3.4.2 Collapse and Simple Facets

The *collapse operation* is an elementary topology-preserving transformation which has been introduced by J.H.C. Whitehead [40], and plays an important role in combinatorial topology. It can be seen as a discrete analogue of a continuous defor-

mation (a strong deformation retract). Collapse is known to preserve the homotopy type.

Consider a pair $(f, g) \in X^2$. If f is the only face of X that strictly includes g, then g is said to be *free for* X and the pair (f, g) is said to be a *free pair for* X. Note that, if (f, g) is a free pair, then f is necessarily a facet of X and $\dim(g) = \dim(f) - 1$.

Let (f, g) be a free pair for X. Let $d = \dim(f)$. The complex $X \setminus \{f, g\}$ is an *elementary collapse of* X, or an *elementary* d*-collapse of* X. The pair (f, g) is also called a *free* d*-pair (for* X*)*.

Let Y be a complex. We say that X *collapses onto* Y, and we write $X \searrow Y$, if $Y = X$ or if there exists a sequence of complexes $\langle X_0, \ldots, X_\ell \rangle$ such that $X_0 = X$, $X_\ell = Y$, and X_i is an elementary collapse of X_{i-1}, for each $i \in \{1, \ldots, \ell\}$. See Fig. 3.22 for an illustration.

We give now a definition of a simple facet, it may be seen as a discrete analogue of the one given by T.Y. Kong in [28] which lies on continuous deformations in the D-dimensional Euclidean space.

Definition 6 ([5]) Let f be a facet of X. We say that $\hat{f}$ and f are *simple for* X if X collapses onto $Detach(\hat{f}, X)$.

For example in Fig. 3.22, we have $X_2 = Detach(\hat{f}, X_0)$, and since $X_0 \searrow X_2$, the facet f is simple for X_0.

The notion of attachment, as introduced by T.Y. Kong [27, 28], leads to a local characterization of simple facets. The *attachment of* $\hat{f}$ *for* X is the complex $Attach(\hat{f}, X) = \hat{f}^* \cap [Detach(\hat{f}, X)]$. In other words, a face g is in $Attach(\hat{f}, X)$ if g is in $\hat{f}^*$ and if g is a face of a facet h distinct from f.

As an easy consequence of the above definitions, the facet f is simple for X if and only if $\hat{f}$ collapses onto $Attach(\hat{f}, X)$. This property led us to introduce new characterizations of simple points in 2D, 3D and 4D [19].

3.4.3 Critical Kernels

Let us briefly recall the framework introduced by one of the authors (in [5]) for thinning, in parallel, discrete objects with the warranty that topology is preserved. We focus here on the two- and three-dimensional cases, but in fact the notions and results in this section are valid for complexes of arbitrary dimension. This framework is based solely on three notions: the notion of an essential face, which allows us to define the core of a face, and the notion of a critical face.

Definition 7 ([5]) We say that f is an *essential face for* X if f is precisely the intersection of all facets of X that contain f. We denote by $Ess(X)$ the set composed of all essential faces of X. If Y is a subcomplex of X and $Ess(Y) \subseteq Ess(X)$, then we say that Y is an *essential subcomplex of* X.

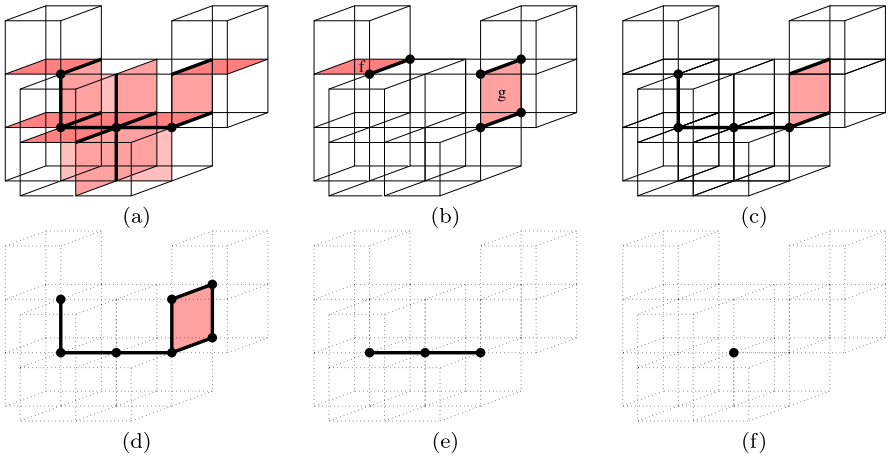


Fig. 3.23 (**a**): A 3-complex X_0, made of 12 cubes. The essential faces for X_0 that are not facets are *highlighted*. (**b**): Two essential 2-faces f, g and their cores (in *black*). (**c**): X_0 and its critical faces (*highlighted*). (**d**): The critical kernel $X_1 = Critic(X_0)$. (**e**): $X_2 = Critic(X_1)$. (**f**): $X_3 = Critic(X_2) = Critic(X_3)$

Observe that a facet of X is necessarily an essential face for X. Observe also that, if X and Y are both pure D-complexes, then Y is an essential subcomplex of X whenever Y is a subcomplex of X.

Definition 8 ([5]) Let $f \in Ess(X)$. The *core of* $\hat{f}$ *for* X is the complex $Core(\hat{f}, X) = \cup\{\hat{g} \mid g \in Ess(X) \cap \hat{f}^*\}$.

Definition 9 ([5]) Let $f \in X$. We say that f and $\hat{f}$ are *regular for* X if $f \in Ess(X)$ and if $\hat{f}$ collapses onto $Core(\hat{f}, X)$. We say that f and $\hat{f}$ are *critical* for X if $f \in Ess(X)$ and if f is not regular for X.

We set $Critic(X) = \cup\{\hat{f} \mid f$ is critical for $X\}$, we say that $Critic(X)$ is the *critical kernel of* X.

Figure 3.23 illustrates these definitions. In Fig. 3.23b, we see that $\hat{f}$ collapses onto the core of f, thus f is regular; and that $\hat{g}$ does not collapse onto the core of g, thus g is critical. Note that, in this complex, all facets (3-faces) are regular.

The following theorem is the most fundamental result concerning critical kernels. Note that the theorem holds whatever the dimension.

Theorem 3 ([5]) *Let Y be an essential subcomplex of X.*

(i) *The complex X collapses onto its critical kernel.*
(ii) *If Y contains the critical kernel of X, then X collapses onto Y.*
(iii) *If Y contains the critical kernel of X, and if Z is an essential subcomplex of X such that $Y \subseteq Z$, then Z collapses onto Y.*

In Fig. 3.23, we show that the very notion of critical kernel can be seen as a powerful thinning algorithm, which consists of computing iteratively the critical kernel of the result of the preceding computation. Furthermore, Theorem 3(ii) tells us that any essential subcomplex Y of X that is "between" X_0 (Fig. 3.23a) and X_1 (Fig. 3.23d) is such that X_0 collapses onto Y. This is true, in particular, of any subcomplex Y that is a pure 3-complex containing X_1. This property gives birth to a wide class of parallel thinning algorithms, where different criterions, based *e.g.* on geometrical notions, can be used in order to choose a particular set as the result of a single thinning step (see Sect. 3.4.5).

3.4.4 Crucial Cliques and Faces

In the image processing literature, a digital image is often considered as a set of pixels in 2D or voxels in 3D. A pixel (resp. a voxel) is an elementary square (resp. cube), thus an easy correspondence can be made between this classical view and the framework of cubical complexes. From now on, we consider only complexes whose facets are all D-faces, *i.e.*, pure D-complexes.

Note that, if X is a pure D-complex in $\mathbb{F}^D$ and if f is a D-face of X, then $Detach(\hat{f}, X)$ is a pure complex in $\mathbb{F}^D$. There is indeed an equivalence between the operation on complexes that consists of removing (by detachment) the closure of a simple D-face, and the removal of an 8-simple (resp. 26-simple) point in the framework of 2D (resp. 3D) digital topology (see [27, 28]).

When X is a pure D-complex (*e.g.*, a union of voxels in $\mathbb{F}^3$), the critical kernel of X is not necessarily a pure D-complex (see Fig. 3.23d). The notion of crucial face, introduced in [6, 7], allows us to recover a pure D-subcomplex Y of an arbitrary pure D-complex X, under the constraint that X collapses onto Y.

Definition 10 ([6]) A face f in X is a *maximal critical face*, or an *M-critical face (for X)*, if f is a facet of $Critic(X)$.

The set of all the facets of X that contain an M-critical face f is called the *crucial clique (for X) induced by f*. Each facet in a crucial clique is called a *crucial face*.

Some 2D crucial cliques are illustrated in Fig. 3.24. The following corollary of Theorem 3 tell us that, informally speaking, a thinning step that preserves all non-simple pixels (voxels) and at least one pixel (voxel) in each crucial clique, preserves topology.

Corollary 2 *Let Y be a subcomplex of X that is also a pure D-complex.*

If any critical D-face of X and at least one D-face of each crucial clique of X is in Y, then X collapses onto Y.

During the process of thinning an object, we often want to keep certain faces like curve extremities for example, if we want to obtain a curvilinear skeleton. That is why we introduce the following definition in order to generalize the previous

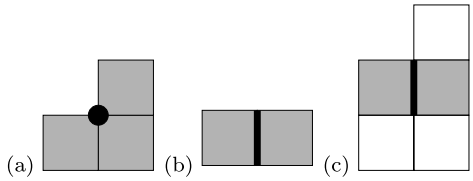
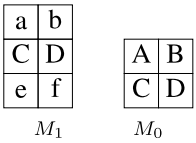


Fig. 3.24 Crucial cliques in $\mathbb{F}^2$ (represented in *light gray*): (**a**) induced by an M-critical 0-face; (**b**, **c**) induced by an M-critical 1-face. The considered M-critical faces are in *bold*. The core of the M-critical face in (**a**, **b**) is empty, in (**c**) it consists of two 0-faces

Fig. 3.25 Masks for 1-crucial (M_1) and 0-crucial (M_0) points



M_1 M_0

notions. Intuitively, the set K corresponds to a set which is preserved by a thinning algorithm (a constraint set).

Definition 11 ([6]) Let K be a set composed of facets of X. A subcomplex C of X is a *crucial clique for* $\langle X, K \rangle$ if C is a crucial clique for X such that $C \cap K = \emptyset$. In this case, each facet in C is called a *crucial face for* $\langle X, K \rangle$.

3.4.5 Parallel Thinning Algorithms

In the sequel, we give a characterization of crucial points or pixels in $\mathbb{Z}^2$, which can be checked in a quite simple manner with the help of masks. Thanks to this characterization, one can easily implement parallel thinning algorithms that are guaranteed to preserve topology. The interested reader is referred to [5–8] for the proofs of the stated properties.[2] Implementations (in source code) are available on the critical kernels web site.[3] We emphasize that no representation of cubical complexes is used for computing this characterization and thinning methods based on it: both inputs and outputs, as well as intermediate results, are mere binary images (*i.e.*, subsets of $\mathbb{Z}^D$). For the sake of simplicity, we limit ourselves to the 2D case, the reader can find a similar characterization for the 3D case in [6].

The masks M_1, M_0 are given in Fig. 3.25. For the mask M_1, we also consider the mask obtained from it by applying a $\pi/2$ rotation: we get 3 masks (2 for M_1, and 1 for M_0).

[2] Note that the characterization that we use in this chapter for the 2D case is actually derived from the ones of [6], which deals with 3D. This allows us to present a characterization that is simpler than the one proposed in [7].

[3] http://www.esiee.fr/~info/ck.

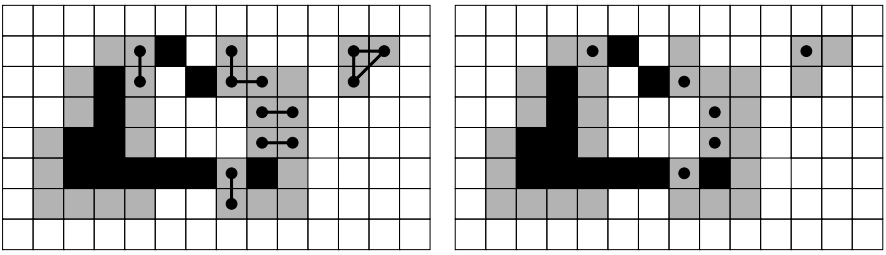


Fig. 3.26 Illustration of crucial points (pixels). *Left*: the simple points are in *gray*, the crucial points are marked by a *black disk*. The *couples of black disks* that are linked by a *bar*, in the biggest connected component, represent all the crucial cliques that are detected by the mask M_1. The *triplet of black disks* that are linked by a *triangle*, in the smallest connected component, represents a crucial clique that is detected by mask M_0. All simple points that are not crucial may be removed in parallel by a topology-preserving algorithm. *Right*: the crucial points marked by a *black disk* constitute a set of points that is sufficient to ensure topology preservation. All other simple points may be safely removed in parallel

Definition 12 Let $X \subseteq \mathbb{Z}^2$, and let M be a set of points of X.

1) The set M *matches the mask* M_1 if:
   (i) $M = \{C, D\}$; and
   (ii) the points C, D are simple for X; and
   (iii) the sets $\{a, b\} \cap X$ and $\{e, f\} \cap X$ are either both empty or both non-empty.
2) The set M *matches the mask* M_0 if:
   (i) $M = \{A, B, C, D\} \cap X$; and
   (ii) the points in M are simple and not matched by M_1; and
   (iii) at least one of the sets $\{A, D\}, \{B, C\}$ is a subset of M.

In the following, the set K plays the role of a constraint set (see Sect. 3.2.4). There exists a "natural" one-to-one correspondence between the subsets of $\mathbb{Z}^D$ and the pure D-complexes in $\mathbb{F}^D$ (see [6, 7]). Namely, with each point (pixel, voxel) of $\mathbb{Z}^D$ we associate a facet of $\mathbb{F}^D$ (unit square, unit cube). We extend our vocabulary accordingly: for instance, we say that a point $x \in X$ is crucial whenever the corresponding facet in the corresponding complex is crucial.

Theorem 4 *Let $X \subseteq \mathbb{Z}^2$, $K \subseteq X$, and let M be a set of points in $X \setminus K$ that are 8-simple for X.*

Then, M is a crucial clique for $\langle X, K \rangle$ if and only if M matches the mask M_0 or the mask M_1.

An illustration is given in Fig. 3.26. From Corollary 2, we deduce that a parallel thinning step that preserves all critical (*i.e.*, non-simple) points and at least one point of each crucial clique, preserves topology.

The simplest parallel thinning algorithm based on crucial points is Algorithm 3. It consists of iteratively detecting the points that are simple and not crucial (with respect to the current object and a possibly empty constraint set), and removing them

in parallel. This algorithm makes no arbitrary choice: whenever a crucial clique is detected, all its points are preserved.

Algorithm 3: CrucialThinning

Data : $D \in \{2, 3\}$, a subset X of $\mathbb{Z}^D$, a set K of points of X
Result : X
repeat
  $V =$ set of points of X that are simple and not crucial for $\langle X, K \rangle$;
  $X = X \setminus V$;
until $V = \emptyset$;

In [6, 7], we provide various algorithms based on the same principle, that compute different kinds of skeletons: curvilinear of surface skeletons in 3D, skeletons that are guaranteed to contain the medial axis, minimal skeletons, asymmetric skeletons, skeletons of three-dimensional objects made of surfels ...

Back to guided thinning, we show with the next algorithm how to use the notion of crucial point in order to avoid arbitrary choices when several candidate points share the same priority. The result of the following procedure is thus uniquely defined, given any shape and any priority function.

Algorithm 4: GuidedParallelThinning_Version_0

Data : $D \in \{2, 3\}$, a subset X of $\mathbb{Z}^D$, a function P from X into $\mathbb{R} \cup \{+\infty\}$
Result : X
repeat
  $\pi = \min\{P(x), x \in X\}$;
  if $\pi < +\infty$ **then**
    $U = \{x \in X \mid P(x) = \pi$ and x is simple for $X\}$;
    $V = \{x \in X \mid x$ is not crucial for $\langle X, X \setminus U \rangle\}$;
    $X = X \setminus V$;
until $(\pi = +\infty)$ *or* $(V = \emptyset)$;

By construction, at each iteration of Algorithm 4, the current set X has the same topology as the initial object. By "stacking" these sets that are nested in each other, we can build a function that is a compact representation of this family of thinnings. The simplest way to do this consists of defining a function F that associates with each point x of X, the number of the iteration where x is deleted, or $+\infty$ whenever x is still in the final set. Hence, thresholding F at any integer level provides one of the thinnings. Instead of the number of the iteration, we can indeed choose any number that increases at each iteration. This is not necessarily the case of the number π, but

Algorithm 5: GuidedParallelThinning

Data : $D \in \{2, 3\}$, a subset X of $\mathbb{Z}^D$, a function P from X into $\mathbb{R} \cup \{+\infty\}$
Result : A function T from X into $\mathbb{R} \cup \{+\infty\}$
$\tau = -\infty$;
foreach $x \in X$ **do** $T(x) = +\infty$;
repeat
  $\pi = \min\{P(x), x \in X\}$;
  if $\pi < +\infty$ **then**
    if $\pi > \tau$ **then** $\tau = \pi$;
    $U = \{x \in X \mid P(x) = \pi$ and x is simple for $X\}$;
    $V = \{x \in X \mid x$ is not crucial for $\langle X, X \setminus U \rangle\}$;
    $X = X \setminus V$;
    foreach $x \in V$ **do** $T(x) = \tau$;
until $(\pi = +\infty)$ *or* $(V = \emptyset)$;

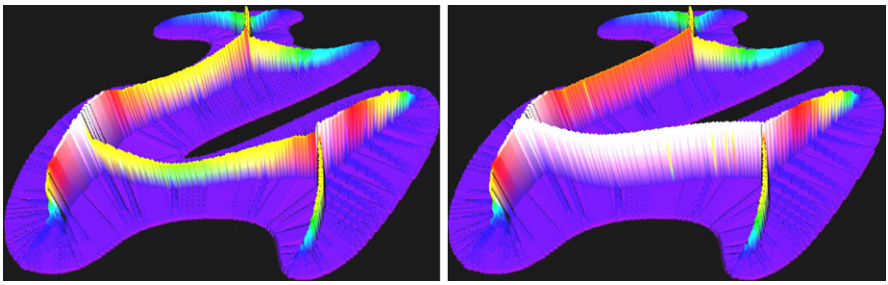


Fig. 3.27 *Left*: a visualization of the map PR_X, for the same shape X as in Fig. 3.8. *Right*: the result of Algorithm *GuidedParallelThinning*

a slight modification of the algorithm allows us to compute a function that is closely related to the priority function used as input. This leads us to Algorithm 5.

Figure 3.27 shows, on the right, an example of function computed by Algorithm *GuidedParallelThinning*, using the same input shape X as in Fig. 3.8, and the priority function PR_X defined in Sect. 3.2.6 (depicted on the left).

As for Algorithm *GuidedThinning*, it is possible to implement this algorithm in $O(n \log n)$ or $O(n)$ time complexity, depending on the nature of the priority function.

3.5 Perspectives

All the algorithms presented in this chapter work on images defined on $\mathbb{Z}^D$. They fit in the framework, called digital topology, pioneered by A. Rosenfeld [34]. The success of digital topology is mainly due to its simplicity, especially for the 2D case.

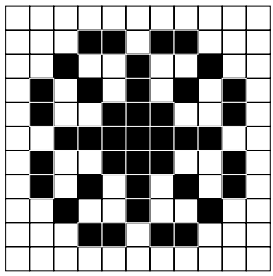


Fig. 3.28 An ultimate skeleton (all pixels are non-simple) that is not thin

However, topological properties in higher dimensions are not easily handled in this framework.

Besides, in Sects. 3.4.1–3.4.4, we described a framework based on cubical complexes in which topological notions are defined quite naturally. Abstract (cubical) complexes have been promoted in particular by V. Kovalevsky [30], in order to provide a sound topological basis for image analysis. The cubical complexes framework allows for retrieving the results obtained using digital topology, providing a better understanding of these results. Furthermore, new properties can be proved and new methods can be developed in this framework, as showed by the example of critical kernels for the study of parallel homotopic thinning in any dimension.

Further developments are needed to fully explore the possibilities and the benefits of working directly on objects that are general cubical complexes, and not only pure ones as it is the case in this chapter. In applications, this should lead in particular to easier characterization, detection and analysis of lower-dimensional structures, such as curves in 2D and 3D, and surfaces in 3D.

To illustrate this, let us consider the example of Fig. 3.28. In the continuous framework, the skeleton of a bounded D-dimensional object always has a dimension that is at most $D - 1$. That is, the skeleton of any object in 2D is made of curves (1D) and points (0D). Figure 3.28 is a classical example showing that this property of thinness is not always true in the digital topology framework.

However it is indeed possible to provide thinness guarantees in the cubical complex framework. Consider the following thinning scheme, based on the collapse operation (see Sect. 3.4.2). Each thinning step is decomposed into four (in 2D) substeps corresponding to the four principal directions of the grid, named north (N), south (S), west (W), and east (E). A *north free k-pair* is a pair of faces (f, g) such that f is the only face that strictly includes g, $\dim(f) = k$, and g is on the north of f. In the substep N, only north free pairs are considered. All north free 2-pairs are marked, and then removed in parallel (see Fig. 3.29a,b). Then, all north free 1-pairs are marked, and then removed in parallel (see Fig. 3.29b,c). South, west and east substeps are defined similarly. The thinning scheme iterates such substeps until no more free pair can be found in a complete step (NSWE). The topological soundness of this scheme can easily be proved. In Fig. 3.29d, we show the final result obtained from the object of Fig. 3.29a. Observe that the obtained skeleton is only composed of 0-faces and 1-faces, and can indeed be interpreted as a set of curves. This thinness property may also be proved in the general case. Of course, additional

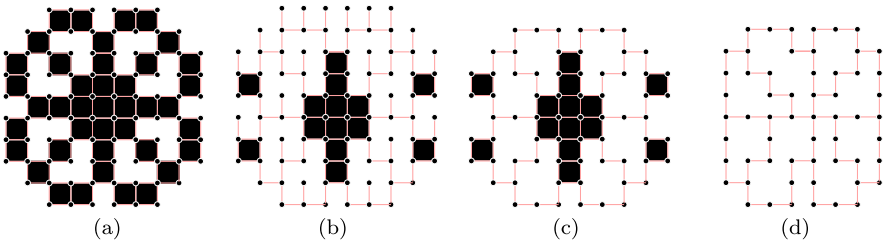


Fig. 3.29 Illustration of a thinning scheme based on collapse (see text)

conditions may be added to this scheme in order to preserve geometrical features such as curve extremities (see [14]).

The cross-section topology approach presented in Sect. 3.3 can also be adapted to the case of functions defined on cubical complexes, and benefit from the ease of defining sound parallel topological operators, based on the critical kernels main property (Theorem 3), or directly on the collapse operation.

3.6 Conclusion

We have seen that it is possible to design topological operators acting on binary 2D and 3D images and also on grayscale images, which are well defined, have proven topological properties and can be implemented through efficient algorithms.

We studied operators that transform an image while preserving its topological characteristics, and also operators that selectively modify these characteristics in order to achieve some filtering, segmentation or restoration.

Thanks to the general scheme of guided thinning that we promote in this chapter, the geometrical features of the processed objects may be taken into account in a flexible way, through the choice of adapted priority functions (*e.g.* distance maps) and constraint sets.

In addition to the sequential approach, which has the advantage of being simple but the drawback of needing arbitrary decisions to be made, we present a tractable way to design sound parallel homotopic thinning algorithms, based on the critical kernels framework. We show that the guided thinning strategy, in particular, may benefit from this approach and result in a well-defined and flexible thinning scheme.

The critical kernels framework is based on cubical complexes, that we shortly presented in Sects. 3.4.1–3.4.4. In this chapter, cubical complexes were used only to prove topological properties of algorithms acting in $\mathbb{Z}^D$. Further developments are needed to build a coherent set of image processing tools based on cubical complexes, dealing with both binary and grayscale images, encompassing and extending the set of digital topology tools.

Acknowledgements This work has been partially supported by the "ANR BLAN07-2_184378 MicroFiss" project and the "ANR-2010-BLAN-0205 Kidico" project.

References

1. Aktouf, Z., Bertrand, G., Perroton, L.: A three-dimensional holes closing algorithm. Pattern Recognit. Lett. **23**(5), 523–531 (2002).
2. Attali, D., Boissonnat, J.D., Edelsbrunner, H.: Stability and computation of the medial axis—a state-of-the-art report. In: Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration, pp. 109–125. Springer, Berlin (2009)
3. Bertrand, G.: Simple points, topological numbers and geodesic neighborhoods in cubic grids. Pattern Recognit. Lett. **15**, 1003–1011 (1994)
4. Bertrand, G.: On P-simple points. C. R. Acad. Sci., Sér. 1 Math. **321**, 1077–1084 (1995)
5. Bertrand, G.: On critical kernels. C. R. Acad. Sci., Sér. 1 Math. **345**, 363–367 (2007)
6. Bertrand, G., Couprie, M.: New 3D parallel thinning algorithms based on critical kernels. In: Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 4245, pp. 580–591. Springer, Berlin (2006)
7. Bertrand, G., Couprie, M.: Two-dimensional thinning algorithms based on critical kernels. J. Math. Imaging Vis. **31**(1), 35–56 (2008)
8. Bertrand, G., Couprie, M.: On parallel thinning algorithms: minimal non-simple sets, P-simple points and critical kernels. J. Math. Imaging Vis. **35**(1), 23–35 (2009)
9. Bertrand, G., Everat, J.C., Couprie, M.: Image segmentation through operators based upon topology. J. Electron. Imaging **6**(4), 395–405 (1997)
10. Bertrand, G., Malandain, G.: A new characterization of three-dimensional simple points. Pattern Recognit. Lett. **15**(2), 169–175 (1994)
11. Blum, H.: An associative machine for dealing with the visual field and some of its biological implications. Biol. Prototypes Synthetic Syst. **1**, 244–260 (1961)
12. Blum, H.: A transformation for extracting new descriptors of shape. In: Models for the Perception of Speech and Visual Form, pp. 362–380. MIT Press, Cambridge (1967)
13. Borgefors, G.: Distance transformations in digital images. Comput. Vis. Graph. Image Process. **34**, 344–371 (1986)
14. Chaussard, J.: Topological tools for discrete shape analysis. Ph.D. thesis, Université Paris-Est (December 2010)
15. Chaussard, J., Couprie, M., Talbot, H.: Robust skeletonization using the discrete λ-medial axis. Pattern Recognit. Lett. **32**(9), 1384–1394 (2011)
16. Chazal, F., Lieutier, A.: The λ-medial axis. Graph. Models **67**(4), 304–331 (2005)
17. Couprie, M.: Note on fifteen 2D parallel thinning algorithms. Tech. Rep. IGM2006-01, Université de Marne-la-Vallée (2006)
18. Couprie, M., Bertrand, G.: Topology preserving alternating sequential filter for smoothing 2D and 3D objects. J. Electron. Imaging **13**(4), 720–730 (2004)
19. Couprie, M., Bertrand, G.: New characterizations of simple points in 2D, 3D and 4D discrete spaces. IEEE Trans. Pattern Anal. Mach. Intell. **31**(4), 637–648 (2009)
20. Couprie, M., Bezerra, F.N., Bertrand, G.: Topological operators for grayscale image processing. J. Electron. Imaging **10**(4), 1003–1015 (2001)
21. Daragon, X., Couprie, M.: Segmentation topologique du neo-cortex cérébral depuis des données IRM. In: Proc. Congrès RFIA, vol. 3, pp. 809–818 (2002)
22. Dokládal, P., Bloch, I., Couprie, M., Ruijters, D., Urtasun, R., Garnero, L.: Segmentation of 3D head MR images using morphological reconstruction under constraints and automatic selection of markers. Pattern Recognit. **36**, 2463–2478 (2003)
23. Fabbri, R., Costa, L.D.F., Torelli, J.C., Bruno, O.M.: 2D Euclidean distance transform algorithms: a comparative study. ACM Comput. Surv. **40**(1), 1–44 (2008)
24. Fourey, S., Malgouyres, R.: A concise characterization of 3D simple points. Discrete Appl. Math. **125**(1), 59–80 (2003)
25. Hall, R.W.: Tests for connectivity preservation for parallel reduction operators. Topol. Appl. **46**(3), 199–217 (1992)

26. Kong, T.Y.: On the problem of determining whether a parallel reduction operator for n-dimensional binary images always preserves topology. In: Vision Geometry II. Proc. SPIE, vol. 2060, pp. 69–77 (1993)
27. Kong, T.Y.: On topology preservation in 2D and 3D thinning. Int. J. Pattern Recognit. Artif. Intell. **9**, 813–844 (1995)
28. Kong, T.Y.: Topology-preserving deletion of 1's from 2-, 3- and 4-dimensional binary images. In: Proc. DGCI. Lecture Notes in Computer Science, vol. 1347, pp. 3–18 (1997)
29. Kong, T.Y., Rosenfeld, A.: Digital topology: introduction and survey. Comput. Vis. Graph. Image Process. **48**, 357–393 (1989)
30. Kovalevsky, V.A.: Finite topology as applied to image analysis. Comput. Vis. Graph. Image Process. **46**, 141–161 (1989)
31. Ma, C.M.: On topology preservation in 3D thinning. Comput. Vis. Graph. Image Process. **59**(3), 328–339 (1994)
32. Malandain, G., Bertrand, G., Ayache, N.: Topological segmentation of discrete surfaces. Int. J. Comput. Vis. **10**(2), 183–197 (1993)
33. Ronse, C.: Minimal test patterns for connectivity preservation in parallel thinning algorithms for binary digital images. Discrete Appl. Math. **21**(1), 67–79 (1988)
34. Rosenfeld, A.: Digital topology. Am. Math. Mon. **86**, 621–630 (1979)
35. Rosenfeld, A., Pfaltz, J.L.: Distance functions on digital pictures. Pattern Recognit. **1**, 33–61 (1968)
36. Rueda, A., Acosta, O., Couprie, M., Bourgeat, P., Fripp, J., Dowson, N., Romero, E., Salvado, O.: Topology-corrected segmentation and local intensity estimates for improved partial volume classification of brain cortex in MRI. J. Neurosci. Methods **188**(2), 305–315 (2010)
37. Rutovitz, D.: Pattern recognition. J. R. Stat. Soc. **129**, 504–530 (1966)
38. Stefanelli, S., Rosenfeld, A.: Some parallel thinning algorithms for digital pictures. J. Assoc. Comput. Mach. **18**(2), 255–264 (1971)
39. Talbot, H., Vincent, L.: Euclidean skeletons and conditional bisectors. In: Proc. VCIP'92. Proc. SPIE, vol. 1818, pp. 862–876 (1992)
40. Whitehead, J.H.C.: Simplicial spaces, nuclei and m-groups. Proc. Lond. Math. Soc. **45**(2), 243–327 (1939)

Chapter 4
Modeling and Manipulating Cell Complexes in Two, Three and Higher Dimensions

Lidija Čomić and Leila De Floriani

Abstract Cell complexes have been used in geometric and solid modeling as a discretization of the boundary of 3D shapes. Also, operators for manipulating 3D shapes have been proposed. Here, we review first the work on data structures for encoding cell complexes in two, three and arbitrary dimensions, and we develop a taxonomy for such data structures. We review and analyze basic modeling operators for manipulating complexes representing both manifold and non-manifold shapes. These operators either preserve the topology of the cell complex, or they modify it in a controlled way. We conclude with a discussion of some open issues and directions for future research.

4.1 Introduction

Cell and simplicial complexes are the most common way to discretize geometric shapes, such as static and dynamic 3D objects, or surfaces and hyper-surfaces describing the behavior of scalar, or vector fields. Representations for these complexes are at the heart of modeling and simulation tools in a variety of application domains, such as computer graphics, Computer Aided Design (CAD), finite element analysis, animation, scientific visualization, and geographic data processing.

Historically, data structures for representing 3D shapes have been developed in the framework of solid modeling. There have been two approaches to the modeling of solid objects in $\mathbb{E}^3$, the *boundary-based representation* and the *object-based volumetric representation*. A *boundary-based representation* consists of a description of a 3D object in terms of its bounding surfaces, which are decomposed into a collection of faces, edges and vertices forming a cell 2-complex. It is the most common representation for 3D objects. The first data structure for boundary representation

L. Čomić (✉)
Faculty of Technical Sciences, University of Novi Sad, Trg D. Obradovića 6, Novi Sad, Serbia
e-mail: comic@uns.ac.rs

L. De Floriani
Department of Computer Science, University of Genoa, via Dodecaneso 35, Genoa, Italy
e-mail: deflo@disi.unige.it

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_4,

is the *winged-edge data structure*, proposed by Baumgardt in 1972 for manifold shapes [3]. The first data structure for arbitrary cell 2-complexes is the *radial edge* data structure, proposed by Weiler in 1988 [40]. This has been only the starting point for the development of a variety of representations, which are at the basis of current solid modeling systems [27]. An *object-based volumetric representation* describes a solid object based on a decomposition into volumetric cells, and thus through a cell 3-complex. Simplicial complexes are a special class of cell complexes. They have been widely used in many application fields. A variety of data structures specific for simplicial complexes have been proposed in the literature, and data structures designed for cell complexes can also be used for the representation of simplicial complexes. For a review of data structures specific for simplicial complexes, see [10].

Here, we focus on

- data structures for representing cell complexes, and
- manipulation operators for modifying these representations.

We review, analyze and compare data structures for cell complexes used for modeling the boundary or the interior of 3D solid objects. We consider also dimension-independent representations, and classify such data structures on the basis of the dimension of the complex and on the topology of the shape discretized by the complex. The comparison among the various data structures is performed in terms of their expressive power and of the efficiency and effectiveness of navigation operations on them (i.e., efficiency in retrieving topological relations not explicitly encoded in the data structure).

We review the operators that modify topological representations of cell complexes. The literature on operators for building and updating cell complexes is vast but quite disorganized. We distinguish here between operators for simplicial and for cell complexes. We briefly review the former (vertex split/edge collapse and stellar operators) and we focus on the latter. We review topological operators designed for building and updating data structures representing cell complexes (Handlebody operators, Euler operators). Handlebody operators are based on handlebody theory, stating that any n-manifold can be obtained from an n-ball by attaching handles to it. The main characteristic of Euler operators is that they maintain the Euler-Poincaré formula. Here, we will focus mainly on Euler operators for cell 2- and 3-complexes. Euler operators are part of the variety of basis operators for modeling cell complexes in a topologically consistent manner proposed in the literature. There has been no systematic and uniform treatment of these operators in the literature.

The remainder of this chapter is organized as follows. Section 4.2 provides some background notions on cell complexes and topological relations between its cells. Section 4.3 presents a taxonomy of data structures for cell complexes. Section 4.4 reviews and compares dimension-independent data structures. Section 4.5 reviews and compares data structures specific for two-dimensional complexes. Due to the large number of data structures in this category, we distinguish between data structures for manifold and non-manifold cell complexes, addressed respectively in Sects. 4.5.1 and 4.5.2. Section 4.6 reviews and compares data structures

for three-dimensional cell complexes. Section 4.7 presents an overview for the update operators on cell complexes. Section 4.8 discusses the Euler-Poincaré formula for various classes of complexes, and presents a classification of Euler operators. Sections 4.9 and 4.10 review Euler operators on manifold and non-manifold complexes, respectively. Finally, Sect. 4.11 summarizes the presented work, discusses some open problems and concludes with future work directions.

4.2 Background Notions

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

A *k-cell* in the Euclidean space $\mathbb{E}^n$ is a homeomorphic image of a k-dimensional ball, and a *cell complex* in $\mathbb{E}^n$ is a finite set Γ of cells in $\mathbb{E}^n$ of dimension at most d, $0 \leq d \leq n$, such that

1. the cells in Γ are pairwise disjoint,
2. for each cell $\gamma \in \Gamma$, the boundary of γ is a disjoint union of cells of Γ.

If the maximum dimension of cells of Γ is equal to d, then Γ is called a *d-dimensional complex*, or simply a *d-complex*. The set of the cells on the boundary of a cell γ is called the (*combinatorial*) *boundary* of γ. The (*combinatorial*) *coboundary* (or *star*) of γ consists of all cells of Γ that have γ on their combinatorial boundary. An h-cell γ' on the boundary of a k-cell γ, $0 \leq h \leq k$, is called an *h-face* of γ, and γ is called a *coface* of γ'. Each cell γ is a face of itself. If $\gamma' \neq \gamma$, then γ' is called a *proper face* of γ, and γ and γ' are said to be *incident*. The *link* of a cell γ is defined as the collection of the cells bounding the cells in the star of γ, which do not intersect γ. A cell is called a *top cell* if it is not on the boundary of any other cell in Γ. The *domain*, or *carrier*, of a Euclidean cell d-complex Γ embedded in E^n, with $0 \leq d \leq n$, is the subset of E^n defined by the union, as point sets, of all the cells in Γ.

Two p-cells, $0 < p \leq d$, are said to be *adjacent* if they share a $(p-1)$-face. Two vertices (i.e., 0-cells) are called *adjacent* if they are both incident in a common 1-cell. A d-complex Γ, in which all top cells are d-cells, is called *uniformly d-dimensional* or *homogeneous*. A (*combinatorial*) *pseudo-manifold* is a uniformly d-dimensional complex in which each $(d-1)$-cell is shared by one or two d-cells, and for any two d-cells γ and γ', there is a sequence $\gamma = \gamma_1, \gamma_2, \ldots, \gamma_n = \gamma'$ of d-cells, and any two consecutive d-cells in the sequence share a $(d-1)$-cell. A pseudo-manifold complex with a manifold domain is called a *manifold complex*. Figure 4.1(a) shows an example of a uniformly d-dimensional complex, which is not a pseudo-manifold, while Figs. 4.1(b) and (c) show an example of a pseudo-manifold complex which is not a manifold complex.

Simplicial complexes can be seen as a subclass of cell complexes. Their cells, called *simplices*, are defined as the convex combination of points in the Euclidean space. A Euclidean *simplex* σ of dimension k is the convex hull of $k+1$ affinely

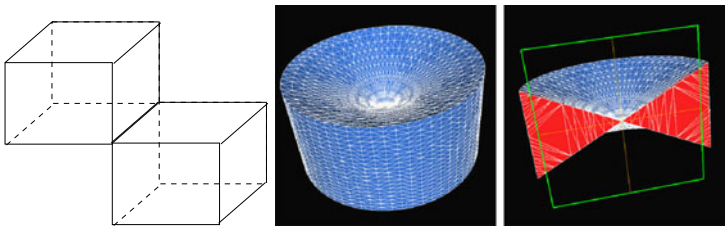


Fig. 4.1 (**a**) A homogeneous cell complex that is not manifold; (**b**) A pseudo-manifold with a non-manifold domain (a 3D pinched pie); (**c**) The cross-section of the pinched pie at the non-manifold vertex

independent points in the n-dimensional Euclidean space E^n, for $0 \leq k \leq n$. We call a *Euclidean simplex* of dimension k a *k-simplex*, and k is called the *dimension* of the simplex.

A $(d-1)$-cell γ in a uniformly d-dimensional cell complex Γ is called a *manifold cell* if and only if γ is incident in at most two d-cells in Γ. Otherwise, $(d-1)$-cell γ is a *non-manifold cell*. In a cell complex embedded in $\mathbb{E}^3$, there can be no non-manifold 2-cells (or 3-cells). A vertex (0-cell) v in a cell (simplicial) d-complex Γ (with $1 \leq d \leq 3$) is a *manifold* vertex if and only if the link of v in Γ is homeomorphic to a triangulation of the $(d-1)$-sphere S^{d-1}, or of the $(d-1)$-disk B^{d-1}. A vertex is called *non-manifold* otherwise. An edge (1-cell) e in a d-complex Γ (with $2 \leq d \leq 3$) is a *manifold* edge if and only if the link of e in Γ is homeomorphic to a triangulation of the $(d-2)$-sphere S^{d-2}, or of the $(d-2)$-disk B^{d-2}. An edge is called *non-manifold* otherwise.

The connectivity information among the entities in a cell complex can be expressed through *topological relations*, which provide an effective framework for defining, analyzing and comparing the wide spectrum of existing data structures [7]. For a p-cell γ in a d-dimensional cell complex Γ, $0 \leq p \leq d$, *topological relations* are defined as follows:

- *Boundary relation* $R_{p,q}(\gamma)$, with $0 \leq q \leq p-1$, consists of the set of q-cells in Γ that are faces of γ.
- *Co-boundary relation* $R_{p,q}(\gamma)$, with $p+1 \leq q \leq d$, consists of the set of q-cells in Γ that have γ as a face.
- *Adjacency relation* $R_{p,p}(\gamma)$, with $0 < p \leq d$, consists of the set of p-cells in Γ that share a $(p-1)$-cell with γ.
- *Adjacency relation* $R_{0,0}(\gamma)$, where γ is a vertex, consists of the set of vertices in Γ that are adjacent to γ through a 1-cell (an edge).

Figure 4.2 illustrates topological relations: $R_{2,1}(f)$ is the set of edges bounding 2-cell f (see Fig. 4.2(a)), relation $R_{0,1}(v)$ is the set of edges incident in vertex v (see Fig. 4.2(b)), relation $R_{2,2}(f)$ consists of the set of 2-cells which share an edge (1-cell) with 2-cell f (see Fig. 4.2(c)). Boundary and co-boundary relations are called *incidence relations*.

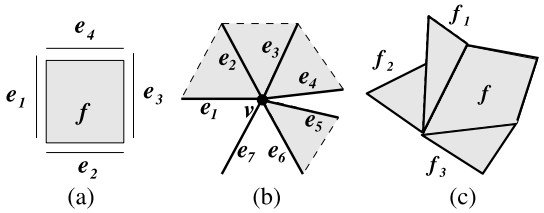


Fig. 4.2 Examples of topological relations: boundary relation $R_{2,1}(f) = \{e_1, e_2, e_3, e_4\}$ for 2-cell f (**a**); co-boundary relation $R_{0,1}(v) = \{e_1, \ldots, e_7\}$ for vertex v (**b**); adjacency relation $R_{2,2}(f) = \{f_1, f_2, f_3\}$ for 2-cell f (**c**)

In the framework of geometric and solid modeling, the notion of a cell complex is usually considered to be too restrictive, and not powerful and versatile enough to satisfy all the demands required from a modeling system and to model characteristics of an object that arise in real designing applications. In some approaches proposed in the literature for modeling boundary representations (manifold surfaces that bound a solid object in $\mathbb{E}^3$), this drawback is overcome by allowing faces (which correspond to 2-cells) to be multiply-connected, but mappable to a plane. Such faces have no genus, and are bounded by several connected components of edges, usually called *loops* or *rings*. In some solid modeling applications, volumes (which correspond to 3-cells) are allowed to have through holes or cavities. We call the complexes, in which cells are not necessarily homeomorphic to a ball, (*general*) *complexes*.

4.3 Data Structures for Cell Complexes: An Overview

We classify the data structures for cell complexes in terms of:

1. *the domain* of the complexes they represent: manifold, pseudo-manifold, uniformly d-dimensional, etc.
2. *the dimension*: dimension-independent data structures can describe cell complexes in any dimension, while dimension-specific data structures are for 2D and 3D cell complexes embedded in the three-dimensional Euclidean space $\mathbb{E}^3$.
3. *the topological information encoded*: in a cell complex, the basic topological entities are the cells. A data structure may encode all the cells of a complex, or only a subset of them.
4. *the way topological information is encoded*: some data structures encode the cells and their topological relations *explicitly*. In such data structures, the cells are *entities* and the relations are associated with the entities. *Implicit* data structures encode the relations among cells indirectly, through tuples of cells in the same relation.

We organize the description of the various data structures on the basis of the dimension of the complex they represent. We present a description of each data structure in terms of the entities and topological relations it encodes, and we discuss

it based on its expressive power and on the efficiency in supporting navigation inside the complex (i.e., in retrieving topological relations not explicitly encoded). In explicit data structures, topological queries are based on cells. In contrast, in implicit data structures, navigation is typically performed by considering tuples of cells as the atomic units.

For the sake of brevity, we review only data structures for cell complexes. A review of data structures for simplicial complexes can be found in [10].

4.4 Dimension-Independent Data Structures

In this section, we discuss dimension-independent data structures for cell complexes. We review first two dimension-independent implicit representations for manifold shapes, namely the *Cell-Tuple* [6] and the *n-G-map* [23] data structures, and the *Incidence Graph* (*IG*) [13], a general data structure for arbitrary cell complexes in any dimension.

A *Cell-Tuple* [6] is a representation for Euclidean cell complexes with a manifold domain, while the *n-G-map* [23] has been developed for abstract cell complexes belonging to the class of quasi-manifolds, which is a superclass of combinatorial manifolds. In essence, however, the Cell-Tuple and the n-G-map data structures are equivalent in terms of the entities and relations they encode. Here, we describe, for brevity, only the Cell-Tuple data structure.

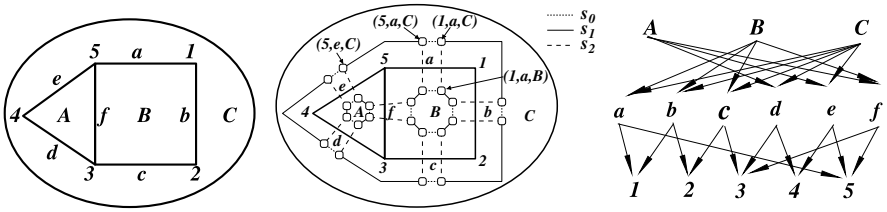
Given a Euclidean d-dimensional cell complex, a *cell-tuple* is a $(d+1)$-tuple t of $d+1$ cells, $t = (c_0, c_1, \ldots, c_d)$, such that c_i is an i-cell on the boundary of cells c_{i+1} to c_d. A function s_i for $i = 0, \ldots, d$, called a *switch function*, is defined on the cell-tuples such that $t' = s_i(t)$ if the (unique) cell-tuple t' is identical to t in every element except the i-th one. The functions s_i partition the set of cell-tuples into equivalence classes of size 2 each. The s_i functions have the following two properties:

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

Figure 4.3(a) provides a simple example of a cell complex defined on a surface without boundary. The cell complex is composed of two internal 2-cells A and B, and of the 2-cell C. Figure 4.3(b) shows all the tuples in small squares, and all the s_i $(i = 0, 1, 2)$ functions. Two tuples are related by function s_0 if they are connected through a dotted line, by s_1 if connected by a thin solid line, or by s_2 if connected through a dashed line.

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

It can be shown that all topological relations can be retrieved efficiently from the Cell-Tuple data structure. As an example, consider the retrieval of relation $R_{0,2}(5)$



(a) (b) (c)

Fig. 4.3 A simple cell complex on a surface homeomorphic to a sphere. The complex is composed of triangle A, square B and the external 2-cell C on the surface (**a**). All the tuples and all the switch functions s_i $(i = 0, 1, 2)$ encoded by the Cell-Tuple data structure (**b**). All the boundary relations encoded in the IG (**c**)

for vertex 5 in Fig. 4.3, which consists of all the 2-cells that are incident in vertex 5. The retrieval starts with any of the tuples that include vertex 5, such as $(5, a, C)$. By alternately applying functions s_2 and s_1 to each new tuple visited, the cyclic sequence $(5, a, C)$, $(5, a, B)$, $(5, f, B)$, $(5, f, A)$, $(5, e, A)$, $(5, e, C)$ is obtained, which produces the set of 2-cells $\{A, B, C\}$ that are incident in vertex 5.

The *Incidence Graph* (*IG*) [13] is an incidence-based explicit data structure for cell complexes. The incidence relations among cells that differ in dimension by one are explicitly encoded. Formally, the IG encodes all the cells of a cell d-complex Γ, and for each p-cell γ in Γ, its immediate boundary, and immediate co-boundary relations, namely:

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

Figure 4.3(c) gives an example that illustrates the boundary relations encoded in the IG in the form of a directed graph.

The design of the IG supports a simple recursive strategy to retrieve boundary and co-boundary relations. Boundary relation $R_{p,q}(\gamma)$ $(p > q)$ for a given p-cell γ is obtained by retrieving the encoded boundary $R_{i,i-1}$ relations of all the i-faces for $i = p, \ldots, q + 1$ of γ. Co-boundary relation $R_{p,r}(\gamma)$ $(p < r)$ is obtained by retrieving the encoded co-boundary $R_{i,i+1}$ relations of all the i-cells for $i = p, \ldots, r - 1$ in the star of γ. The retrieval of any relation for a cell γ can be done in time linear in the number of cells in the star or on the boundary of cell γ.

The IG has been used as the underlying framework for the implementation of Selective Geometric Complexes (SGCs) [33], which describe arbitrary-dimensional non-manifold objects through collections of mutually disjoint (not necessarily topological) cells defined as open subsets of d-manifolds. A simplified and more space efficient versions of the incidence graph for simplicial complexes in arbitrary dimension have been proposed in [9] and [8]. Another concise instance of the Incidence graph for cell 2-complexes is the *Adjacency and Incidence Framework* (*AIF*) [36].

4.5 Data Structures for Two-Dimensional Cell Complexes

In this section, we discuss data structures for two-dimensional cell complexes embedded in the 3D Euclidean space. We classify them according to the taxonomy introduced in Sect. 4.3, and organize their description in two subsections according to the domain of the complexes (manifold or non-manifold).

4.5.1 Representing Manifold 2-Complexes

Here, we briefly review the *Star-Vertex* [19], the *Winged-Edge* [3], the *Doubly-Connected Edge List* (*DCEL*) [31], the *Half-Edge* [27] the *Quad-Edge* [16] and the *Lath-based* [18] data structures. The Winged-Edge, DCEL and the Half-Edge data structures are all edge-based representations, since they represent the edge as the primary entity and the relations around it. The Quad-Edge and the Lath-based data structures are implicit representations. A thorough analysis and comparison of data structures for manifold cell 2-complexes can be found in [34].

The *Star-Vertex* (*SV*) data structure [19] is an adjacency-based data structure for manifold planar cell 2-complexes. In terms of topological relations, the Star-Vertex data structure encodes relation $R_{0,0}$ explicitly. Relation $R_{2,0}(f)$ is partially encoded since face f is implicitly described through one of its vertices. The Star-Vertex data structure only supports the retrieval of $R_{2,0}$ relation and $R_{0,0}$ relation efficiently. Co-boundary relations cannot be retrieved locally. In terms of storage cost, it is the most economical of the data structures for cell 2-complexes, but it does not have the full navigation capability as the other data structures.

The *Winged-Edge* (*WE*) data structure [4] is the first one proposed for two-dimensional cell complexes. It encodes: (i) for each edge e, its two vertices, the two faces incident in e, and the four edges that are both adjacent to e and are on the boundary of the two faces incident in e (see Fig. 4.4(a)); (ii) for each face f, a reference to one edge on the boundary of f; (iii) for each vertex v, a reference to one edge incident in v. It supports the retrieval of all topological relations efficiently. The cells in the star of a vertex or on the boundary of a face can be traversed in both clockwise or counterclockwise direction.

The *Doubly-Connected Edge List* (*DCEL*) data structure [31] is a simplified version of the WE representation, though it has been developed independently. For each edge e, instead of encoding all four edges on the boundary of the two faces incident in e (see Fig. 4.4(b)), it stores only two edges, one for each of the two faces incident in e. The DCEL supports the traversal of all topological relations, but only in counterclockwise direction in the star of a vertex, and in clockwise direction around the boundary of a face.

The *Half-Edge* (*HE*) data structure [27] encodes two copies of each edge, each of which is called a *half-edge*. A half-edge has a direction with respect to the face which it bounds. For each half-edge, the following information is encoded: its start vertex, the face associated with it, the previous and the next edges on the same face,

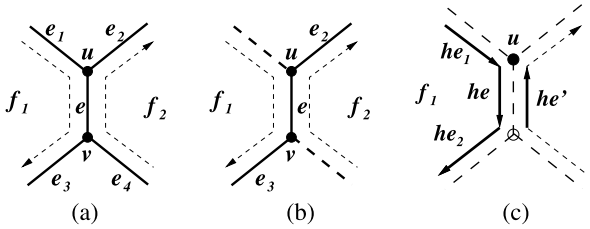


Fig. 4.4 Edge-based relations represented in the edge-based data structures: (**a**) In the Winged-Edge data structure, e has a reference to e_1, e_2, e_3, e_4, u, v, f_1 and f_2. (**b**) In the DCEL, e has a reference to e_2, e_3, u, v, f_1 and f_2. (**c**) In the Half-Edge data structure, each edge e is represented as two half-edges he and he'. Half-edge he has a reference to he', he_1, he_2, u and f_1

the companion half-edge (see Fig. 4.4(c)). Relations encoded at vertices and faces are the same as those encoded in the Winged-Edge and DCEL data structures. The Half-Edge data structure supports the efficient retrieval of all topological relations, and also the cells in the star of a vertex or on the boundary of a face can be traversed in both clockwise or counterclockwise directions. The *Handle-Edge* data structure in [24] extends the Half-Edge data structure to manifold surfaces with boundary. It includes also an explicit representation of vertices, edges and faces.

All the edge-based data structures presented in this section encode, for each edge, relations $R_{1,0}$ and $R_{1,2}$, different partial $R^*_{1,1}$ relations, since only two or four edges adjacent to a given edge e are encoded, a partial $R^*_{0,1}$ relation for each vertex, which consists of one edge in the star of the vertex, and a partial $R^*_{2,1}$ relation for each face, which consists of one edge on the boundary of the face. All these data structures support the efficient retrieval of all topological relations.

The *Quad-Edge* data structure [16] and the *Lath-based* data structure [18] are implicit data structures for cell 2-complexes with a manifold domain. In such complexes, edges in the star of each vertex can be ordered radially around the vertex, and the edges on the boundary of a face can be ordered clockwise or counterclockwise around the face. Thus, each edge belongs to four loops: the two at its extreme vertices, and the two at the faces sharing it. Such representations exploit this property.

In the Quad-Edge data structure, each *quad-edge* is associated with its two extreme vertices, its two adjacent faces and the next edges in its four loops. Basically, the Quad-Edge data structure encodes the same information as the Winged-Edge data structure. In a quad-edge that corresponds to edge e, the four adjacent edges of e are organized as part of the two loops around two faces, and two loops around two vertices. In the Winged-Edge data structure, the same four edges belong to the two loops of the two faces. Relations encoded at vertices and faces are the same as in the Winged-Edge data structure. As the other edge-based data structures, the Quad-Edge data structure encodes partial relations $R^*_{2,1}$ for each face, partial relation $R^*_{0,1}$ for each vertex, complete relations $R_{1,0}$, $R_{1,2}$ and a partial relation $R^*_{1,1}$ for each edge. As in the other edge-based representations, all topological relations can be retrieved efficiently.

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

In the *Split-Edge* data structure, each lath corresponds to one side of an edge and encodes a link to its start vertex, a link to the lath of the other side of the same edge, and a link to the lath of the next edge in the clockwise direction on the same face. In the *Half-Edge-Lath* data structure, each lath is associated with half of an edge. It encodes a link to its vertex, a link to the lath of the other half of the same edge, and a link to the lath of the next edge in the clockwise direction around the same vertex. In the *Corner* data structure, a lath is associated with one corner of a vertex. Each lath encodes: a link to the associated vertex, a link to the lath of the next vertex in the clockwise direction on the same face, and a link to the lath of the next face in the clockwise direction around the same vertex.

All Lath-based data structures support the retrieval of all topological relations through laths in time linear in the entities involved in the relations. However, because of the implicitness of the faces, edges and vertices, access from these cells to their associated laths is not efficient.

4.5.2 Representing Non-manifold 2-Complexes

We review data structures for non-manifold shapes discretized as cell complexes. The first data structure proposed in the literature for 2-complexes is the *Radial Edge* (*RE*) data structure [40], which has been extended and specialized in [17, 44]. More recent simplified representations are the *Partial Entities* (*PE*) data structure [21] and the *Loop Edge-use* (*LE*) data structure [30]. The PE data structure has the same representation power as the RE, but it is considerably more compact. The LE data structure is a specialization of the RE data structure to encode uniformly 2-dimensional cell complexes. The *Handle-Cell* data structure [32] is based on the concept of stratification (the decomposition of the boundary of a non-manifold 3D shape into manifold parts). We describe here the Radial-Edge data structure, its variant the Partial-Entity data structure, and the Handle-Cell data structure.

The *Radial Edge* (*RE*) data structure [40] has been developed to describe the decomposition of the boundary of non-manifold three-dimensional objects, composed of parts of different dimensions. This decomposition is not a cell complex as defined in algebraic topology, since the 2-cells are not necessarily homeomorphic to 2-balls, but they can be multiply connected 2-manifolds with boundary, mappable to a plane. The connected components formed by the edges bounding any 2-cell (face) are called *loops* or *rings*. The entities in the RE data structure are thus: regions,

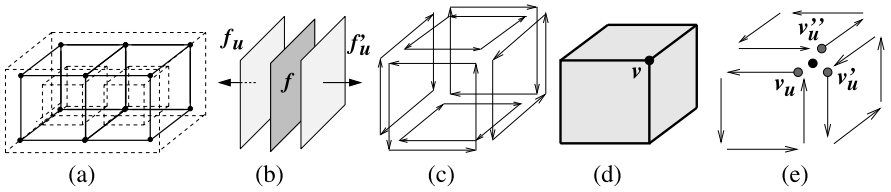


Fig. 4.5 An illustration of the entities in the RE data structure: Three shells exist in a complex describing two hollow cubes sharing a common face (**a**); Each face f has two face-uses f_u and f'_u (**b**); Each face-use on the inner shell of a cube is bounded by a loop-use which is composed of a circular list of edge-uses (**c**); Vertex v shared by three faces (**d**), and the vertex-uses v_u, v'_u and v''_u that start at v (**e**)

shells, faces, loops, edges and vertices. A *region* is a solid object, which is bounded by a collection of shells. (In Fig. 4.5(a), there are three shells on a shape of two hollow cubes sharing a face.) A *shell* is an oriented boundary surface of a region, consisting of maximal connected sets of faces. In addition, faces, loops, edges and vertices are characterized by orientations, namely *face-uses*, *loop-uses*, *edge-uses* and *vertex-uses*. A face f has two face-uses associated with it, which correspond to the two possible orientations of f (see Fig. 4.5(b)). The oriented boundary of a face-use is described by loop-uses. A loop-use is composed of a circular list of edge-uses. Each edge-use associates an edge e with the orientation induced on e by the face-use to which it belongs. Thus, an edge-use represents the association between an edge and a face-use (see Fig. 4.5(c)). A top 1-simplex, called a *wire-edge*, is described by two edge-uses forming a loop that connects the two boundary vertices of the wire-edge. Since each edge is bounded by two vertices, each edge-use is associated with a vertex-use. Thus, a vertex-use describes the association between a vertex and an edge-use that originates from it (see Figs. 4.5(d) and (e)).

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

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

- for each face f: partial relation $R^*_{2,1}(f)$, consisting of one edge on the boundary of f,
- for each edge e:
  - relation $R_{1,2}(e)$, in which the faces are ordered around e;
  - partial relation $R^*_{1,1}(e)$, defined as the collection of the pair of edges adjacent to e and bounding the faces incident in e, ordered around e, so that both the

$2i$-th element and the $(2i+1)$-element in this relation belong to the i-th face in $R_{1,2}(e)$;
  - relation $R_{1,0}(e)$, ordered by the indices of the vertices;
- for each vertex v: relation $R_{0,1}(v)$, unordered.

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

The primary difference between the RE and the *Partial Entities* (*PE*) *data structure* [21] is that the PE data structure considers each face to have one orientation geometrically defined based on its face normal. The orientation of its boundary can thus be uniquely defined. In the RE data structure, a face entity is without orientation. The face-uses of the RE data structure describe all the possible orientations of each face. In the RE data structure, the connectivity among the faces, edges and vertices is defined through face-uses, edge-uses and vertex-uses. In the PE data structure, the connectivity among faces, edges and vertices is captured at the faces, at partial-edges and at partial-vertices. The PE data structure encodes the same relations as the RE data structure, with the exception of $R_{0,1}(v)$ relation at vertex v, which is partially encoded in the PE, but fully encoded in the RE. All topological relations can be retrieved efficiently from the PE data structure. In [21], an implementation of the PE data structure is presented that has half the storage cost of the RE data structure for non-manifold cell 2-complexes, and uses twice as much space as that of the winged-edge data structure for manifold cell 2-complexes.

Another way to represent non-manifold shapes consists of decomposing them into manifold, or nearly-manifold, components, called *strata* (see [41] for stratifications of analytic sets). In [32], a *combinatorial stratification* of a cell 2-complex Γ describing the boundary of a 3D non-manifold shape is defined as a collection of k-dimensional connected combinatorial manifolds $S = \{M_1, \ldots, M_n\}$ $(k = 0, 1, 2)$ with or without boundary such that the union $\bigcup_i M_i$ gives Γ and each non-empty intersection between two elements M_i and M_j in S is a sub-complex of both M_i and M_j. The resulting set of strata and their connectivity provides a description of the original shape which is used as the basis for the *Handle-Cell* (*HC*) data structure. It consists of the *global cells*, which are the cells of Γ, and the *local cells*, which are the cells that describe the strata (points, curves and surfaces). The connectivity among the strata is captured through the sharing of global cells (vertices and edges).

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

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

  - relation $R_{1,2}(e)$, which consists of all faces incident in e;
  - partial relation $R^*_{1,1}(e)$, ordered around edge e, so that both the $2i$-th element and the $(2i+1)$-element in this relation are on the i-th face of e;
  - relation $R_{1,0}(e)$, which consists of the extreme vertices of edge e;
- for each vertex v: relation $R_{0,1}(v)$, which consists of the set of edges incident in v.

Since strata are 2-complexes with a manifold domain, surfaces are represented through the Half-Edge data structure. This is conceptually similar to the representation of the surfaces of the 3-cells in the Handle-Face (HF) data structure (see Sect. 4.6). The Handle-Cell data structure is also similar to the data structures for non-manifold 2-complexes, such as the Radial Edge data structure and Partial Entities data structure. The primary difference is that in the Handle-Cell data structure stratification is explicitly encoded.

4.6 Data Structures for Three-Dimensional Cell Complexes

In this section, we discuss representations for three-dimensional cell complexes embedded in the three-dimensional Euclidean space $\mathbb{E}^3$. There are relatively few representations for describing 3D shapes discretized as a cell 3-complex. Most of such representations are limited to the manifold domain. Representations for manifold cell complexes are the *Facet-Edge* [11] and the *Handle-Face* [25] data structures.

The *Facet-Edge* (*FE*) *data structure* [11] is a 3D extension of the quad-edge data structure developed for cell 2-complexes (see Sect. 4.5.1), and thus it is an implicit representation for manifold cell 3-complexes. The basic entities encoded in the Facet-Edge data structure are the vertices and the so-called *facet-edges* defined on the 2-cells and 1-cells (faces and edges). The three-dimensional cells and their topological information are encoded through the topological vertex-based relations of the dual complex to the given one. The 0-cells of the dual complex correspond to the 3-cells of the original complex, its 1-cells to the faces, its 2-cells to the edges and its 3-cells to the vertices.

The boundary of each face f contains a ring of edges $e_1, \ldots, e_n$. This ring is called a *face-ring*, and may be ordered in two directions. The star of an edge e contains a ring of faces $f_1, \ldots, f_m$. This ring is called an *edge-ring* and can also be ordered in two directions. A *facet-edge* pair uniquely associates a face f with an edge e on the boundary of f. Each facet-edge pair exists in four versions. Each version is associated with exactly one face-ring of f and exactly one edge-ring of e. Each version of a facet-edge has its dual which is defined in the dual complex. For the cell complex in Fig. 4.6(a), the four edges in the face-ring of face f_1 are shown in Fig. 4.6(b), and the four facet-edges formed by face f_1 and edge e_1 are shown in Figs. 4.6(c)–(f).

The Facet-Edge data structure describes a complex Γ by encoding, for each facet-edge pair a associated with edge e in a face-ring and with face f in an edge-

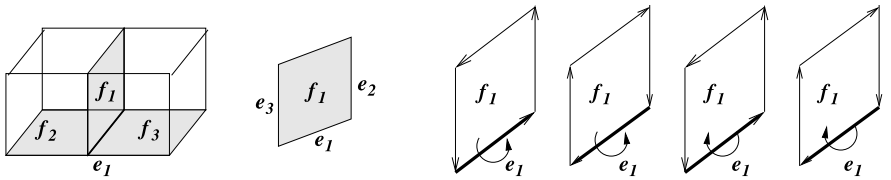


Fig. 4.6 An illustration of the concept of facet-edge: a model of two cubes (**a**); three edges incident in face f_1 (**b**); the four facet-edge pairs formed by face f_1 and edge e_1 (**c**–**f**)

ring, the preceding and succeeding facet-edges in both the face-ring and the edge-ring of a. For the example in Fig. 4.6(a), the successors of (f_1, e_1) in the face-ring of e_1 in both directions are respectively (f_2, e_1) and (f_3, e_1). In Fig. 4.6(b), the successors of (f_1, e_1) in both directions in the edge-ring of f_1 are (f_1, e_2) and (f_1, e_3).

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

The *Handle-Face* (*HF*) *data structure* [25] is an explicit representation for manifold cell 3-complexes. It is similar to the RE and the PE data structures (see Sect. 4.5.2), since the 3-cells of the 3-complex are described in the HF data structure through their boundaries, made by faces, edges and vertices. The HF data structure contains two types of entities: the *basic entities*, which are the *faces*, *edges* and *vertices* in the cell complex, and the *surface entities*, which describe the boundary of each 3-cell of the complex. The HF data structure further distinguishes between surface entities that are on the boundary of the entire manifold shape and surface entities that are in the interior. Here we present a simplified version of the HF data structure in which the surface entities simply describe the boundary of each 3-cell. The surface entities are *surfaces*, *surface edges* and *surface vertices*.

Figure 4.7 illustrates the entities in the HF data structure. Figure 4.7(a) shows all the faces, edges and vertices in a cell complex that is composed of two 3-cells. Figure 4.7(b) shows the half-faces, surface-edges and surface-vertices on the surface of each 3-cell of Fig. 4.7(a). Figure 4.7(c) shows the surface-oriented edges bounding each half-face shown in Fig. 4.7(b).

The HF data structure encodes the following topological relations:

- For each face f, partial $R^*_{2,1}(f)$ relation, which encodes one edge bounding face f;
- For each edge e, relation $R_{1,2}$, which encodes all faces around e, radially ordered around e; partial $R^*_{1,1}(e)$ relation, which encodes for each face f incident in e the edge following e on the boundary of f; and relation $R_{1,0}(e)$, which encodes the two vertices incident in edge e;

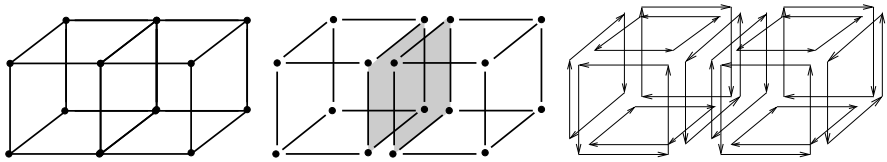


Fig. 4.7 A 3-complex with two cubic 3-cells, composed of 11 faces, 20 edges, and 12 vertices (**a**); There are 12 half-faces, 24 surface-edges, and 16 surface vertices in the whole complex (**b**); Surface-oriented edges of the two 3-cells (**c**)

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

The 3-cells are not represented explicitly in the HF data structure. The drawback is that no attribute can be attached to the 3-cells. Also, the HF representation encodes the same relations as in the RE and PE data structures. On the other hand, unlike the RE and the PE data structures, the HF data structure cannot represent shapes with dangling edges or faces, or 3D shapes with non-manifold vertices and edges. As all representations which encode orientations by duplicating the basic entities, the HF data structure is quite verbose. All topological relations at faces, edges and vertices can be retrieved efficiently from the HF data structure, and the representation of surface entities allows retrieving the boundaries of the 3-cells even if these latter are not explicitly represented.

The *Compact Half-Face* (*CHF*) *data structure* [20] is a specialization of the HF data structure for representing manifold simplicial 3-complexes (usually called *tetrahedral meshes*).

4.7 Manipulation Operators on Cell Complexes: An Overview

Manipulation operators that we will review in the rest of the chapter can be classified based on the type of the complexes they apply to into:

- operators on simplicial complexes, such as
  - edge collapse/vertex-pair collapse;
  - stellar operators;
- operators on cell complexes, such as
  - handle operators;
  - Euler operators.

We will focus our attention on Euler operators on cell complexes. In this section, we will review edge collapse/vertex-pair collapse operators and stellar operators on simplicial complexes, and handle operators on cell complexes.

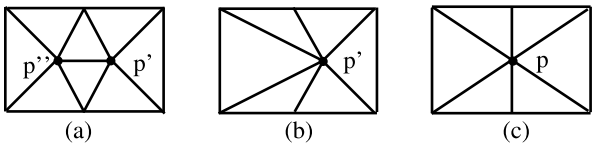


Fig. 4.8 A portion of a simplicial 2-complex with edge $e = (p', p'')$ (**a**); after half-edge collapse (**b**); after full-edge collapse (**c**)

4.7.1 Collapse Operators on Simplicial Complexes

Triangle and tetrahedral meshes (simplicial 2- and 3-complexes, usually with manifold domain) have been used for approximating 2D and 3D objects in $\mathbb{E}^3$. Simplification and modification of such meshes is an important task, aimed at reducing the size of densely sampled models, or at improving mesh quality, by obtaining meshes where the top simplexes (triangles in 2D and tetrahedra in 3D) satisfy some geometric constraints (concerning e.g. aspect ratio, size, diameter).

In the literature, there have been many approaches to the updating of simplicial complexes. The most common update operators for simplicial complexes are edge collapse, which is based on contracting an edge to one of its extreme vertices, or to a new vertex, and vertex-pair collapse, which collapses two vertices that are not connected through an edge, together with the inverse refinement operators called vertex split.

Edge collapse is a simplification operator. It contracts an edge $e = (p', p'')$ to a vertex: either to one of its endpoints, e.g. p' (half-edge collapse), or to a new vertex p (full-edge collapse). Half-edge collapse modifies the simplexes in the star of vertex p'', while full-edge collapse modifies the simplexes in the star of both vertex p' and p''. The effect of the half-edge collapse is that the n-simplexes incident in edge e become $(n-1)$-simplexes incident in vertex p'. Other simplexes incident in vertex p'' become incident in p'. The effect of the full-edge collapse is that the n-simplexes incident in edge e become $(n-1)$-simplexes incident in vertex p. Other simplexes incident in either vertex p' or p'' become incident in p. Figure 4.8 illustrates the edge collapse operator in 2D.

Vertex split is a refinement operator, inverse to edge collapse. Full-vertex split expands a vertex p into two new vertices p' and p'', and an edge $e = (p', p'')$. Half-vertex split expands a vertex p' into vertex p', new vertex p'', and edge $e = (p', p'')$.

Vertex-pair collapse merges two vertices that are not connected through an edge, and is called also *virtual edge collapse*. Its effect is to collapse the virtual edge joining the two vertices. This operator does not change the number of triangles in the mesh, but it updates the triangles incident in the collapsed vertices. This operator may merge two components into one (if the two merged vertices belong to different components), or it may create or destroy a hole (if the two merged vertices belong to the same component). Vertex-pair collapse operator is illustrated in Fig. 4.9.

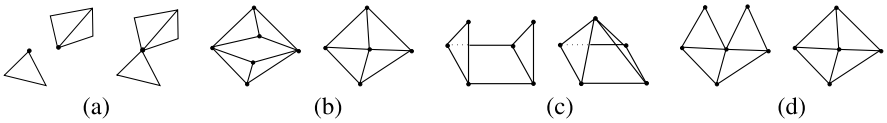


Fig. 4.9 Vertex-pair collapse operator may merge two components into one (**a**); destroy a hole (**b**); create a hole (**c**); not change the topology of the mesh (**d**)

4.7.2 Stellar Operators

A class of operators on simplicial complexes introduced in [38] are called *stellar operators*. They are motivated by the work in [22] on stellar subdivision theory. Stellar operators are defined in arbitrary dimensions. They change the local neighborhood of a simplex in a simplicial complex, but they do not affect the topology of the complex.

In [38], two types of stellar operators in the 2D case are defined, called *split* (and its inverse *weld*) and *flip*. Split (stellar subdivision) operators are refinement operators, the inverse weld operators are simplification operators. Flip (bistellar) operators change the connectivity of the simplicial complex, but they do not alter the number of simplexes in the complex. Each flip operator can be expressed through split and weld operators.

In 2D, there are two instances of the *split* operator, namely *face split* and *edge split*. *Face split* inserts a new vertex p inside a triangle t, and connects it to the three vertices of t. *Edge split* introduces a new vertex p in the interior of an existing edge e splitting it in two, and it splits also the two triangles incident in e (in the star of e) by connecting the new vertex to the vertices in the link of e. Face and edge split operators are illustrated in Figs. 4.10(a) and (b), respectively.

The inverse weld operators are simplification operators. A vertex weld operator may be applied on a vertex of degree three (incident in exactly three edges). It deletes the vertex, the three edges incident in it, and it merges the three triangles incident in the vertex in one triangle. An edge weld operator may be applied on a vertex of degree four. It deletes the vertex and two non-consecutive edges incident in it (in a cyclic order), thus merging two pairs of triangles incident in the vertex.

The edge flip operator deletes one edge e in the complex, and replaces it by another edge, which connects the two vertices in the link of the deleted edge e. The edge flip operator is illustrated in Fig. 4.10(c).

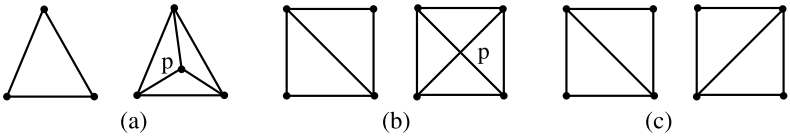


Fig. 4.10 A portion of a simplicial 2-complex before and after a face split (**a**), edge split (**b**), and edge flip operator (**c**)

4.7.3 Handle Operators

The *handle operators* on a manifold cell 2-complex Γ triangulating a surface S have been introduced in [24]. They are based on the handlebody theory for surfaces, stating that any surface S can be obtained from a 2-ball by iteratively attaching handles (0-, 1- and 2-handles).

The initialization operator in [24] corresponds to attaching a 0-handle. It creates a new surface with one face, three edges and three vertices.

There are three operators that correspond to attaching a 1-handle. They identify two boundary edges of Γ (incident in exactly one face) with no vertices in common. If the two identified edges belong to two different components of Γ, then the number of connected components and of boundary curves (connected components of boundary edges) in Γ is decreased by one. If the two identified edges belong to the same component and the same boundary curve of Γ, then the number of holes (independent 1-cycles) and the number of boundary curves in Γ is increased by 1. If the two identified edges belong to the same component and two different boundary curves of Γ, then the number of holes (independent 1-cycles) is increased by 1, and the number of boundary curves in Γ is decreased by 1.

The operator that corresponds to the attachment of a 2-handle identifies two edges on the boundary of Γ with two vertices in common. It decreases the number of holes and the number of boundary curves in Γ by 1. Another operator, which does not alter the topology of Γ, identifies two edges on the boundary of Γ that have one vertex in common.

The analogous work in 3D in [25] uses the fact that each compact orientable 3-manifold S can be obtained by iteratively attaching handles (0-, 1-, 2- and 3-handles) to a 3-ball. Operators that correspond to handle attachments are called Morse operators in [25].

Operator that creates a new 3-ball (initialization operator) corresponds to the attachment of a 0-handle. Other operators identify two boundary faces (incident in exactly one 3-cell) of a cell 3-complex Γ triangulating a solid S.

The attachment of a 1-handle can be applied in three situations: if the two identified boundary faces are on different connected components of Γ, then the two components are merged into one; if the two identified faces belong to the same boundary surface component of Γ (connected component of boundary faces) and have no edges in common, then a hole is created; if the two identified faces belong to different boundary surfaces of the same connected component of Γ, the operator can be realized only if Γ is embedded in a space of dimension greater than 3.

The attachment of a 2-handle corresponds to identifying two faces on the same boundary surface component of Γ that have some edges in common. The operator can create cavities and/or close holes in Γ.

The attachment of a 3-handle is applicable if the two identified faces belong to the same boundary surface component and have all edges in common. This operator fills in the cavity formed by the two identified faces.

4.8 Euler Operators on Cell Complexes: An Overview

In this section, we review the Euler-Poincaré formula for cell complexes, and for complexes in which the cells are not necessarily homeomorphic to a ball, that we have called *general* complexes. The operators on such complexes, which maintain the validity of Euler-Poincaré formula, are called *Euler operators*. We give a taxonomy of Euler operators and we review Euler operators MEV, MEF and MEKR, that are most commonly used in the applications.

4.8.1 Euler-Poincaré Formula for Cell Complexes

The Euler-Poincaré formula expresses the necessary validity condition of a cell complex with manifold or non-manifold carrier [1]. The Euler-Poincaré formula for a cell d-complex Γ (with or without boundary, of homogeneous or non-homogeneous dimension) with n_i i-cells states that

$$\sum_{i=0}^{d}(-1)^i n_i = n_0 - n_1 + \cdots + (-1)^d n_d = \sum_{i=0}^{d}(-1)^i \beta_i$$
$$= \beta_0 - \beta_1 + \cdots + (-1)^d \beta_d.$$

Here, β_i is the i-th Betti number of Γ. It measures the number of independent non-bounding i-dimensional cycles in Γ. The alternating sum $n_0 - n_1 + \cdots + (-1)^d n_d$ of the number of i-cells in Γ is denoted as $\chi(\Gamma)$, and is called the *Euler-Poincaré characteristic* of Γ.

If Γ is a (manifold or non-manifold) cell complex embedded in $\mathbb{E}^3$, then $\beta_3 = 0$ and the number of 0-cells (vertices), 1-cells (edges), 2-cells (faces) and 3-cells (volumes) is usually denoted as v, e, f and c, respectively. For a cell 3-complex in $\mathbb{E}^3$,

$$v - e + f - c = \beta_0 - \beta_1 + \beta_2. \tag{4.1}$$

For a cell 2-complex in $\mathbb{E}^3$, also $c = 0$ and

$$v - e + f = \beta_0 - \beta_1 + \beta_2. \tag{4.2}$$

Intuitively, in $\mathbb{E}^3$, β_0 is the number of connected components, β_1 is the number of holes (non-bounding 1-cycles), and β_2 is the number of cavities (non-bounding 2-cycles) in Γ.

For a manifold cell 2-complex Γ whose carrier is the boundary of a solid object in $\mathbb{E}^3$ (a boundary representation), the Euler-Poincaré formula states that

$$v - e + f = 2(s - g). \tag{4.3}$$

Here, s is the total number of shells (connected components of Γ) and g is the genus of the boundary surface.

If Γ is a cell 2-complex homeomorphic to a 2-sphere, then

$$v - e + f = 1 - 0 + 1 = 2. \tag{4.4}$$

This formula is maintained by the operators in [4].

If Γ is a cell 1-complex (a graph), also called a wire frame, then

$$v - e = \beta_0 - \beta_1. \tag{4.5}$$

This formula is used in the approach in [42].

4.8.2 Euler-Poincaré Formula for General Complexes

The Euler-Poincaré formula has been extended to the case of general complexes, in which cells are not necessarily homeomorphic to a ball, as discussed in Sect. 4.2.

If Γ is a 3-complex, in which faces may be bounded by more than one ring of edges but are mappable to a plane, and volumes may have cavities and holes, then the Euler-Poincaré formula

$$v - e + (f - r) - (c - ch + cc) = \beta_0 - \beta_1 + \beta_2 \tag{4.6}$$

holds. Here, r is the total number of inner rings (loops) in the faces (2-cells) of Γ, ch is the total number of holes in the volumes (3-cells) of Γ, and cc is the total number of cavities in the volumes (3-cells) of Γ. The operators in [28, 29] maintain formula (4.6).

If Γ is a 2-complex embedded in $\mathbb{E}^3$, in which faces may be bounded by multiple rings, then

$$v - e + (f - r) = \beta_0 - \beta_1 + \beta_2. \tag{4.7}$$

The operators in [21] maintain formula (4.7).

If the carrier of Γ is the boundary of a solid object in $\mathbb{E}^3$, then

$$v - e + (f - r) = 2(s - g). \tag{4.8}$$

Here, s is the total number of shells (connected surface components) and g is the genus of the boundary surface. The operators in [2, 5, 12, 26, 27] maintain formula (4.8).

We note here that in the literature, the numbers β_0, β_1 and β_2 are usually denoted as C (the number of connected components), Ch (the number of complex holes) and Cc (the number of complex cavities), respectively. The numbers c, ch and cc are also denoted as Vl (the number of volumes), Vh (the number of volume holes) and Vc (the number of volume cavities), respectively.

4.8.3 Classification of Euler Operators

Euler-Poncaré formula can be regarded as the equation of a hyperplane (a subspace) of dimension $k - 1$ in the discrete space (lattice) $\mathbb{Z}^k$, where k is the number of topological entities involved in the formula [27]. The basis of this subspace, i.e., the

minimal set of independent vectors that span the hyperplane, has $k-1$ independent k-dimensional vectors. These vectors may be interpreted as operators, called *Euler operators*, in which each coordinate corresponds to an entity in the Euler-Poincaré formula. Each coordinate represents the number of the topological entities introduced or removed from the model, depending on the sign of the coordinate. In the literature, there have been many proposals for the basis vectors of this discrete space, which correspond to Euler operators for updating complexes satisfying the corresponding Euler-Poincaré formula.

We classify Euler operators acting on topological shape representations in the form of a cell complex in three classes.

- *Initialization* operators, which create an initial (simple) model with few topological entities. The initial model is created such that the Euler-Poincaré formula involving those entities is satisfied. Initialization operators can also create a new connected component in the model, thus increasing the zeroth Betti number β_0.
- *Topology preserving* operators, which change the combinatorial description of the model, but do not change its topology. They introduce or remove cells in the cell complex Γ which is the topological model representing the object, but they do not change the topology (expressed through the Euler-Poincaré characteristics and the Betti numbers) of the complex.
- *Topology modifying* operators, which introduce or remove topological entities in a way that changes the topology of the complex, but they do not influence the validity of the Euler-Poincaré formula. They change the topological characteristics of the model expressed through the Betti numbers (e.g. connected components, genus and cavities, i.e., the zeroth, first and second Betti number of the complex embedded in $\mathbb{E}^3$). Initialization operators may be considered as topology modifying operators, as they create an initial model starting from the empty set.

4.8.4 MEV, MEF and MEKR Operators

We adopt the naming convention widely used for Euler operators. Letters *M* and *K* stand for *Make* and *Kill* (create and delete) a topological entity. *Kill* operators are inverse to the corresponding *Make* ones. They undo the effect of *Make* operators. Despite the wide variety of the basis Euler operators proposed in the literature, some of the operators are a part of virtually all of the bases. We will review here the Euler operators that belong to the majority of the basis, and are common to most of the proposals for building and updating cell complexes. These operators are called *MEV*, *MEF* and *MEKR*.

MEV (*Make Edge and Vertex*) operator creates a new edge and a new vertex. There are three instances of *MEV* operator, depending on the way the new edge and vertex are introduced in Γ:

- an existing vertex is split in two vertices, connected through the new edge (*MEV*1);

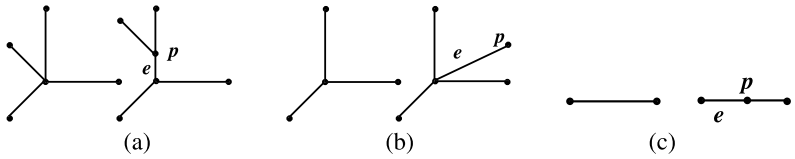


Fig. 4.11 The instances of the *MEV* operator: *MEV*1 (**a**); *MEV*2 (**b**) and *MEV*3 (**c**)

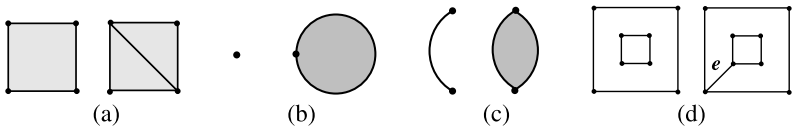


Fig. 4.12 The instances of the *MEF* operator: *MEF*1 (**a**); *MEF*2 (**b**) and *MEF*3 (**c**). The *MEKR* operator (**d**)

- the new vertex is incident in the new edge only (*MEV*2). The new edge may be a wire edge, or it may be inside a face;
- the new vertex is introduced in the interior of an existing edge, splitting it in two (*MEV*3).

*MEV*1 instance of the *MEV* operator is a generalization to cell complexes of the half-vertex split operator, commonly used in mesh processing, and reviewed in Sect. 4.7. The *MEV* operator is illustrated in Fig. 4.11.

MEF (*Make Edge and Face*) operator creates a new edge and a new face. In the literature, three instances of *MEF* operator have been proposed:

- the new edge connects two different vertices on the same face, splitting the existing face in two (*MEF*1);
- the new edge is a loop edge, such that the endpoint of the new edge is an existing vertex (*MEF*2). The new face may be introduced in the interior of an existing face;
- an existing edge is expanded into a new edge and a new face, bounded only by the new edge and the edge that has been split (*MEF*3).

The three instances of the *MEF* operator are illustrated in Fig. 4.12.

In the 2D manifold case, operators *MEF*1, *MEF*2 and *MEF*3 are dual to operators *MEV*1, *MEV*2 and *MEV*3, respectively, in the sense that duality exchanges faces with vertices, and boundary with co-boundary. All the instances of the *MEV* operators are common in practice, while among the *MEF* operators, the *MEF*1 is the most commonly used.

Another topology preserving operator is used in many approaches to updating complexes in which faces may be bounded by several rings (loops). It is called *MEKR* (*Make Edge, Kill Ring*). It makes an edge connecting two vertices on two distinct rings (loops) bounding the same face. The new edge belongs two times to the face. The *MEKR* operator is illustrated in Fig. 4.12(d). This operator introduces an edge in the model, and eliminates an inner ring in the face. The repeated use

of this operator may produce a complex in which all faces are simply connected, showing that Euler-Poincaré formulas (4.8) and (4.2) are consistent.

4.9 Euler Operators on Manifolds

We consider the various basis sets of Euler operators proposed in the literature for modeling manifold cell complexes, and we analyze them in the order of increasing complexity of the complex. The first approach to modeling polyhedral models (surfaces) homeomorphic to a 2-sphere and satisfying formula (4.4) has been proposed in [4]. In this approach, only topology preserving operators are defined: *MEV*3 instance of the *MEV* operator, and *MEF*1 instance of the *MEF* operator. We review here the operators for modeling manifold 2-complexes of arbitrary topology. We review also the splice operator, defined for manifold cell 2- and 3-complexes. It unifies the various Euler operators, both topology preserving and topology modifying.

4.9.1 Euler Operators on Manifold 2-Complexes Bounding a Solid (Boundary Representations)

We review the Euler operators that can manipulate a representation Γ of an orientable manifold surface S bounding a solid object in 3D and satisfying Euler-Poincaré formula (4.8) [5, 12, 26, 27]. We first review briefly the operators common to all the approaches, and then we review in greater detail the operators defined for specific approaches.

Topology preserving operators used in all the approaches are *MEV* and *MEF* operators, described in Sect. 4.8. In [12], all three instances of the *MEV* operator are used, under different names, but only its *MEV*1 instance (the new vertex is inserted in an existing edge, splitting it in two) is in the set of basis operators. In [26], all three instances of the *MEV* operators are used. In [27], the *MEV*1 and *MEV*2 (the new vertex is connected to the new edge only) instances of the *MEV* operator are used. In [5], *MEV*1 and *MEV*2 instances of the *MEV* operator are used.

The instance of *MEF* operator used in [5, 12, 26] is *MEF*1. It makes an edge and a face, by connecting two existing vertices on the same boundary component of an existing face, thus splitting the existing face in two. In [27], both *MEF*1 and *MEF*2 (the new face is bounded only by the new edge) instances of the *MEF* operator are used.

MEKR operator, described in Sect. 4.8, is used in [5, 12, 27]. It may transform each face of Γ into a topological 2-cell.

In [12], two topology modifying operators are defined:

- *MEKFS* (*Make Edge, Kill Face and Shell*) operator joins two vertices belonging to two faces on two different shells through a new edge, merging the two faces into

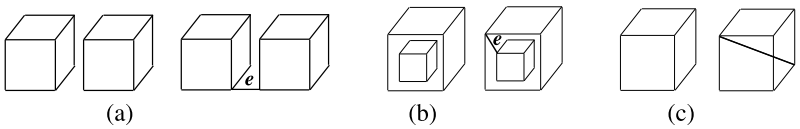


Fig. 4.13 The two instances of the *MEKFS* (*Make Edge, Kill Face and Shell*) operator: the new edge joins two connected components (**a**) or kills a cavity (**b**). The *MEH* (*Make Edge and Hole*) operator: the two joined vertices are on the same shell

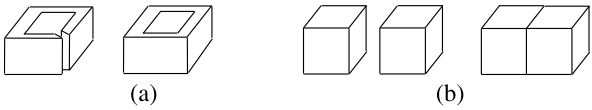


Fig. 4.14 The two instances of the glue operator: *KFMH* (*Kill Face, Make Hole*) (**a**); *KFS* (*Kill Face and Shell*) (**b**)

one face, and merging the two shells into one shell. It reduces by one the number β_0 of connected components. *MEKFS* operator is illustrated in Figs. 4.13(a) and (b).

- *Glue* operator merges two faces and deletes them both. It corresponds to the connected sum operator on manifold surfaces. Two faces may be glued by the glue operator if they have no inner rings (they are simply-connected), their outer rings have the same number of vertices, and the two faces have no edges in common. The glue operator deletes not only the two faces, but it deletes also all the edges and vertices on the boundary of one of the deleted faces. There are two instances of the glue operator, illustrated in Fig. 4.14.
  - If the two glued faces belong to the same shell, a handle (genus) is created, and the operator is called *KFMH* (*Kill Face, Make Hole* (*Handle*)).
  - If the two glued faces belong to two different shells, one shell is deleted, and the operator is called *KFS* (*Kill Face and Shell*).

The operator *MEKR* (*Make Edge, Kill Ring*), which connects two different rings bounding the same face, thus producing faces that are topological cells, is also used. *MEF* (*Make Edge and Face*), *MEKR* (*Make Edge, Kill Ring*) and *MEKFS* (*Make Edge, Kill Face and Shell*) are grouped together into one operator, called *ME* (*Make Edge*), since they all make an edge in the model.

In [5], the topology modifying operator is called *MRKF* (*Make Ring, Kill Face*). It creates a ring and deletes a face from the model, by gluing a (simply connected) face to another face, thus deleting one face and making an (inner) ring in another face. It has two instances:

- *KFMRH* (*Kill Face, Make Ring and Hole* (*Handle*)) operator glues two faces belonging to the same shell, thus making a handle in the surface.
- *MRKFS* (*Make Ring, Kill Face and Shell*) operator glues together two faces belonging to two different shells, thus merging two shells into one.

MRKF operator is similar to the glue operator in [12], but it imposes looser conditions on the glued faces, and it deletes only one of the faces. In [5], the *MEKFS*

(*Make Edge, Kill Face and Shell*) operator is also used. It is the same as the homonymous operator in [12]. Together with *MEF* (*Make Edge and Face*) and *MEKR* (*Make Edge, Kill Ring*), it is grouped into a *ME* (*Make Edge*) operator, similar to [12].

The topology modifying operator in [26] is *MEKFS* (*Make Edge, Kill Face and Shell*). It joins two faces by joining a vertex of each face through an edge, where the new edge belongs twice to the new joined face. It has three instances. The first two are the same as the two instances of the homonymous operator in [5, 12], illustrated in Figs. 4.13(a) and (b). The third one occurs if the two joined faces belong to the same shell. Then, a handle is created. The third instance of the *MEKFS* operator is called *MEHKF* (*Make Edge and Hole* (*Handle*), *Kill Face*), and is illustrated in Fig. 4.13(c).

Topology modifying operator *MRKF* (*Make Ring, Kill Face*) in [27] is the same as *MRKF* operator in [5]. It glues two faces together, by making the boundary of one face an inner loop of another face.

Euler operators have also been defined on face oriented data structures in [2] and [43].

4.9.2 Splice Operator

The *splice* operator has been designed in 2D specifically for the Quad-Edge data structure [16], and in 3D for the Facet-Edge data structure [11]. This operator has a straightforward implementation in these data structures, and it unifies the various Euler operators in a single operator.

The *splice* operator in 2D, proposed in [16], takes as argument two edges. Depending on the cycles the two edges belong to, the splice operator can be either topology preserving or topology modifying. In the case when it is topology preserving, splice can be expressed through topology preserving Euler operators as a *MEF* followed by *KEV*, or as a *MEV* followed by *KEF*. Thus, it either increases the number of faces by one and decreases the number of vertices by one, or it decreases the number of faces by one and increases the number of vertices by one.

The *splice* operator in [11] extends the splice operator in [16] to the Facet-Edge data structure. There are two instances of the splice operator: *splice facet* and *splice edge*. The two operators are defined on the edge rings and the face rings of the involved facet-edges, but they are not guaranteed to produce a valid complex.

Because of these drawbacks, another set of operators is introduced in [11]. The *splice* operator in [16] is adapted to act on edges in the subdivision of a surface bounding a 3-cell, and is defined and implemented through the Facet-Edge operator splice edge. The *meld* operator is topology modifying. It glues two faces with equal number of edges on their boundaries. If the two faces belong to different connected components, the two components are merged. If the two faces belong to the same component, a genus is created. It corresponds to the glue operator in [12], but it deletes only one of the glued faces, not both of them.

4.10 Euler Operators on Non-manifolds

Only a few approaches to the update operators on non-manifolds have been proposed in the literature. The earliest approaches to solid modeling considered a *wireframe model*, which is a non-manifold cell 1-complex (a graph). In [42], a set of operators on wireframes is defined: topology preserving operator is the *MEV*2 instance of the *MEV* (*Make Edge and Vertex*) operator; topology modifying operator is called *ME* (*Make Edge*), which either merges two components or creates a hole. We review here Euler operators on non-manifold complexes in the order of increasing dimension of the complexes on which they act: operators on 2-complexes, and operators on 3-complexes.

4.10.1 Euler Operators for 2-Complexes

The first approach to modeling and updating the boundary of a non-manifold solid object is introduced in [39]. The set of basis operators is not proposed, but a verbose list of operators for updating such models is presented, and the change in the number of topological entities (vertices, edges, rings, faces, shells, regions or volumes, and models) is discussed for each operator.

In [21], the set of basis operators for a non-manifold 2-complex Γ satisfying the Euler-Poincaré formula (4.7) is proposed.

Topology preserving operators are:

- *MEV*2 instance of the *MEV* (*Make Edge and Vertex*) operator, which makes a new edge and a new vertex, incident in the new edge only (the new edge may belong to a face or it may be a wire edge inside a shell);
- *MVR* (*Make Vertex and Ring*), which makes a vertex inside a face, creating a new inner ring, consisting of a single vertex.

Topology modifying operators are

- *MVC* (*Make Vertex and Connected Component*), which makes a new connected component, composed of the new vertex only;
- *MECh* (*Make Edge and Complex Hole*), which makes a new edge connecting two existing vertices on the same connected component in the complex, and creates a hole;
- *MFKCh* (*Make Face, Kill Complex Hole*), which makes a face, which fills in and deletes an existing complex hole (cycle of edges);
- *MFCc* (*Make Face and Complex Cavity*), which makes a face filling in a loop of edges that is not a cycle, and closes off a cavity.

Topology modifying operators in [21] are illustrated in Fig. 4.15.

In addition to this set of basis Euler operators, additional macro-operators are introduced to ease the use of Partial-Entity data structure.

- *SE* (*Split Edge*) is the *MEV*3 instance of the *MEV* (*Make Edge and Vertex*) operator. It inserts a new vertex in the interior of an existing edge, splitting it in two.

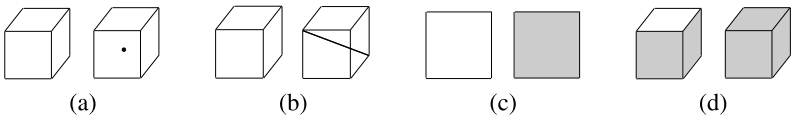


Fig. 4.15 Topology modifying operators in [21]: *MVCc* (*Make Vertex and Cavity*) (**a**); *MECh* (*Make Edge and Hole*) (**b**); *MFKCh* (*Make Face, Kill Hole*) (**c**); *MFCc* (*Make Face and Cavity*) (**d**)

- *MEF* (*Make Edge and Face*) operator is its *MEF*1 instance. It inserts an edge in the interior of an existing face by connecting two vertices on the same boundary component of the face through the new edge, splitting the face in two.
- *MEKR* operator is described in Sect. 4.8. It is used to eliminate inner rings bounding a face by creating a new edge, which connects two vertices on the same face, but on two different boundary components of the face.
- *MEKC* makes a new edge, which connects two vertices on different shells, and merges the two shells.

In [37], another basis set of Euler operators is proposed, operating on 2-complexes satisfying the Euler-Poincaré equation $v - (e - eh) + (f - fh + fc) = \beta_0 - \beta_1 + \beta_2$, where eh, fh and fc are the number of edge holes (edges not incident in any vertex, and not bounding any face), face holes (holes inside faces) and face cavities (faces without boundary, i.e., not incident in any edge or vertex), respectively.

Topology preserving operators are

- *MEV*2 and *MEV*3 instances of the *MEV* (*Make Edge and Vertex*),
- all three instances of the *MEF* (*Make Edge and Face*),
- *MVKEh* (*Make Vertex, Kill Edge without boundary*), which makes a vertex on an edge without boundary, converting that edge to an edge with boundary,
- *MVFh* (*Make Vertex and Ring*), which makes an isolated vertex in a face, making thus a ring (a 1-cycle) in the face,
- *MVKFc* (*Make Vertex and Shell, Kill Face Cavity*), which makes a vertex in a face homeomorphic to a sphere, converting the face to a face with boundary,
- *MEEhFFh* (*Make Edge without boundary and Face bounded by that edge*), which makes an edge without boundary, and a face bounded by that the edge, either around some connected set of vertices (and edges), or inside a face, which becomes multiply connected.

Topology modifying operators are called *MECh* (*Make Edge and Complex Hole*) and *MFCc* (*Make Face and Complex Cavity*). *MECh* makes a loop edge, bounded by a single vertex, and it thus creates a 1-cycle in the complex Γ. *MFCc* makes a face bounded by a single vertex, thus creating a 2-cycle in Γ.

4.10.2 Euler Operators on 3-Complexes

In [29], the set of operators for updating a 3-complex Γ satisfying the Euler-Poincaré formula (4.6) is proposed.

The topology preserving operator is the *MEV*2 instance of the *MEV* (make edge and vertex) operator. The new edge connects the new vertex to an existing one.

Operators that modify the topology of initial cells, but do not modify the topology of the underlying shape (and thus the Betti numbers of the complex decomposing the shape), are

- *MVR* (*Make Vertex and Ring*), which makes a new isolated vertex inside a face, and a new ring composed of a single vertex. It is the same as the homonymous operator in [21],
- *MVVc* (*Make Vertex and Volume Cavity*), which makes a new isolated vertex inside a volume,
- *MEVh* (*Make Edge and Volume Hole*), which makes a new isolated edge (not incident in any face) inside a volume.

These operators produce cells that are not topological cells.

Operators that change the global topological characteristics of the complex are

- *MECh* (*Make Edge and Complex Hole*), which makes an edge connecting two existing vertices on the same connected component of edges, and makes a hole,
- *MFKCh* (*Make Face, Kill Complex Hole*), which makes a face that fills in a cycle of edges,
- *MFCc* (*Make Face and Complex Cavity*), which makes a face that closes off a cavity, and
- *MVlKCc* (*Make Volume, Kill Complex Cavity*), which makes a volume that fills in a cavity.

The *MECh*, *MFKCh* and *MFCc* operators are the same as the homonymous operators in [21].

In [28], the operators in [29] are described in greater detail. In particular, three types of the *MEV*2 instance of the *MEV* (*Make Edge and Vertex*) operator are considered, depending on the position of the existing vertex w that is connected to the new edge: w does not belong to a face or a volume; w is in the interior of a face; w is in the interior of a volume.

In addition to the basis Euler operators, other topological macro-operators are introduced. The macro-operators can be expressed through the basis operators, and they allow for the more flexible implementation of high-level operators. There are two groups of the macro-operators.

- *ME* (*Make Edge*) operators. They make an edge connecting two existing vertices.
  - If the two vertices are on the two different rings on the same face, the operator merges the two rings (i.e., deletes one of the rings), thus decreasing the number r by one. This is the *MEKR* (*Make Edge, Kill Ring*) operator described in Sect. 4.8.

  - If the two vertices belong to two different cavities inside the same connected component, the two cavities are merged, and the number Cc of volume cavities is decreased by one.
  - If the two vertices belong to two different complexes (connected components) in the model, the two complexes are merged, and the number C of connected components is decreased by one.
- *Split* operators.
  - Split edge operator is the *MEV*3 instance of the *MEV* (*Make Edge and Vertex*) operator. It inserts a vertex in an existing edge, splitting it in two edges.
  - Split face operator inserts an edge in an existing face, splitting it in two faces. It is the same as the *MEF*1 instance of the *MEF* (*Make Edge and Face*) operator.
  - Split volume operator inserts a face in an existing volume, splitting the volume in two.

In [15], the operators in [29] have been extended to complexes called *stratifications*, in which cells, called *strata*, are defined by analytic equalities and inequalities. The cells are not necessarily homeomorphic to a ball, and they may have incomplete boundaries. We will review briefly only the operators acting on cells with complete boundaries. Topology preserving operators are called *cell subdividers* and *local hole shapers*. A cell subdivider subdivides an n-cell by inserting into it an $(n-1)$-cell. A local hole shaper merges a topological cell into an incident cell of higher dimension, changing the number of holes in the higher dimensional cell. Topology modifying operators are called *global hole shapers*. A global hole shaper either attaches or detaches a cell, thus creating or deleting a hole.

4.11 Discussion

We have presented a taxonomy for data structures that model cell complexes, and classified such data structures according to the dimension of the complex they represent into dimension independent data structures, data structures for 2-complexes, and data structures for 3-complexes. We have further classified the data structures in each group according to the basic kind of the topological entities they represent. Table 4.1 summarizes the characteristics of the data structures we reviewed according to the proposed taxonomy. We have described each data structure in terms of the entities and topological relations it encodes, and we have discussed it based on its expressive power and on the efficiency in supporting navigation inside the complex. Explicit data structures encode directly all cells in the complex, while implicit data structures encode some relations among the cells in the complex. All data structures support the retrieval of all topological relations in time linear in the number of topological entities involved in the relation. The only exception is the Star-Vertex data structure. It retrieves efficiently only $R_{2,0}$ and $R_{0,0}$ relations, while the other topological relations are retrieved in time linear in the number of vertices in the complex. We also distinguish between the data structures that represent manifold cell complexes from the ones that model non-manifold complexes. A special class of cell

Table 4.1 Characteristics of data structures

Data structure	Domain[a]	Method
	Dimension-independent data structures	
Cell-Tuple	M	Implicit
IG	NM	Incidence-based
	Data structures for 2D models	
Winged-Edge	M	Edge-based
DCEL	M	Edge-based
Half-Edge	M	Edge-based
Quad-Edge	M	Implicit
Lath	M	Implicit
Star-Vertex	M	Adjacency-based
RE	NM	Edge-based
PE	NM	Edge-based
HC	NM	Edge-based
	Data structures for 3D models	
Facet-Edge	M	Implicit
HF	M	Edge-based

[a]M = Manifold, NM = Non-manifold

complexes are simplicial complexes. Data structures for cell complexes can also be used to represent simplicial complexes, but, because of the widespread use of such complexes in the applications, specific data structures have been developed [10].

There is a vast literature on building and update operators on cell complexes. Here, we have focused on Euler operators. These latter have been defined in the literature for manipulating both manifold and non-manifold complexes. In our analysis, we have further classified them according to the dimension of the complex. The summary of the most commonly used Euler operators is given in Table 4.2. As we pointed out, the notion of a cell complex is not general enough to accommodate all the requirements of modeling applications. Thus, in the literature, instead of cell complexes, a more general type of complexes is considered. Usually, faces are not supposed to be homeomorphic to a 2-ball, but are allowed to be multiply connected: bounded by several rings of edges, but mappable to a plane. In some approaches, also volumes are allowed to have through holes and cavities. We summarize the operators that affect the complexes in which only faces may be multiply connected, while volumes are topological 3-cells (homeomorphic to a 3-ball). In the last column of the table, the change in the number of topological entities is indicated for each operator. It can easily be seen that all the operators maintain the validity of the corresponding Euler-Poncaré formula. The entities appear in the tuples in the same order as in the corresponding Euler-Poincaré formula. For general topology preserving Euler operators *MEV*, *MEF*, and *MEKR*, which form part of almost all

Table 4.2 Characteristics of Euler operators

Euler operator	Domain[a]	Type[b]	Effect
	General Euler operators		
MEV	M & NM	TP	$e = e + 1, v = v + 1$
MEF	M & NM	TP	$e = e + 1, f = f + 1$
MEKR	M & NM	TP	$e = e + 1, r = r - 1$
	Manifold 2-complexes without boundary, $v - e + (f - r) = 2(s - g)$		
MEKF—MEKFS	M	TM	$(0, 1, -1, 0, -1, 0)$
MEKF—MEHKF	M	TM	$(0, 1, -1, 0, 0, 1)$
Glue—KFMH	M	TM	$(0, 0, -2, 0, 0, 1)$
Glue—KFS	M	TM	$(0, 0, -2, 0, -1, 0)$
KFMR—KFMRH	M	TM	$(0, 0, -1, 1, 0, 1)$
KFMR—KFSMR	M	TM	$(0, 0, -1, 1, -1, 0)$
	Non-manifold 2-complexes with boundary, $v - e + (f - r) = C - Ch + Cc$		
MVC	NM	TM	$(1, 0, 0, 0, 1, 0, 0)$
MECh	NM	TM	$(0, 1, 0, 0, 0, 1, 0)$
MFKCh	NM	TM	$(0, 0, 1, 0, 0, -1, 0)$
MFCc	NM	TM	$(0, 0, 1, 0, 0, 0, 1)$
MVR	NM	TM	$(1, 0, 0, 1, 0, 0, 0)$
	Non-manifold 3-complexes with boundary, $v - e + (f - r) - V = C - Ch + Cc$		
MECh	NM	TM	$(0, 1, 0, 0, 0, 0, 1, 0)$
MFKCh	NM	TM	$(0, 0, 1, 0, 0, 0, -1, 0)$
MFCc	NM	TM	$(0, 0, 1, 0, 0, 0, 0, 1)$
MVlKCc	NM	TM	$(0, 0, 0, 0, 1, 0, 0, -1)$

[a] M = Manifold, NM = Non-manifold

[b] TP = Topology preserving, TM = Topology modifying

the basis operators proposed in the literature, for manifold and non-manifold domains, modeled by cell or general complexes, in two or three dimensions, we do not use the tuples to describe the change in the number of cells in the complex, but we describe this change in a pseudo-code like style.

An interesting class of operators is provided by the higher-level operators based on the handlebody theory proposed in [24, 25, 32]. Their definition is based on the fact that any n-manifold can be generated from an n-ball by attaching handles. Handle operators are defined for cell 2- and 3-complexes with boundary in [24, 32] and [25], respectively. They can be classified as topology modifying operators. Handle operators in 2D can be expressed through Euler operators as discussed below.

- The attachment of a 0-handle corresponds to creating an initial triangle (a 2-ball). It can be expressed through the initialization Euler operator *MFC* (*Make Face*

and Component), followed by *MVR* (*Make Vertex and Ring*), two *MEV* operators and one *MEF* operator, which together create a triangle.

- The attachment of a 1-handle identifies two boundary edges with no vertices in common. It can be expressed through
  - two *ME* (*Make Edge*) operators that connect the corresponding endpoints of the two edges to be identified (one *MEKFS* (*Make Edge, Kill Face and Shell*) and one *MECh* (*Make Edge and Complex Hole*) if the two identified edges are on different connected components; two *MECh* (*Make Edge and Complex Hole*) operators if the two identified edges are on the same component),
  - two *KEV* (*Kill Edge and Vertex*) operators that contract the two edges created by *ME* operators, and identify the corresponding endpoints,
  - *MFKCh* (*Make Face, Kill Complex Hole*) which creates a face that fills the ring formed by two edges to be identified, and
  - *KEF*3 (inverse of the *MEF*3 instance of *MEF* operator) which contracts the created face and merges the two edges.
- The attachment of a 2-handle identifies two edges with both vertices in common. It can be expressed as a *MFKCh* operator, followed by *KEF*3 operator.

Handle operators in 3D identify two boundary faces with the same number of edges and vertices, and convert them into an inner face. The corresponding edges and vertices on the boundary of the two identified faces are also identified. Thus, the handle operators in 3D generalize the glue operator in [12], since the two glued faces may have some, or all, edges in common. An interesting research direction would be to generalize handle operators to higher dimensions, i.e., for an n-manifold discretized as a cell complex, and to express them as macro-operators through Euler operators in higher dimensions.

As we have seen, there are many non-unified proposals in the literature for the manipulation operators on cell complexes, and on general complexes. So, we propose a minimal set of Euler operators which subsume all the other Euler operators. Let us consider the case of cell 2-complexes embedded in the 3D Euclidean space $\mathbb{E}^3$. We can define the following operators:

- Topology preserving operators: *MEV* (*Make Edge and Vertex*) and *MEF* (*Make Edge and Face*).
- Topology modifying operators:
  - *MV*0*Cycle* (*Make Vertex and* 0*-Cycle*)
  - *ME*1*Cycle* (*Make Edge and* 1*-Cycle*) and
  - *MF*2*Cycle* (*Make Face and* 2*-Cycle*).

Operator *MV*0*Cycle* creates a new vertex and a new connected component, it increases by one the number of vertices (0-cells) and the zeroth Betti number β_0. It is also an initialization operator. Operator *ME*1*Cycle* creates a new edge and forms a 1-cycle, thus increasing by one the number of edges (1-cells) and the first Betti number β_1. Operator *ME*2*Cycle* creates a new face and forms a 2-cycle, thus increasing by one the number of faces (2-cells) and the second Betti number β_2. Figure 4.16 shows an example of *ME*1*Cycle* and *ME*2*Cycle* operators. For 3-complexes embedded in the 3D Euclidean space, there will be an additional topology preserving

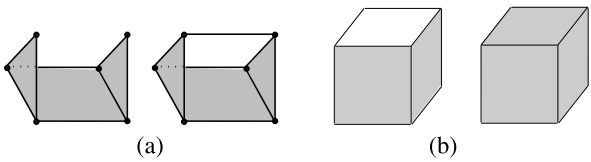


Fig. 4.16 Topology modifying operators: *ME*1*Cycle* (*Make Edge and* 1*-Cycle*) (**a**); *MF*2*Cycle* (*Make Face and* 2*-Cycle*) (**b**)

operator *MFVl* (*Make Face and Volume* (*3-Cell*)) which adds a new face (2-cell) and a new three-dimensional (volumetric) cell. The topology modifying operators will be the same as for 2-complexes, since in this case the Betti number β_3 is null.

The above operators can be naturally generalized to arbitrary dimensions. If we consider a complex of dimension d, we will have:

- d topology preserving operators: $MiC(i+1)C$ (*Make* i*-Cell and* $(i+1)$*-Cell*), which create an i-cell and an $(i+1)$-cell,
- $d+1$ topology modifying operators: $MiCiCycle$ (*Make* i*-Cell and* i*-Cycle*), which create an i-cell and an i-cycle.

It is easy to see that other operators we reviewed here, defined on 2-complexes or 3-complexes, are instances of one of the above operators when we restrict the attention to cell complexes. Also, it should be easy to prove that the above operators form a complete basis of Euler operators. An interesting open question is the generalization of the above operators to general complexes, which are a superset of cell complexes.

Moreover, an important issue would be to define a complete set of basic operators, like Euler operators, for simplicial complexes, first for two-dimensional and three-dimensional ones, and then in arbitrary dimensions. Also, it would be interesting to express common operators on simplicial complexes, like edge collapse, or vertex-pair collapse, in terms of such operators.

Finally, we want to mention an approach to modeling 3D shapes that has been proposed in [14, 35]. This approach is based on the notion of stratified sets, which generalize cell complexes in the sense that a stratum need not to be connected, nor bounded, nor globally homeomorphic to an open ball. In [15], the author proposes a set of Euler operators for 3D objects with incomplete boundaries. It will be interesting to see how these operators, when restricted to objects with complete boundary, relate to the other existing Euler operators for non-manifold objects both for general and classical cell complexes, that we have reviewed here.

Finally, the definition of Euler operators on stratifications is an interesting research direction, since stratifications have a solid mathematical basis.

Acknowledgements This work has been partially supported by the Italian Ministry of Education and Research under the PRIN 2009 program, and by the National Science Foundation under grant number IIS-1116747. The authors want to thank Annie Hui for the material on data structures for three-dimensional cell complexes and for many helpful discussions.

References

1. Agoston, M.K.: Computer Graphics and Geometric Modeling: Mathematics. Springer, London (2005). ISBN:1-85233-817-2
2. Ansaldi, S., De Floriani, L., Falcidieno, B.: Geometric modeling of solid objects by using a face adjacency graph representation. In: Cole, P., Heilman, R., Barsky, B.A. (eds.) SIGGRAPH, pp. 131–139 (1985)
3. Baumgart, B.G.: Winged edge polyhedron representation. Technical Report CS-TR-72-320, Stanford University, Department of Computer Science (October 1972)
4. Baumgart, B.G.: A polyhedron representation for computer vision. In: Proceedings AFIPS National Computer Conference, vol. 44, pp. 589–596 (1975)
5. Braid, I.C., Hillyard, R.C., Stroud, I.A.: Stepwise construction of polyhedra in geometric modelling. In: Brodlie, K.W. (ed.) Mathematical Methods in Computer Graphics and Design, pp. 123–141. Academic Press, San Diego (1980)
6. Brisson, E.: Representing geometric structures in D dimensions: topology and order. In: Proceedings 5th ACM Symposium on Computational Geometry, pp. 218–227. ACM, New York (1989)
7. De Floriani, L.: Topological relations in cell and simplicial complexes (Course Lecture Notes). Technical report, University of Maryland (2003)
8. De Floriani, L., Greenfieldboyce, D., Hui, A.: A data structure for non-manifold simplicial d-complexes. In: Boissonnat, J.-D., Alliez, P. (eds.) Symposium on Geometry Processing, Nice, France. ACM International Conference Proceeding Series, vol. 71, pp. 85–94. ACM, New York (2004)
9. De Floriani, L., Hui, A., Panozzo, D., Canino, D.: A dimension-independent data structure for simplicial complexes. In: Proceedings of the IMR, pp. 403–420. Springer, Berlin (2010)
10. De Floriani, L., Hui, A.: Data structures for simplicial complexes: an analysis and a comparison. In: Desbrun, M., Pottmann, H. (eds.) Symposium on Geometry Processing, Vienna, Austria. ACM International Conference Proceeding Series, vol. 255, pp. 119–128. Eurographics Association, Aire-la-Ville (2005)
11. Dobkin, D., Laszlo, M.: Primitives for the manipulation of three-dimensional subdivisions. Algorithmica **5**(4), 3–32 (1989)
12. Eastman, C.M., Weiler, K.: Geometric modeling using the Euler operators. In: 1st Annual Conference on Computer Graphics in CAD/CAM Systems, MIT, May 1979
13. Edelsbrunner, H.: Algorithms in Combinatorial Geometry. Springer, Berlin (1987)
14. Gomes, A., Middleditch, A., Reade, C.: A mathematical model for boundary representation of n-dimensional geometric objects. In: Proceedings of 5th ACM Symposium on Solid Modeling, pp. 270–277. ACM, Ann Arbor (1999)
15. Gomes, A.J.P.: Euler operators for stratified objects with incomplete boundaries. In: Proceedings of the Ninth ACM Symposium on Solid Modeling and Applications, SM'04, Genova, Italy, pp. 315–320. IEEE Comput. Soc., Los Alamitos (2004)
16. Guibas, L., Stolfi, J.: Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams. ACM Trans. Graph. **4**(2), 74–123 (1985)
17. Gursoz, E.L., Choi, Y., Prinz, F.B.: Vertex-based representation of non-manifold boundaries. In: Wozny, M.J., Turner, J.U., Preiss, K. (eds.) Geometric Modeling for Product Engineering, pp. 107–130. Elsevier, Amsterdam (1990)
18. Joy, K.I., Legakis, J., MacCracken, R.: Data structures for multiresolution representation of unstructured meshes. In: Farin, G., Hamann, B., Hagen, H. (eds.) Hierarchical and Geometrical Methods in Scientific Visualization. Springer, Heidelberg (2003)
19. Kallmann, M., Thalmann, D.: Star-vertices: a compact representation for planar meshes with adjacency information. J. Graph. GPU Game Tools **6**(1), 7–18 (2001)
20. Lage, M., Lewiner, T., Lopes, H., Velho, L.: CHF: a scalable topological data structure for tetrahedral meshes. In: SIBGRAPI, pp. 349–356. IEEE Comput. Soc., Los Alamitos (2005)

21. Lee, S.H., Lee, K.: Partial entity structure: a fast and compact non-manifold boundary representation based on partial topological entities. In: Proceedings Sixth ACM Symposium on Solid Modeling and Applications, pp. 159–170. ACM, Ann Arbor (2001)
22. Lickorish, W.B.R.: Simplicial moves on complexes and manifolds. In: Proceedings of the Kirbyfest, vol. 2, pp. 299–320 (1999)
23. Lienhardt, P.: N-dimensional generalized combinatorial maps and cellular quasi-manifolds. Int. J. Comput. Geom. Appl. **4**(3), 275–324 (1994)
24. Lopes, H., Pesco, S., Tavares, G., Maia, M., Xavier, A.: Handlebody representation for surfaces and its applications to terrain modeling. Int. J. Shape Model. **9**(1), 61–77 (2003)
25. Lopes, H., Tavares, G.: Structural operators for modeling 3-manifolds. In: Proceedings Fourth ACM Symposium on Solid Modeling and Applications, pp. 10–18. ACM, New York (1997)
26. Mantyla, M.: A note on the modeling space of Euler operators. Comput. Vis. Graph. Image Process. **26**(1), 45–60 (1984)
27. Mantyla, M.: An Introduction to Solid Modeling. Comput. Sci. Press, New York (1988).
28. Masuda, H.: Topological operators and boolean operations for complex-based non-manifold geometric models. Comput. Aided Des. **25**(2), 119–129 (1993)
29. Masuda, H., Shimada, K., Numao, M., Kawabe, S.: A mathematical theory and applications of non-manifold geometric modeling. In: IFIP WG 5.2/GI International Symposium on Advanced Geometric Modeling for Engineering Applications, pp. 89–103. North-Holland, Berlin (1989)
30. McMains, S., Hellerstein, C.S.J.: Out-of-core building of a topological data structure from a polygon soup. In: Proceedings Sixth ACM Symposium on Solid Modeling and Applications, pp. 171–182 (2001)
31. Muller, D.E., Preparata, F.P.: Finding the intersection of two convex polyhedra. Theor. Comput. Sci. **7**, 217–236 (1978)
32. Pesco, S., Tavares, G., Lopes, H.: A stratification approach for modeling two-dimensional cell complexes. Comput. Graph. **28**(2), 235–247 (2004)
33. Rossignac, J., O'Connor, M.: A dimension-independent model for pointsets with internal structures and incomplete boundaries. In: Wosny, M., Turner, J., Preiss, K. (eds.) Geometric Modeling for Product Engineering, Proceedings of the IFIP Workshop on CAD/CAM, pp. 145–180. North-Holland, Amsterdam (1989)
34. Samet, H.: Foundations of Multidimensional and Metric Data Structures: Computer Graphics, Image Processing, and GIS. Morgan Kaufmann, San Mateo (2006)
35. Shapiro, V., Vossler, D.L.: Construction and optimization of CSG representations. Comput. Aided Des. **23**(1), 4–20 (1991)
36. Silva, F.G., Gomes, A.J.: AIF—a data structure for polygonal meshes. In: Computational Science and Its Applications (ICCSA). Lecture Notes in Computer Science, vol. 2669. Springer, Berlin (2003)
37. Silva, F.G.M., Gomes, A.J.P.: Oversimplified Euler operators for a non-oriented, non-manifold B-Rep data structure. In: Bebis, G., Boyle, R.D., Koracin, D., Parvin, B. (eds.) ISVC. Lecture Notes in Computer Science, vol. 3804, pp. 25–34. Springer, Berlin (2005)
38. Velho, L.: Stellar subdivision grammars. In: Kobbelt, L., Schröder, P., Hoppe, H. (eds.) Symposium on Geometry Processing, Aachen, Germany. ACM International Conference Proceeding Series, vol. 43, pp. 188–199. Eurographics Association, Aire-la-Ville (2003)
39. Weiler, K.: Boundary graph operators for non-manifold geometric modeling topology representations. In: Encarnacao, J.L., Wozny, M.J., McLaughlin, H.W. (eds.) Geometric Modeling for CAD Applications, pp. 37–66. Elsevier, Amsterdam (1988)
40. Weiler, K.: The radial edge data structure: a topological representation for non-manifold geometric boundary modeling. In: Encarnacao, J.L., Wozny, M.J., McLaughlin, H.W. (eds.) Geometric Modeling for CAD Applications, pp. 3–36. Elsevier, Amsterdam (1988)
41. Whitney, H.: Local properties of analytic varieties. In: Cairns, S.S. (ed.) Differential and Combinatorial Topology, A Symposium in Honor of Marston Morse, pp. 205–244. Princeton University Press, Princeton (1965)

42. Wilson, P.R.: Euler formulas and geometric modeling. IEEE Comput. Graph. Appl. **5**, 24–36 (1985)
43. Wu, S.-T.: A new combinatorial model for boundary representations. Comput. Graph. **13**(4), 477–486 (1989)
44. Yamaguchi, Y., Kimura, F.: Non-manifold topology based on coupling entities. IEEE Comput. Graph. Appl. **15**(1), 42–50 (1995)

Chapter 5
Binarization of Gray-Level Images Based on Skeleton Region Growing

Xiang Bai, Quannan Li, Tianyang Ma, Wenyu Liu, and Longin Jan Latecki

Abstract In this chapter, we introduce a new binarization method of gray level images. We first extract skeleton curves from Canny edge image. Second, a Skeleton Strength Map (SSM) is calculated from Euclidean distance transform. Starting from the boundary edges, the distance transform is firstly computed and its gradient vector field is calculated. After that, the isotropic diffusion is performed on the gradient vector field and the SSM is computed from the diffused vector field. It has two advantages that make it useful for skeletonization: 1) the SSM serves as the form of the likelihood of a pixel being a skeleton point: the value at pixels of the skeleton is large while at pixels that are away from the center of object, the SSM value decays very fast; 2) By computing the SSM from the distance transform, a parameterized noise smoothing is obtained. Then, skeleton curves are classified into foreground and background classes by comparing the mean value of their local edge pixels and neighbors lowest intensity. Finally, the binarization result is obtained by choosing foreground skeleton curve pixels as seed points and presenting region growing algorithm on gray scale image with certain growing criteria. Images with different types of document components and degradations are used to test the effectiveness of the

X. Bai (✉) · W. Liu
Huazhong University of Science and Technology, Wuhan, China
e-mail: xiang.bai@gmail.com

W. Liu
e-mail: liuwy@hust.edu.cn

Q. Li
University of California, Los Angeles, CA, USA
e-mail: quannan.li@gmail.com

T. Ma · L.J. Latecki
Temple University, Philadelphia, PA, USA

T. Ma
e-mail: ma.tianyang@gmail.com

L.J. Latecki
e-mail: latecki@temple.edu

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_5,

proposed algorithm. Results demonstrate that the method performs well on images with low contrast, noise and non-uniform illumination.

5.1 Introduction

Image binarization is an image pixel labeling problem, which aims to classify the pixels of an image into two classes: foreground and background [39]. It is an important pre-processing step for most document analysis algorithms, and is widely used in medical image analysis and automatic recognition of targets such as character, fingerprint, etc.

Currently, the existing binarization methods are mainly categorized into two classes: global and local [41, 48]. The global thresholding method [24, 25, 36] converts a gray-level image into a binary image based on an image intensity value called global threshold, which is invariant in the whole image domain. For images with an obvious bimodal histogram, global binarization methods are effective. However, in reality bimodality in histograms is not guaranteed, and single global threshold is not robust and sufficient for images with low contrast or non-uniform illumination. In local thresholding techniques [6, 13, 33, 40], thresholding value can vary from one pixel location to next, and is selected according to the local area information adaptively. Therefore, local binarization methods, which are also called adaptive binarization methods, have better performance on the images with more complex situations. But local thresholding method also has limitations: when neighborhood is small, binarization results will suffer from fake shadows and the loss of objects. Some hybrid methods therefore are proposed. In those methods, both global and local information are used to label the pixel. In Trier and Jain's goal-directed evaluation for 19 methods, Niblack's method performs best among 11 different locally adaptive binarization methods. Sezgin and Sankur [41] ranked 40 binarization methods based on their performance on non-destructive testing (NDT) images and document images, Kittler's method is the best performing thresholding method in both cases.

Edge information has been utilized by several thresholding methods, and is proved to be useful. Yanowitz and Bruckstein [51] proposed an adaptive thresholding method, in which threshold surface is determined by interpolating the image gray levels at points where the gradient is high, indicating probable object edges. However, because this method depends only on the information of points with high gradient, fake object or object loss may exist if there is large noise in edge information or object edges are too weak. Cao [11] introduced an improved Canny edge detector [10] and utilize it to localize the object. By presented Otsu's algorithm on the object region, the binarization result is finally obtained. However, for objects with non-uniform illumination this method does not perform well by only using a non-adaptive global threshold for whole object region.

Methods incorporating edge detector and region growing algorithm have been proposed. Liu [19] chose hot and cold seeds near the edges, and then foreground and background regions are grown from these seeds simultaneously. However, Canny edges may occur inside the object or background region when illumination change

exists there. In this case, there will be spurious object or background in the binarization result.

We first compute the edge map of an image by Canny detector [10]. Then, the skeleton curves are extracted based on the Canny edges. After that, we take skeleton pixels belong to object region as seeds, and present region growing algorithm to obtain the binarization result. We show that, by selecting skeleton pixels as seeds, our method is more robust against the noise in edge image. The experimental results demonstrate our method has a good performance on images with low contrast and non-uniform illumination.

In this paper, we introduce a skeletonization algorithm based on the Skeleton Strength Map (SSM) introduced in [52]. Different from [52], the SSM is computed via isotropic diffusion of the gradient vector field of the distance transform. This provides advantages over both the distance transform and the Skeleton Strength Map used in [52]. Besides, though we do not have an accurate segmentation, by computing the distance transform from the incomplete boundaries, we can also benefit from the existence of the boundary and make the SSM more symmetric.

Skeleton is a very important compact shape descriptor in computer vision since it can preserve both the topological and geometrical information of an object [5]. It has been studied extensively since and plays an important role in areas of object representation and recognition.

Typical skeletonization approaches can be categorized into four types [16]: thinning and boundary propagation [7, 26, 27, 29], geometric methods such as algorithms based on the Voronoi diagram [9, 34, 35], algorithms based on distance transform [12, 21, 31], and algorithms based on general-field functions [2, 17].

Beside these methods, there are also several other kinds of algorithms on skeletonization of binary images. Siddiqi et al. [43] measure the average outward flux of the vector field that underlies the Hamiltonian system and combine the flux measurement with a homotopy preserving thinning process applied in a discrete lattice. This approach leads to a robust and accurate algorithm for computing skeletons in 2D as well as 3D. However, the flux is both limited by the pixel resolution and the error is proportional to the curvature of the boundary evolution front. This makes the exact location of endpoints difficult to be found. An analysis of the system using the Hamilton–Jacobi equations of classical mechanics has shown how the skeleton can be detected using the divergence of the distance map for the object boundary [44]. Torsello and Hancock [47] overcome this problem by taking into account variations of density due to boundary curvature and eliminating the curvature contribution to the error. Aslan and Tari [3] present an unconventional approach for shape recognition using unconnected skeletons in the coarse level. This approach can lead to stable skeletons in the presence of boundary deformations; however, the obtained skeletons do not contain explicit topological structure. In [4], a skeleton pruning method is proposed to obtain stable skeletons via discrete curve evolution.

The algorithms related to our work are based on Euclidean distance transform. They extract the skeleton by detecting ridges on the distance transform surface [12, 21, 31]. Those algorithms can ensure the accurate localization of skeleton points, but they suffer from the problem of robustness: they are very sensitive to boundary noise.

In addition, for these algorithms, there is a common requirement that the complete contour of an object must be known before. However, for gray-scale images, due to the great challenge of segmentation, a complete and robust contour is often unavailable and the distance transform is undefined in gray-scale images. Thus it is difficult to apply the conventional algorithms in gray-scale images. To cope with this issue, many different segmentation-free skeletonization algorithms have been proposed [23, 38, 52]. In [23], a computational algorithm is proposed to compute pseudo-distance map directly from the original image using a nonlinear governing equation. It can extract the skeleton of the narrow objects efficiently without losing information. But for the objects that are wider, this algorithm fails. Anisotropic vector diffusion is used to address the bias problem in [52], but this algorithm also fails to extract the skeleton of wide objects. Also, this method pre-assumes that the object is always brighter or darker than the background as [14] does. There are also other methods. In [45], Tari et al. proposed a method that extracted the skeletons from a set of level-set curves of the edge strength map. Tek et al. [46] have shown that an orientation sensitive distance propagation function can be used to extract symmetries from fragmented contours by labeling skeletal points according to whether or not they represent the collision of consistently oriented boundary fronts. In [32, 37], the authors proposed a method based on scale-space theory which extracts the "cores" from the ridges of a medialness function in scale-space.

The rest of the paper is organized as follows. In Sect. 5.2, we introduce the skeleton strength map. In Sects. 5.3 and 5.4, we introduce the skeletonization approaches for binary images and gray-scale images respectively. In Sect. 5.5 we present the proposed approach to binarization of gray-level images based on their skeletons. In Sect. 5.6 we demonstrate that the proposed approach is able to outperform the state-of-the-art binarization methods on challenging gray-level image.

5.2 Skeleton Strength Map (SSM)

The computation of the Skeleton Strength Map has been introduced in [52] and [30]. Here we briefly review the process and stresses on the differences. The SSM in [52] is based on the anisotropic diffusion of gray-scale images. In this paper, we use isotropic diffusion instead because it is much faster while achieving comparable performance.

5.2.1 Computation of the Skeleton Strength Map

We compute the SSM from the distance transform. The distance transform $dt(\mathbf{r})$ is defined as the distance of an interior point $\mathbf{r}$ to the nearest boundary point [8] shown in Fig. 5.1(a). In gray-scale images, we treat the points on the edge map as boundary points and compute the distance transform. In our approach we compute

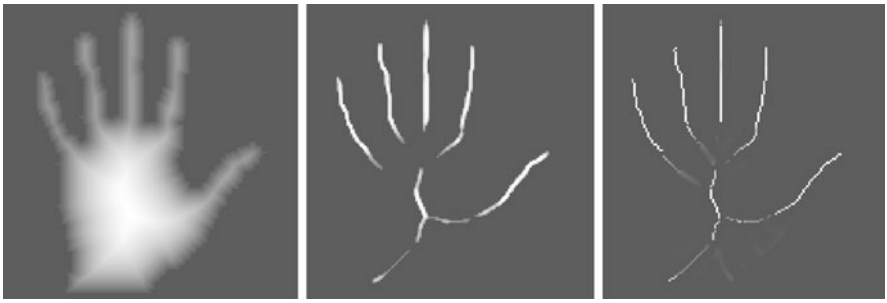


Fig. 5.1 Computation of the SSM. (**a**) shows the distance transform, (**b**) shows the SSM, (**c**) shows the thinned SSM

$f(\mathbf{r}) = 1 - \| \bigtriangledown G_\sigma(\mathbf{r}) * dt(\mathbf{r}) \|$ to replace the distance transform $dt(\mathbf{r})$, where $G_\sigma(\mathbf{r})$ is the Gaussian kernel function, σ is its standard deviation and $*$ is the convolution operator. $f(\mathbf{r})$ can be treated as an inverted version of the smoothed gradient magnitude of $dt(\mathbf{r})$. The main advantage of using $f(\mathbf{r})$ instead of $dt(\mathbf{r})$ is based on the fact that the relative value between skeleton point and it neighbors is significantly larger for $f(\mathbf{r})$ than for $dt(\mathbf{r})$ and it provides an effective way to filter the distortions by boundary noise. Therefore, we work with the gradient vector field of $f(\mathbf{r})$.

Isotropic Diffusion After the gradient vector field of $f(\mathbf{r})$ is computed, the isotropic diffusion is performed. The diffusion process is ruled by a partial differential equation set as in [50],

$$\begin{cases} \dfrac{du}{dt} = \mu \nabla^2 u - (u - f_x)\left(f_x^2 + f_y^2\right) \\ \dfrac{du}{dt} = \mu \nabla^2 v - (v - f_y)\left(f_x^2 + f_y^2\right) \end{cases} \tag{5.1}$$

Here, μ is the regular parameter and is set to 0.07 in our experiments, u, v are two components of the diffused gradient vector field $f(\mathbf{r})$. Initializing u, v with $(u_0, v_0) = \nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$, the partial differential equation (5.1) can be solved.

Skeleton Strength Map To localize the skeleton points from the diffused gradient vector field, we need to compute a Skeleton Strength Map (SSM) from it. In the SSM the value at each point indicates the probability of being a skeleton point, Fig. 5.1(b). The higher the value at a point is, the more likely it is a skeleton point. It is known that the skeleton points are located where two or more vectors confront, so based on this principle, the skeleton strength map is computed by adopting the formula from [52]:

$$SSM(\mathbf{r}) = \max\left(0, \sum_{\mathbf{r}' \in N(\mathbf{r})} \frac{grf(\mathbf{r}') \cdot (\mathbf{r}' - \mathbf{r})}{\|\mathbf{r} - \mathbf{r}'\|}\right), \tag{5.2}$$

where $N(\mathbf{r})$ denotes the eight-neighbors of $\mathbf{r}$. $grf(\cdot)$ is the gradient vector field of $f(\mathbf{r})$. Each of $\mathbf{r}$'s eight-neighbors projects its vector to the unit vector pointing

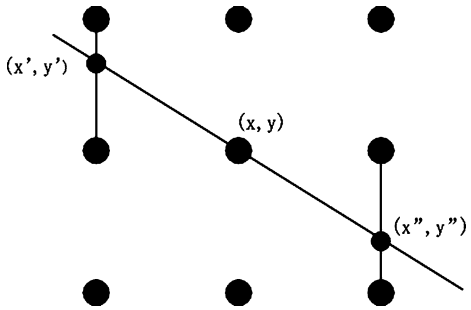


Fig. 5.2 Illustration of non-maximum suppression. (x', y') and (x'', y'') are two intersection points, if $SSM(x, y)$ is greater than $SSM(x', y')$ and $SSM(x'', y'')$, then it's a local maximum and retained

from $\mathbf{r}'$ to $\mathbf{r}$. The intensity of the SSM at $\mathbf{r}$ is then assigned the value of the sum of projections if the sum is positive. The intuition here is that if all the neighbors of $\mathbf{r}$ have gradient vector pointing to it, the intensity of SSM at $\mathbf{r}$ is high and it is likely to be a skeleton point.

Non-Maximum Suppression After the SSM has been computed, we use non-maximum suppression to thin the SSM values. The non-maximum is a very efficient thinning technique that has been applied in edge detection such as Canny operator [10, 22]. For each point (x, y), let $\Theta(x, y)$ be a direction of the gradient vector at (x, y). With reference to Fig. 5.2, we examine two responses $SSM(x', y')$ and $SSM(x'', y'')$ in the adjacent positions (x', y') and (x'', y'') that are intersection points of a line passing through (x, y) with the orientation $\Theta(x, y)$. If the response $SSM(x, y)$ at (x, y) is greater than $SSM(x', y')$ and $SSM(x'', y'')$, i.e., (x, y) is a local maximum, than (x, y) belongs to SSM; otherwise, it will be discarded. The effect of non-maximum suppression is shown in Fig. 5.1(c).

5.2.2 Comparison Between SSM and Distance Transform

We have stated that as the likelihood map of the skeleton, the SSM value decays rapidly at pixels away from center of the object. This can be demonstrated in Fig. 5.3. The distance of pixels inside the hand has value greater than zero, while after diffusion, except for pixels near the skeleton points, other pixels have SSM value zero (in the middle of Fig. 5.3). After non-maximum suppression, the SSM value of some pixels is further suppressed (on the right of Fig. 5.3).

5.3 Skeletonization of Binary Images

The primary goal of this paper is to extract skeletons from gray-scale images. Still, we can show that, this method can achieve good results for binary images. In this section, we introduce the process of skeletonization for binary images.

For binary images, we compute the SSM and thin it with non-maximum suppression. We then select the local maxima as the seeds from the thinned SSM and

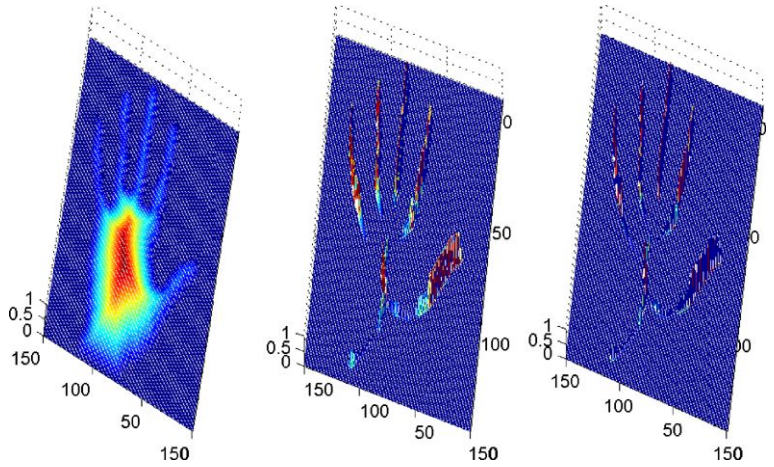


Fig. 5.3 The mesh of the SSM of a hand. On the *left* is the mesh of the distance transform, in the *middle* is the mesh of the SSM and on the *right* is the mesh of the SSM thinned by non-maximum suppression

connect them using the shortest path algorithm (Dijkstra's algorithm) on $f(\mathbf{r})$. An example of skeletonization of binary images is shown in Fig. 5.4.

5.3.1 Local Maxima Detection

Definition 1 A *local maximum* of SSM is a point $\mathbf{r}$ whose SSM value $SSM(\mathbf{r})$ satisfies the following conditions:

$$SSM(\mathbf{r}) \geq \max_{\mathbf{r}' \in N(\mathbf{r})} SSM(\mathbf{r}) \tag{5.3}$$

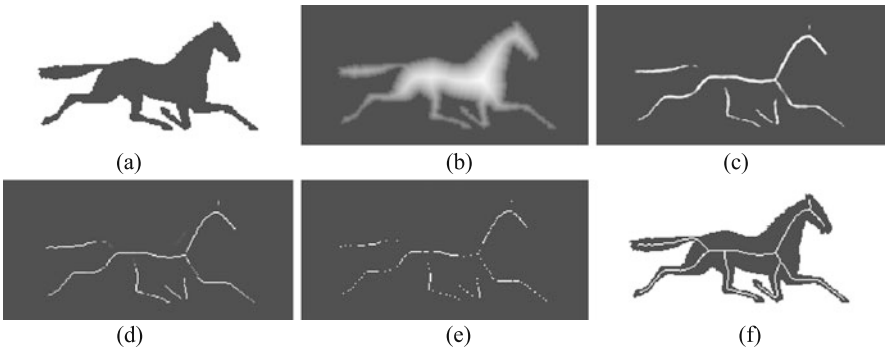


Fig. 5.4 (**a**) is the binary mask of a horse, (**b**) shows the distance transform of (**a**), (**c**) is the SSM computed from (**b**), (**d**) is the SSM thinned by non-maxima suppression, (**e**) shows the local maxima, and (**f**) is the final skeleton

To make the skeleton more robust, we also exclude pixels that have the SSM value $SSM(\mathbf{r}) \leq T$ (generally, T is set to be 0.15). By adjusting the threshold T, we can also get a cleaner skeleton.

The local maxima used in this paper can be regarded as the "seeds" to extract the ridge of the SSM. This is very similar to the "ridge" of the distance transform. Since the SSM is computed from the distance transform, it can locate skeleton accurately as the distance transform does.

5.3.2 Local Maxima Connection

To connect the local maxima, a distance measure is defined based on the surface of function $f(\mathbf{r})$.

Definition 2 Given an 8-connected path $R = \{\mathbf{r_1}, \mathbf{r_2}, \ldots, \mathbf{r_n}\}$, its gradient length is defined as $|R|_G = \sum_{i=1}^{n} |f(\mathbf{r_i})|$.

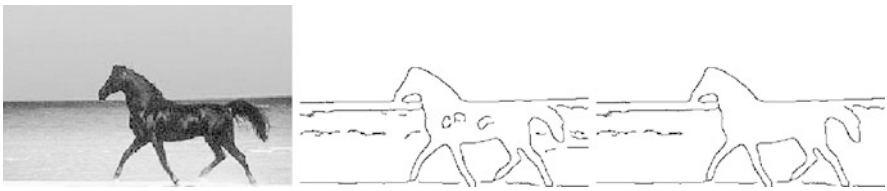
The gradient distance between two points $\mathbf{r}$ and $\mathbf{r}'$ is defined as the minimum over the gradient lengths of all 8-connected paths connecting them. The 8-connected path with the smallest gradient distance is called a gradient path. The gradient path corresponds to a geodesic path on the surface defined by $f(\mathbf{r})$. It is computed with Dijkstra's shortest path algorithm. We obtain the skeleton by connecting the local maxima with gradient paths. To begin, we choose the point having the maximum distance transform as the center of the skeleton and connect iteratively all the other local maxima to it until all the local maxima are connected.

5.4 Skeletonization of Gray-Scale Images

The computation of the distance transform is very natural for binary images, however, for gray-scale images, things are different since we do not have a complete contour. In our approach, we treat all non-boundary pixels as object pixels and boundary points as background and then compute the distance transform. There are many methods to extract boundary, and in this paper we choose the Canny operator [10].

5.4.1 Noise Smoothing of Boundaries

The skeleton is easily influenced by noise. As we hope to extract the skeleton of gray-scale images from boundaries, we should filter the boundary noise. Here we use two heuristics to measure the significance of edge segments: 1) long boundary segments are more significant than short boundary segments and 2) if one short



(a) (b) (c)

Fig. 5.5 Effect of noise smoothing: (**a**) the original image; (**b**) the edge map of (**a**); (**c**) the edge map after noise removed

boundary segment is close to a long segment, it is still possible to compute skeleton branches from them, so we shall treat one short segment more significant if its endpoints are closer to endpoints of a long segment.

To implement these two heuristics, let $S = \{s_i | 1 \leq i \leq N\}$ be all the boundary segments and $L(s_i)$ is the length of s_i. We partition S into two subsets $S_1 = \{s'_i | 1 \leq i \leq N_1\}$ and $S_2 = \{s''_i | 1 \leq i \leq N_2\}$, $N_1 + N_2 = N$, containing segments longer or shorter than a threshold T_1 respectively. For each boundary segment s''_i in S_2, we compute the distance of its endpoints to all endpoints of segments in S_1. If the minimum distance is greater than certain threshold, we treat this segment as noise and remove it. This process is straightforward and an example is given shown in Fig. 5.5. Figure 5.5(b) shows the edge of Fig. 5.5(a) obtained by canny operator, and Fig. 5.5(c) shows the edges after removing insignificant segments. We can see from this figure, removing some short segments with endpoints distant from endpoints of long segments does not affect the whole shape of the horse.

5.4.2 Computation of SSM from Gray-Scale Images

To compute the SSM, the boundary values are firstly reversed and the Euclidean distance transform (shown in Fig. 5.6(a)) is performed. We then compute the SSM and perform non-maximum suppression as described in Sect. 5.2, e.g., see Fig. 5.6(b). A small problem exists for gray-scale images: as we can see in Fig. 5.6(c), there are some high values of SSM caused by edges. It is easy to remove them, since we know the boundary in advance; we can assign each edge pixel a window of size $k \times k$ and assign SSM values of zero to pixels within that window ($k = 3$ is generally sufficient). Figure 5.6(d) shows the SSM after edge strength values are removed.

By adopting hysteresis thresholding, we obtain the skeleton shown in Fig. 5.6(e). The same skeleton overlaid on the original gray level image is shown in Fig. 5.6(f). The skeleton obtained in this way is not connected, as we have mentioned before, since we have no position information about the object.

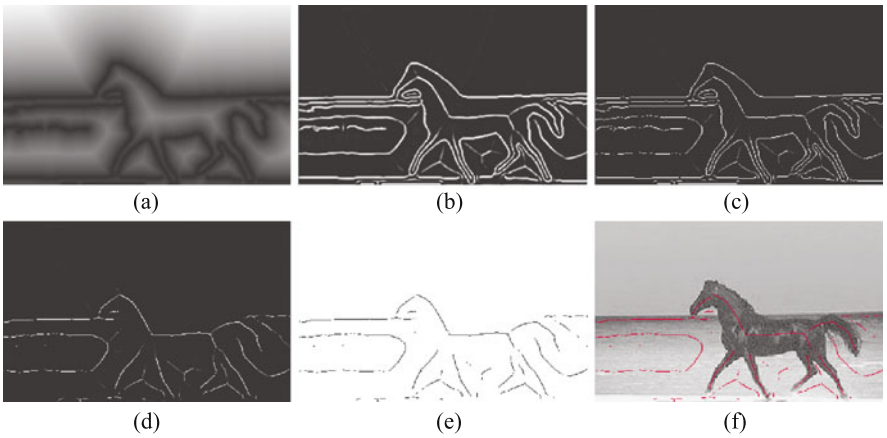


Fig. 5.6 An example of skeletonization of gray level images from boundaries: (**a**) the distance transform; (**b**) the SSM; (**c**) thinned SSM; (**d**) is the SSM with values around original edges removed; (**e**) is the final skeleton by hysteresis thresholding; (**f**) the final skeleton overlaid on the original image

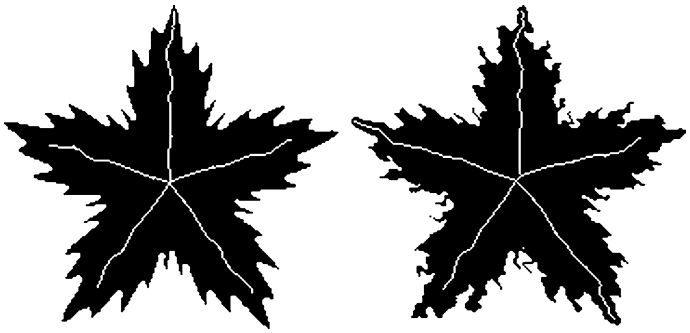


Fig. 5.7 The skeleton is robust to boundary noise

5.4.3 Robustness Under Boundary Noise and Deformation

Figure 5.7 provides two examples that show the insensitivity of the proposed method to boundary noise. The two stars both have substantial noise on the contours, however, the two skeletons obtained are very similar and preserve the topological and geometric structure of the two stars with no spurious branches. This demonstrates the robustness of our algorithm (for both stars, the parameter σ is set as 2.5). The computation of the SSM provides an effective way to filter boundary noise. In Fig. 5.8, by varying σ from 1, 1.5, to 2.5, more robust skeleton can be generated for the left star in Fig. 5.7.

Fig. 5.8 The skeleton smoothing obtained by varying the parameter σ

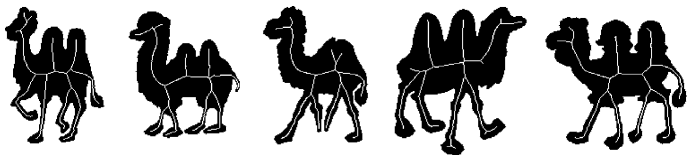


Fig. 5.9 Comparative study of the algorithm on several camels with great inner variations

The stability of our algorithm in the presence of large inner-class shape variations is demonstrated in Fig. 5.9. Although these camels have different poses and differ significantly from each other, the obtained skeletons have the same global structure.

5.4.4 Results on Gray-Scale Images

In Fig. 5.10, two examples of skeletons obtained from boundary of gray-scale images are provided. Column (a) shows the edge images obtained by Canny operator. We observe that though there are lots of gaps, e.g., gaps on back of the dog, and gaps on feet of the horse, good SSM can still be computed as shown in column (b). Column (c) shows the skeleton (in red) obtained by thresholding the SSM. This illustrates the ability of our approach to extract the skeleton from incomplete boundaries. Compared to the skeleton obtained by [52] shown in column (d), two advantages have been achieved. One is that the skeletons obtained by our method are more symmetric to the objects' boundary than [52], e.g., the torsos of the dog and the horse. The reason for this is attributed to the usage of boundaries and the distance transform, while the algorithm in [52], it fails to locate the skeleton points accurately when the object is wide. The other advantage is that, the results are more complete than the results of [52]. For example, the skeletons of the dog and the horse do not have branches corresponding to the 4 limbs, while our algorithm generates such branches and main branches are preserved.

These experiments have demonstrated that the SSM can be readily extended to the case of gray-scale images and that the good skeletons can be computed even when the boundaries have many gaps, clutters and noise.

Fig. 5.10 Skeleton results based on the SSM computed from Canny edges. (**a**) shows the edge images obtained by Canny operator. (**b**) shows our SSM. (**c**) shows our skeletons. (**d**) shows skeletons obtained by the method in [52]

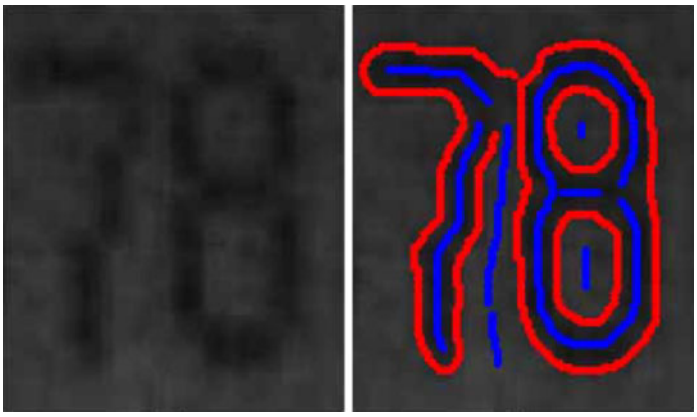
5.5 Binarization of Gray-Level Images Based on Their Skeletons

5.5.1 Seeds Selection

We aim to select foreground object skeletons of gray-level images as seeds. There are two main benefits of using skeletons as seed candidates. First, skeleton points belonging to the same region has spatial connectivity, which makes them easy to be grouped into one curve. This enables us to consider skeleton points in the same region as a whole, and have a global instead of local perception about whether these skeleton pixels belong to a foreground or background region. Second, the number of skeleton points will adaptively increase or decrease when a region expands or shrinks. Therefore, we do not need to specify how many seeds are needed for a region in advance. We illustrate these facts in Fig. 5.11.

5.5.2 Classifying Skeleton Segments into Foreground and Background

We start with grouping skeleton pixels in the same region. Since skeleton pixels in the same region are spatially adjacent, they can be easily grouped according to their connectivity. Here, the edge-linking algorithm [28] is used to achieve this goal. In practice, we first compute the edge image E using Canny edge detector on gray-scale image I. Then we generate the skeleton image S using SSM of the distance transform on edge image E. By applying edge-linking algorithm on S, we obtain n skeleton curves, i.e., $S = SS_1 \cup SS_2 \cup \ldots SS_n$. For each grouped skeleton curve,



(a) (b)

Fig. 5.11 (**a**) is a gray image of two characters. (**b**) shows Canny edges in *red* and the skeletons in *blue*

our goal is to classify it into foreground or background. Therefore, we calculate the average intensity value of its contained skeleton pixels, then compare it with the gray intensity level of the region boundary, i.e., nearby edges. The average intensity $G(SS_i)$ for skeleton curve SS_i is calculated as $G(SS_i) = \frac{1}{|SS_i|}\sum_{j=1}^{|SS_i|} |I(ss_j)|$, $ss_j \in SS_i$. Since we want to compare $G(SS_i)$ with the gray intensity level of the nearby boundaries, we need to determine the local region $R(SS_i)$ of skeleton curve SS_i first. $R(SS_i)$ is defined as a band containing SS_i, that is: for every pixel $m \in R(SS_i)$, $\exists ss_j \in SS_i$, $|m - ss_j| < r$, r indicates the radius of the band. Thus, the edge points in the local region of SS_i is $LE_i = R(SS_i) \cap E$. For every edge point $e \in E$, its intensity level is indicated by its 8-neighborhood minimum intensity $f(e) = \min I(n_e^i)$ for $i = 1 \ldots 8$. n_e^i is the neighborhood of edge pixel e. For the nearby edge pixels LE_i of skeleton curve SS_i, its average 8-neighborhood minimum intensity $M(LE_i) = \frac{1}{|LE_i|}\sum_{j=1}^{|LE_i|} |f(le_j)|$, $le_i \in LE_i$. Thus, if the average intensity value $G(SS_i)$ of skeleton curve is lower than the gray intensity level $M(LE_i)$ of the nearby edges, the skeleton pixels are considered belonging to foreground region, so we add pixels of SS_i to foreground skeleton OS. Otherwise, they are considered belonging to background. Here, we assume object regions have lower intensity than background regions. As mentioned above, we will use foreground skeleton pixels OS as seeds to perform the region growing algorithm.

5.5.3 Dynamic Threshold Computation

Similar to Sauvola's method [40], we also have a dynamic threshold $T(i, j)$ for each pixel (i, j). We determine the adaptive threshold $T(i, j)$ for edge and non-edge pixel

respectively.

$$T(i,j) = \begin{cases} f(i,j) & \text{if } (i,j) \in E \\ \frac{1}{|W(i,j) \cap E|} \sum_{n=1}^{|W(i,j) \cap E|} |f(i_n, j_n)| & \\ \quad (i_n, j_n) \in W(i,j) \cap E & \text{if } (i,j) \notin E \end{cases}$$

Here, $W(i,j)$ is the rectangle region whose center is (i,j), and function $f(i,j)$ has the same definition as in Sect. 5.5.2, which computes the 8-neighborhood lowest intensity of pixel (i,j). The dynamic threshold $T(i,j)$ is used as one of the criteria for the following region growing algorithm.

5.5.4 Region Growing Algorithm

We take every pixel ob_i of object skeleton OS as a seed point to perform a 4-neighborhood region growing algorithm. $RG(ob_i)$ represents its growing region, $RG(ob_i) = \{p_1, p_2 \dots p_n\}$. If adjacent pixel (i,j) satisfies at least one of the following criteria, (i,j) is added in to $RG(ob_i)$, and is classified as object pixel.

(1) $|I(i,j) - \frac{1}{n}\sum_{k=1}^{n} I(p_k)| < t,\ p_k \in RG(ob_i)$
(2) $I(i,j) \leq T(i,j)$

Therefore, the final binarization result is obtained by aggregating all $RG(ob_i)$.

5.6 Experimental Results and Analysis

Our experimental images are degraded document images, NDT images, and images with non-uniform illumination. We compare the performance of our method to classical binarization methods that also utilize edge information. Figure 5.12 shows a comparison between 7 binarization methods on 2 degraded document images in which smear and low contrast exist. Our method is compared to methods by Otsu [36], Kapur [24], Niblack [33], Kittler [25], Yanowitz and Bruckstein [51], Sauvola and Pietikainen [40]. Among these methods, [40] and our method perform better than the others. Niblack's method cannot resist the noise brought by inhomogeneous intensity in the background region, and therefore, generates many fake object pixels in binarization results. The other 4 methods cannot solve the smear problem in these images.

Figure 5.13(a) shows two NDT images with ground truth segmentation in (b). Here our method is compared to methods by Bernsen [6], Niblack [33], Otsu [36], Kittler [25], and Yanowitz and Bruckstein [51]. Otsu's, YB's and our methods provide better performance while the others show weak resistance against the local noise in the background region.

Star and PCB images in Fig. 5.14(a) both exhibit sharp non-uniform illumination. Here our method is compared to methods Otsu [36], Sauvola and Pietikainen [40],

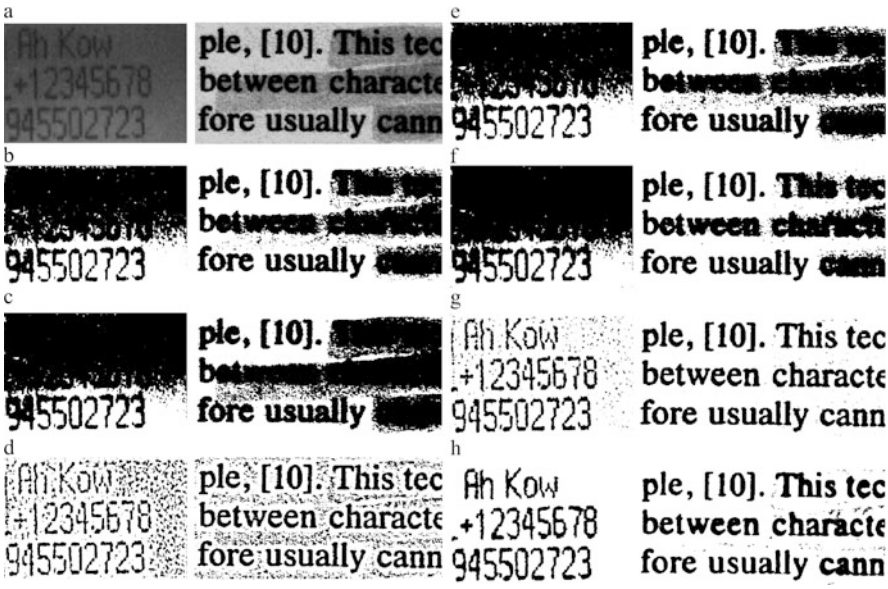


Fig. 5.12 Comparison between 7 binarization methods with 2 degraded document images: (**a**) two degraded document images; (**b**) Otsu's method; (**c**) Kapur's method; (**d**) Niblack's method; (**e**) Kittler's method; (**f**) YB's method; (**g**) Sauvola's method; (**h**) our method

Bernsen [6], Niblack [33], Kittler [25], and Yanowitz and Bruckstein [51]. Only our method provides very good binarization results. From other methods, only YB's method yields acceptable results on both images.

5.7 Future Work and Open Problems

In this paper, we propose a novel binarization technique that utilizes both Canny edge detector and seeded region growing algorithm. The main idea of the proposed binarization technique is to take skeleton pixels belonging to different regions as seed candidates, and then according to their local edge information, we classify them into two groups. We select objects' skeleton pixels as seeds to a region growing algorithm. Several experimental results are presented that confirm the effectiveness of the proposed binarization method. For images with low contrast, noise and non-uniform illumination, our method can overcome the deficiencies of some classical methods.

We observe that the Skeleton Strength Map (SSM) can be computed for any gray level image, e.g., Fig. 5.6, and consequently, for any color 2D image. We only need to compute its edge map and then SSM on its gray level version. Thus, theoretically the proposed method can be used to obtain a binary segmentation into foreground and background objects of any color image.

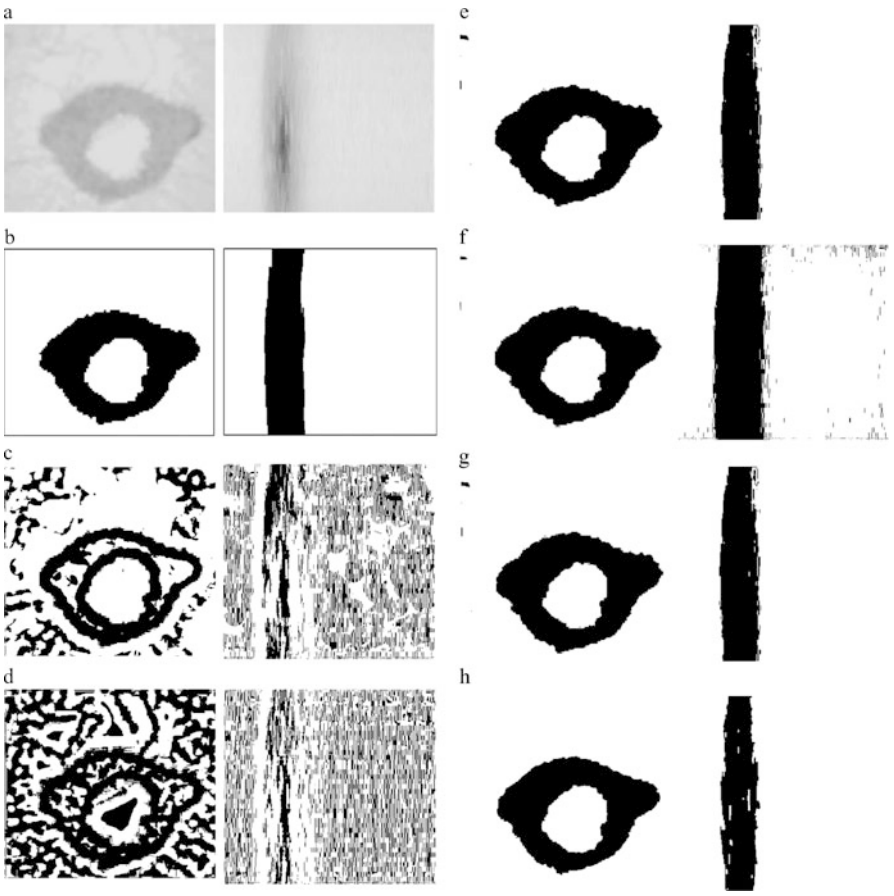


Fig. 5.13 Comparison between 7 binarization methods with 2 NDT images: (**a**) two NDT images; (**b**) Ground truth; (**c**) Bernsen's method; (**d**) Niblack's method; (**e**) Otsu's method; (**f**) Kittler's method; (**g**) YB's method; (**h**) our method

The key challenge in using this method to segment a color image is to determine which SSM skeletons belong to the foreground objects. This seems to be the main open problem for this very important application. Consequently, a positive answer to this problem will allow us to utilize the proposed method in the unsupervised image segmentation.

Unsupervised image segmentation is a fundamental and challenging problem in computer vision. The goal is to segment an image into several regions without any prior knowledge. There is a huge literature on this topic. Some of the most influential approaches are [1, 15, 18, 20, 42, 49]. Most of them focus on minimizing an energy function while assigning region labels to image pixels.

A serious challenge these approaches face is the locality of the affinity relation between pixels or regions, which we call a locality problem. Since pixels with sim-

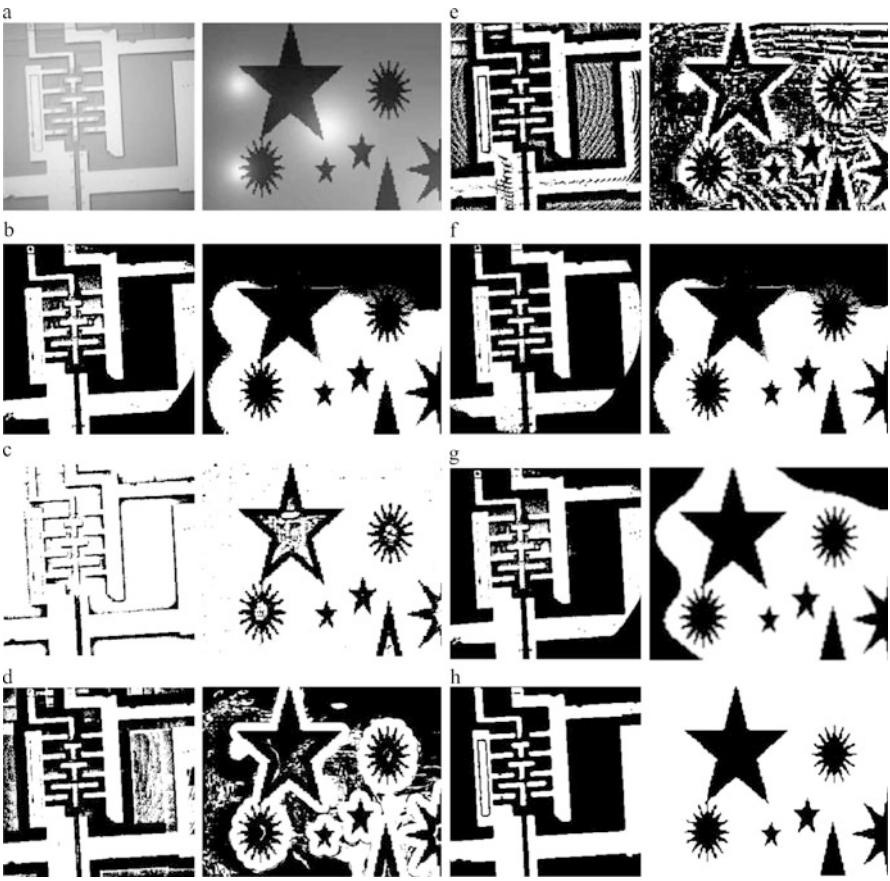


Fig. 5.14 Comparison between 7 binarization methods with 2 non-uniform illumination images: (**a**) Star and PCB images; (**b**) Otsu's method; (**c**) Sauvola's method; (**d**) Bernsen's method; (**e**) Niblack's method; (**f**) Kittler's method; (**g**) YB's method; (**h**) our method

ilar colors in a small neighborhood are more likely to belong to the same object than such pixels in a large neighborhood, only local affinities are input to unsupervised image segmentation algorithms. Obviously, adding long range information could lead to improved performance. However, doing so not only increases computation cost but often creates a challenge of finding relevant long range information or, equivalently, filtering out irrelevant or noisy information.

A simple step towards solving the locality problem is using regions (superpixels) instead of pixels. Superpixels have been used in image segmentation recently, e.g., in [49]. Although superpixels alleviate this problem and lead to more robust affinity relations, their affinity relation still remains local, e.g., only adjacent regions are related in [49]. Moreover, superpixels are usually small regions, since otherwise they can erase object boundary information. In other words, the problem of local affinity relations also applies to superpixels.

The proposed methods solves the locality problem by utilizing the SSM skeletons of the foreground objects. Simply speaking, pixels of a single SSM skeleton are very likely to belong to the same object. Therefore, region growing starting from this skeleton seems to provide an adequate solution to the locality problem.

Our future work will focus on integrating the proposed method, and, in particular, the global information of the SSM skeletons of foreground objects in the framework of superpixel based image segmentation. As stated above, the main challenge is to determine which SSM skeletons belong to foreground objects.

References

1. Arbelaez, P., Maire, M., Fowlkes, C.C., Malik, J.: From contours to regions: an empirical evaluation. In: CVPR, pp. 2294–2301 (2009)
2. Ahuja, N., Chuang, J.: Shape representation using a generalized potential field model. IEEE Trans. Pattern Anal. Mach. Intell. **19**(2), 169–176 (1997)
3. Aslan, C., Tari, S.: An axis based representation for recognition. In: ICCV, pp. 1339–1346 (2005)
4. Bai, X., Latecki, L.J., Liu, W.: Skeleton pruning by contour partitioning with discrete curve evolution. IEEE Trans. Pattern Anal. Mach. Intell. **29**(3), 449–462 (2007)
5. Blum, H.: Biological shape and visual science (Part I). J. Theor. Biol. **38**, 205–287 (1967)
6. Bernsen, J.: Dynamic thresholding of gray-level images. In: Proceedings of the Eighth International Conference on Pattern Recognition, pp. 1251–1255. IEEE Comput. Soc., Paris (1986)
7. Bertrand, G., Aktouf, Z.: A three-dimensional thinning algorithm using subfields. In: Vision Geometry. Proc. SPIE, vol. 2356, pp. 113–124 (1995)
8. Borgefors, G.: Distance transformations in digital images. Comput. Vis. Graph. Image Process. **34**, 344–371 (1986)
9. Brandt, J.W., Alazi, V.R.: Continuous skeleton computation by Voronoi diagram. CVGIP, Image Underst. **55**(3), 329–338 (1992)
10. Canny, J.: A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**(6), 679–698 (1986)
11. Cao, R., Tan, C.L., Wang, Q., Shen, P.: Segmentation and analysis of double-sided handwritten archival documents. In: Proceedings of the Fourth IAPR Intl. Workshop Document Analysis Systems, pp. 147–158 (2000)
12. Choi, W.P., Lam, K.M., Siu, W.C.: Extraction of the Euclidean skeleton based on a connectivity criterion. Pattern Recognit. **36**(3), 721–729 (2003)
13. Chow, C.K., Kaneko, T.: Automatic boundary detection of the left-ventricle from cineangiograms. Comput. Biomed. Res. **5**, 388–410 (1972)
14. Chung, D.H., Sapiro, G.: Segmentation-free skeletonization of gray-scale images via PDEs. In: ICIP, pp. 927–930 (2000)
15. Comaniciu, D., Meer, P.: Mean shift: a robust approach toward feature space analysis. IEEE Trans. Pattern Anal. Mach. Intell. **24**(5), 603–619 (2002)
16. Cornea, N.D., Silver, D., Min, P.: Curve-skeleton properties, applications, and algorithms. IEEE Trans. Vis. Comput. Graph. **13**(3), 520–548 (2007)
17. Cornea, N.D., Demirci, M.F., Silver, D., Shokoufandeh, A., Dickinson, S.J., Kantor, S.J.: 3D object retrieval using many-to-many matching of curve-skeletons. In: Proc. Shape Modeling International, pp. 366–371 (2005)
18. Cour, T., Benezit, F., Shi, J.: Spectral segmentation with multiscale graph decomposition. In: CVPR, pp. 1124–1131 (2005)
19. Fan, S., Man, Z., Samur, R.: Edge based region growing: a new image segmentation method. In: Proceedings of the 2004 ACM SIGGRAPH International Conference on Virtual Reality Continuum and Its Applications in Industry, pp. 302–305 (2004)

20. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient graph-based image segmentation. Int. J. Comput. Vis. **59**(2), 167–181 (2004)
21. Ge, Y., Fitzpatrick, J.M.: On the generation of skeletons from discrete Euclidean distance maps. IEEE Trans. Pattern Anal. Mach. Intell. **18**(11), 1055–1066 (1996)
22. Grigorescu, C., Petkov, N., Westenberg, M.A.: Contour and boundary detection improved by surround suppression of texture edges. Image Vis. Comput. **22**(8), 609–622 (2004)
23. Jang, J.H., Hong, K.S.: A pseudo-distance map for the segmentation-free skeletonization of gray-scale images. In: ICCV, pp. 18–23 (2001)
24. Kapur, J.N., Sahoo, P.K., Wong, A.K.: A new method for gray-level picture thresholding using the entropy of the histogram. Comput. Vis. Graph. Image Process. **29**(3), 273–285 (1985)
25. Kittler, J., Illingworth, J., Foglein, J.: Minimum error thresholding. Pattern Recognit. **19**(1), 41–47 (1986)
26. Kong, T.Y., Rosenfeld, A.: Digital topology: introduction and survey. Comput. Vis. Graph. Image Process. **48**(3), 357–393 (1989)
27. Kong, T.Y., Roscoe, A.W., Rosenfeld, A.: Concepts of digital topology. Topol. Appl. **46**(3), 219–262 (1992)
28. Kovesi, P.: http://www.csse.uwa.edu.au/~pk/research/matlabfns/
29. Lam, L., Lee, S.W., Suen, C.Y.: Thinning methodologies—a comprehensive survey. IEEE Trans. Pattern Anal. Mach. Intell. **14**(9), 869–885 (1992)
30. Latecki, L.J., Li, Q., Bai, X., Liu, W.: Skeletonization using SSM of the distance transform. In: ICIP, pp. 349–352 (2007)
31. Leymarie, F., Levine, M.: Simulating the grassfire transform using an active contour model. IEEE Trans. Pattern Anal. Mach. Intell. **14**(1), 56–75 (1992)
32. Morse, B.S., Pizer, S.M., Puff, D.T., Gu, C.: Zoom-invariant vision of figural shape: effects on cores of images disturbances. Comput. Vis. Image Underst. **69**, 72–86 (1998)
33. Niblack, W.: An Introduction to Digital Image Processing. Prentice Hall, Englewood Cliffs (1986)
34. Ogniewicz, R.: A multiscale MAT from Voronoi diagrams: the skeleton-space and its application to shape description and decomposition. In: Aspects of Visual Form Processing, pp. 430–439 (1994)
35. Ogniewicz, R., Kübler, O.: Hierarchic Voronoi skeletons. Pattern Recognit. **28**(3), 343–359 (1995)
36. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Trans. Syst. Man Cybern. **9**(1), 62–66 (1979)
37. Pizer, S.M., Eberly, D., Fritsch, D.S., Morse, B.S.: Zoom-invariant vision of figural shape: the mathematics of cores. Comput. Vis. Image Underst. **69**, 55–71 (1998)
38. Qiu, H., Hancock, E.R.: Grey scale image skeletonisation from noise-damped vector potential. In: ICPR, pp. 839–842 (2004)
39. Saha, B.N., Ray, N.: Image thresholding by variational minimax optimization. Pattern Recognit. **42**, 843–856 (2009)
40. Sauvola, J., Pietikainen, M.: Adaptive document image binarization. Pattern Recognit. **33**(2), 225–236 (2000)
41. Sezgin, M., Sankur, B.: Survey over image thresholding techniques and quantitative performance evaluation. J. Electron. Imaging **13**, 146 (2004)
42. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **22**, 888–905 (2000)
43. Siddiqi, K., Bouix, S., Tannenbaum, A., Zucker, S.W.: The Hamilton-Jacobi skeleton. In: ICCV, pp. 828–864 (1999)
44. Siddiqi, K., Shokoufandeh, A., Dickinson, S.J., Zucker, S.W.: Shock graphs and shape matching. Int. J. Comput. Vis. **35**(1), 13–32 (1999)
45. Tari, S., Shah, J., Pien, H.: Extraction of shape skeletons from gray-scale images. Comput. Vis. Image Underst. **66**, 133–146 (1997)
46. Tek, H., Stoll, P.A., Kimia, B.B.: Shocks from images: propagation of orientation elements. In: CVPR, pp. 839–845 (1997)

47. Torsello, A., Hancock, E.R.: Correcting curvature-density effects in Hamilton-Jacobi skeleton. IEEE Trans. Image Process. **15**(4), 877–891 (2006)
48. Trier, O.D., Jain, A.K.: Goal-directed evaluation of binarization methods. IEEE Trans. Pattern Anal. Mach. Intell. **17**(12), 1191–1201 (1995)
49. Wang, J., Jia, Y., Hua, X.-S., Zhang, C., Quan, L.: Normalized tree partitioning for image segmentation. In: CVPR, pp. 1–8 (2008)
50. Xu, C., Prince, J.L.: Snakes, shapes, and gradient vector flow. IEEE Trans. Image Process. **7**(3), 359–369 (1998)
51. Yanowitz, S.D., Bruckstein, A.M.: A new method for image segmentation. Comput. Vis. Graph. Image Process. **46**(1), 82–95 (1989)
52. Yu, Z., Bajaj, C.: A segmentation-free approach for skeletonization of gray-scale images via anisotropic vector diffusion. In: CVPR, pp. 18–23 (2004)

Chapter 6
Topology Preserving Parallel 3D Thinning Algorithms

Kálmán Palágyi, Gábor Németh, and Péter Kardos

Abstract A widely used technique to obtain skeletons of binary objects is thinning, which is an iterative layer-by-layer erosion in a topology preserving way. Thinning in 3D is capable of extracting various skeleton-like shape descriptors (i.e., centerlines, medial surfaces, and topological kernels). This chapter describes a family of new parallel 3D thinning algorithms for (26, 6) binary pictures. The reported algorithms are derived from some sufficient conditions for topology preserving parallel reduction operations, hence their topological correctness is guaranteed.

6.1 Introduction

Skeleton is a region-based shape descriptor which represents the general shape of objects. 3D skeleton-like shape features (i.e., centerlines, medial surfaces, and topological kernels) play important role in various applications in image processing, pattern recognition, and visualization [6, 10, 38, 41, 42, 44].

An illustrative definition of the skeleton uses the prairie-fire analogy: the object boundary is set on fire, and the skeleton is formed by the loci where the fire fronts meet and extinguish each other [5]. Thinning is a digital simulation of the fire front propagation: the border points that satisfy certain topological and geometric constraints are deleted in iteration steps [12].

A *3D binary picture* [11, 12] is a mapping that assigns a value of 0 or 1 to each point with integer coordinates in the 3D digital space $\mathbb{Z}^3$. Points having the value of 1 are called *black* points, and those with a zero value are called *white* ones. Black points form the components of a picture, while white points form the background and the cavities. We consider (26, 6)-pictures, where 26-adjacency and 6-adjacency are, respectively, used for the components and their complement [12].

K. Palágyi (✉) · G. Németh · P. Kardos
Institute of Informatics, University of Szeged, Szeged, Hungary
e-mail: palagyi@inf.u-szeged.hu

G. Németh
e-mail: gnemeth@inf.u-szeged.hu

P. Kardos
e-mail: pkardos@inf.u-szeged.hu

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_6,

A *reduction operation* transforms a binary picture only by changing some black points to white ones (which is referred to as the *deletion* of 1's). A *parallel reduction operation* deletes all points satisfying its condition simultaneously. A reduction operation does *not* preserve topology [11] if

- any component in the input picture is split (into several components) or is completely deleted,
- any cavity in the input picture is merged with the background or another cavity, or
- a cavity is created where there was none in the input picture.

There is an additional concept called *hole (or tunnel)* in 3D pictures. A hole (which doughnuts have) is formed of 0's, but it is not a cavity [12]. Topology preservation implies that eliminating or creating any hole is not allowed.

There are three types of 3D thinning algorithms for producing the three types of skeleton-like shape features: *curve-thinning* algorithms are used to extract medial lines or centerlines, *surface-thinning* algorithms produce medial surfaces, while *kernel-thinning* algorithms are capable of extracting topological kernels. A topological kernel is a minimal set of points that is topologically equivalent [12] to the original object (i.e., if we remove any further point from it, then the topology is not preserved). Note that kernel-thinning algorithms are often referred to as reductive shrinking algorithms [9]. 3D curve-thinning and surface-thinning algorithms use operations that delete some points which are not *endpoints*, since preserving endpoints provides important geometrical information relative to the shape of the objects. Kernel-thinning algorithms for extracting topological kernels do not take any endpoint characterization into consideration. Medial surfaces are usually extracted from general shapes, tubular structures can be represented by their centerlines, and extracting topological kernels is useful in topological description.

Most of the existing thinning algorithms are parallel as the fire front propagation is by nature parallel. These algorithms are composed of parallel reduction operations. Parallel reduction operations delete a set of points simultaneously which may lead to altering the topology. Note that deletion rules of parallel thinning algorithms are generally given by matching templates. In order to verify that a given parallel 3D thinning algorithm preserves the topology for all possible (26, 6) pictures, some sufficient conditions for topology preservation have been proposed [11, 18, 36]. However, verifying these conditions usually means checking several configurations of points, hence papers presenting thinning algorithms contain long proof parts. Despite of complex proofs, it was claimed in [14, 45] that two parallel 3D thinning algorithms [18, 19] are not topology preserving. That is why we propose a safe technique for designing topologically correct parallel 3D thinning algorithms. Our approach is based on some new sufficient conditions for topology preservation. These conditions consider individual points (instead of point configurations) and can be combined with various thinning strategies.

In this chapter we present 15 algorithms that are derived from the new sufficient conditions combined with the three major strategies for parallel thinning (i.e., fully parallel, subiteration-based, and subfield-based [8]) and three types of endpoints.

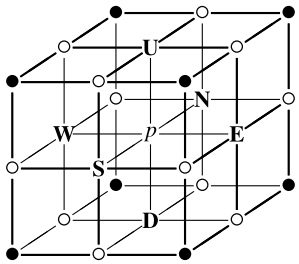


Fig. 6.1 Frequently used adjacencies in $\mathbb{Z}^3$. The set $N_6(p)$ contains point p and the six points marked **U**, **D**, **N**, **E**, **S**, and **W**. The set $N_{18}(p)$ contains $N_6(p)$ and the twelve points marked by "○". The set $N_{26}(p)$ contains $N_{18}(p)$ and the eight points marked by "●"

The rest of this chapter is organized as follows. Section 6.2 reviews the basic notions and results of 3D digital topology, and we present our sufficient conditions for topology preservation. Then, in Sect. 6.3 we propose 15 parallel 3D thinning algorithms and their topological correctness is proved. Since fast extraction of skeleton-like shape features is extremely important in numerous applications for large 3D shapes, Sect. 6.4 is devoted to the efficient implementation of the proposed algorithms, and Sect. 6.5 presents some illustrative results. In Sect. 6.6 some possible future works and open problems are outlined. Finally, we round off the chapter with some concluding remarks.

6.2 Topology Preserving Parallel Reduction Operations

In this section, we present new sufficient conditions for topology preservation. First we outline some concepts of digital topology and related key results that will be used in the sequel.

Let p be a point in the 3D digital space $\mathbb{Z}^3$. Let us denote $N_j(p)$ (for $j = 6, 18, 26$) the set of points that are *j-adjacent* to point p (see Fig. 6.1).

The sequence of distinct points $\langle x_0, x_1, \ldots, x_n \rangle$ is called a *j-path* (for $j = 6, 26$) of length n from point x_0 to point x_n in a non-empty set of points X if each point of the sequence is in X and x_i is j-adjacent to x_{i-1} for each $1 \leq i \leq n$ (see Fig. 6.1). Note that a single point is a j-path of length 0. Two points are said to be *j-connected* in the set X if there is a j-path in X between them ($j = 6, 26$). A set of points X is *j-connected* in the set of points $Y \supseteq X$ if any two points in X are j-connected in Y ($j = 6, 26$).

A *3D binary* $(26, 6)$ *digital picture* $\mathscr{P}$ is a quadruple $\mathscr{P} = (\mathbb{Z}^3, 26, 6, B)$ [12]. Each element of $\mathbb{Z}^3$ is called a *point* of $\mathscr{P}$. Each point in $B \subseteq \mathbb{Z}^3$ is called a *black point* and has a value 1. Each point in $\mathbb{Z}^3 \setminus B$ is called a *white point* and has a value 0. An *object* is a maximal 26-connected set of black points, while a *white component* is a maximal 6-connected set of white points. Here it is assumed that a picture contains finitely many black points.

The *lexicographical order* relation "$\prec$" between two distinct points $p = (p_x, p_y, p_z)$ and $q = (q_x, q_y, q_z)$ in $\mathbb{Z}^3$ is defined as follows:

$$p \prec q \quad \Leftrightarrow \quad (p_z < q_z) \vee (p_z = q_z \wedge p_y < q_y) \vee (p_z = q_z \wedge p_y = q_y \wedge p_x < q_x).$$

Let $C \subseteq \mathbb{Z}^3$ be a set of points. Point $p \in C$ is the *smallest element* of C if for any $q \in C \backslash \{p\}$, $p \prec q$.

A *unit lattice square* is a set of four mutually 18-adjacent points in $\mathbb{Z}^3$, while a *unit lattice cube* is a set of eight mutually 26-adjacent points in $\mathbb{Z}^3$.

A black point is called a *border point* in $(26, 6)$ pictures if it is 6-adjacent to at least one white point. A border point p is called a **U**-*border point* if the point marked $\mathbf{U} = u(p)$ in Fig. 6.1 is a white point. We can define **D**-, **N**-, **E**-, **S**-, and **W**-border points in the same way. A black point is called an *interior point* if it is not a border point. A *simple* point in a $(26, 6)$ picture is a black point whose deletion is a topology preserving reduction operation [12]. Note that simplicity of point p in $(26, 6)$ pictures is a local property that can be decided by investigating the set $N_{26}(p)$ [12].

Parallel reduction operations delete a set of black points and not just a single simple point. Hence we need to consider what is meant by topology preservation when a number of black points are deleted simultaneously.

Ma [17] gave some *sufficient conditions* for 3D parallel reduction operations to preserve topology. Later, Palágyi and Kuba proposed the following simplified conditions [36]:

Theorem 1 ([36]) *The parallel reduction operation $\mathcal{O}$ is topology preserving for* $(26, 6)$ *pictures if all the following conditions hold.*

1. *Only simple points are deleted by $\mathcal{O}$.*
2. *Let p be any black point in a picture $(\mathbb{Z}^3, 26, 6, B)$ such that p is deleted by $\mathcal{O}$. Let $Q \subseteq B$ be any set of simple points in $(\mathbb{Z}^3, 26, 6, B)$ such that $p \in Q$, and Q is contained in a unit lattice square. Then point p is simple in picture $(\mathbb{Z}^3, 26, 6, B \backslash (Q \backslash \{p\}))$.*
3. *No object contained in a unit lattice cube is deleted completely by $\mathcal{O}$.*

Theorem 1 provides a general method of verifying that a parallel thinning algorithm preserves topology. In this section, we present some new sufficient conditions for topology preservation as a basis for designing 3D parallel thinning algorithms.

Theorem 2 *The parallel reduction operation $\mathcal{O}$ is topology preserving for* $(26, 6)$ *pictures if all the following conditions hold for any black point p in any picture $(\mathbb{Z}^3, 26, 6, B)$ such that p is deleted by $\mathcal{O}$.*

1. *Point p is simple in $(\mathbb{Z}^3, 26, 6, B)$.*
2. *Let $Q \subseteq B$ be any set of simple points in $(\mathbb{Z}^3, 26, 6, B)$ such that $p \in Q$, and Q is contained in a unit lattice square. Then point p is simple in picture $(\mathbb{Z}^3, 26, 6, B \backslash (Q \backslash \{p\}))$, or p is not the smallest element of Q.*
3. *Point p is not the smallest element of any object $C \subseteq B$ in $(\mathbb{Z}^3, 26, 6, B)$ such that C is contained in a unit lattice cube.*

Proof It can be readily seen that if the parallel reduction operation $\mathcal{O}$ satisfies Condition i of Theorem 2, then Condition i of Theorem 1 is also satisfied ($i =$

1, 2, 3). Hence, parallel reduction operation $\mathscr{O}$ is topology preserving for (26, 6) pictures. □

6.3 Variations on Parallel 3D Thinning Algorithms

In this section, 15 parallel 3D thinning algorithms are presented. These algorithms are composed of parallel reduction operations derived from our sufficient conditions for topology preservation (see Theorem 2).

Thinning algorithms preserve endpoints and some border points that provide relevant geometrical information with respect to the shape of the object. Here, we consider three types of endpoints.

Definition 1 There is no endpoint of type **TK**.

To standardize the notations, shrinking algorithms capable of producing topological kernels are considered as kernel-thinning algorithms, where no endpoint is preserved, hence we use endpoints of type **TK** (i.e., the empty set of the endpoints).

Definition 2 A black point p in picture $(\mathbb{Z}^3, 26, 6, B)$ is a curve-endpoint of type **CE** if $(N_{26}(p)\backslash\{p\}) \cap B$ contains exactly one point (i.e., p is 26-adjacent to exactly one further black point).

Endpoints of type **CE** have been considered by numerous existing 3D curve-thinning algorithms [26–28, 34–36, 38].

Definition 3 A black point p in picture $(\mathbb{Z}^3, 26, 6, B)$ is a surface-endpoint of type **SE** if there is no interior point in $N_{26}(p) \cap B$.

Note that the characterization of endpoints **SE** is applied in some existing surface-thinning algorithms [24, 26–28, 31, 33, 37].

In the rest of this section we present parallel 3D thinning algorithms composed of parallel reduction operations that satisfy Theorem 2.

6.3.1 Fully Parallel Algorithms

In fully parallel algorithms, the same parallel reduction operation is applied in each iteration step [1, 15, 16, 18, 19, 33, 45].

The scheme of the proposed fully parallel thinning algorithm 3D-FP-ε using endpoint of type ε is sketched in Algorithm 1 ($\varepsilon \in \{\mathbf{TK}, \mathbf{CE}, \mathbf{SE}\}$). Note that Palágyi and Németh reported three fully parallel 3D surface-thinning algorithms in [37], but they are based on sufficient conditions that differ from the conditions of Theorem 2.

Algorithm 1 Algorithm 3D-FP-ε

1: *Input*: picture $(\mathbb{Z}^3, 26, 6, X)$
2: *Output*: picture $(\mathbb{Z}^3, 26, 6, Y)$
3: $Y = X$
4: **repeat**
5: // *one iteration step*
6: $D = \{p \mid p$ is *3D-FP-ε-deletable* in $Y\}$
7: $Y = Y \setminus D$
8: **until** $D = \emptyset$

3D-FP-ε-deletable points are defined as follows:

Definition 4 A black point is *3D-FP-ε-deletable* if it is not an endpoint of type ε, and all conditions of Theorem 2 hold ($\varepsilon \in \{$**TK**, **CE**, **SE**$\}$).

We have the following theorem.

Theorem 3 *Algorithm* 3D-FP-ε ($\varepsilon \in \{$**TK**, **CE**, **SE**$\}$) *is topology preserving for* (26, 6) *pictures*.

Proof Deletable points of the proposed fully parallel algorithms (see Definition 4) are derived directly from conditions of Theorem 2. Hence, all of the three algorithms are topology preserving. □

Note that all objects contained in a unit lattice cube are formed of endpoints of type **SE**. Hence, Condition 3 of Theorem 2 can be ignored in algorithm 3D-FP-**SE**.

6.3.2 Subiteration-Based Algorithms

In subiteration-based (or frequently referred to as directional) thinning algorithms, an iteration step is decomposed into k successive parallel reduction operations according to k deletion directions [8]. If the current deletion direction is d, then a set of d-border points can be deleted by the parallel reduction operation assigned to it. Since there are six kinds of major directions in 3D cases, 6-subiteration algorithms were generally proposed [2, 7, 13, 20, 25, 34, 43, 46]. Moreover, 3-subiteration [30–32], 8-subiteration [35], and 12-subiteration [36] algorithms have also been developed for this task.

In what follows, we present three examples of parallel 3D 6-subiteration thinning algorithms. Algorithm 2 sketches the scheme of 3D 6-subiteration parallel thinning algorithm 3D-6-SI-ε that uses the endpoint of type ε ($\varepsilon \in \{$**TK**, **CE**, **SE**$\}$).

Algorithm 2 Algorithm 3D-6-SI-ε

Input: picture $(\mathbb{Z}^3, 26, 6, X)$
Output: picture $(\mathbb{Z}^3, 26, 6, Y)$
$Y = X$
repeat
  // one iteration step
  for each $i \in \{\mathbf{U}, \mathbf{D}, \mathbf{N}, \mathbf{E}, \mathbf{S}, \mathbf{W}\}$ **do**
    // subiteration for deleting some i-border points
    $D(i) = \{p \mid p$ is a 3D-6-SI-i-ε-deletable point in $Y\}$
    $Y = Y \setminus D(i)$
  end for
until $D(\mathbf{U}) \cup D(\mathbf{D}) \cup D(\mathbf{N}) \cup D(\mathbf{E}) \cup D(\mathbf{S}) \cup D(\mathbf{W}) = \emptyset$

The ordered list of deletion directions $\langle \mathbf{U}, \mathbf{D}, \mathbf{N}, \mathbf{E}, \mathbf{S}, \mathbf{W} \rangle$ [7, 34] is considered in the proposed algorithm 3D-6-SI-ε ($\varepsilon \in \{\mathbf{TK}, \mathbf{CE}, \mathbf{SE}\}$). Note that subiteration-based thinning algorithms are not invariant under the order of deletion directions (i.e., choosing different orders may yield various results).

In the first subiteration of our 6-subiteration thinning algorithms, the set of *3D-6-SI-***U**-ε*-deletable* points are deleted simultaneously, and the set of *3D-6-SI-***W**-ε*-deletable* points are deleted in the last (i.e., the 6th) subiteration. Now we lay down *3D-6-SI-***U**-ε*-deletable* points.

Definition 5 A black point p in picture $(\mathbb{Z}^3, 26, 6, X)$ is 3D-6-SI-**U**-ε-deletable if all of the following conditions hold:

1. Point p is a simple and **U**-border point, but it is not an endpoint of type ε in picture $(\mathbb{Z}^3, 26, 6, X)$.
2. Let $\mathscr{A}(p)$ be the family of the following 13 sets (see Fig. 6.2b):

$$\{e\}, \{s\}, \{se\}, \{sw\}, \{dn\}, \{de\}, \{ds\}, \{dw\},$$
$$\{e, s\}, \{e, se\}, \{s, se\}, \{s, sw\},$$
$$\{e, s, se\}.$$

   For any set A in the family $\mathscr{A}(p)$ composed of simple and **U**-border points, but not endpoints of type ε in picture $(\mathbb{Z}^3, 26, 6, X)$, point p remains simple in picture $(\mathbb{Z}^3, 26, 6, X \setminus A)$.
3. Let $\mathscr{B}(p)$ be the family of the following 42 objects in picture $(\mathbb{Z}^3, 26, 6, X)$ (see Fig. 6.2c):

$$\{a, h\}, \{b, g\}, \{c, f\}, \{d, e\},$$
$$\{a, h, b\}, \{a, h, c\}, \{a, h, f\}, \{a, h, g\}, \{b, g, a\}, \{b, g, d\}, \{b, g, e\}, \{b, g, h\},$$
$$\{c, f, a\}, \{c, f, d\}, \{c, f, e\}, \{c, f, h\}, \{d, e, b\}, \{d, e, c\}, \{d, e, f\}, \{d, e, g\},$$
$$\{b, c, h\}, \{d, g, f\}, \{a, d, f\}, \{b, e, h\}, \{b, c, e\}, \{a, f, g\}, \{a, d, g\}, \{c, e, h\},$$
$$\{a, h, b, c\}, \{a, h, b, g\}, \{a, h, c, f\}, \{a, h, f, g\}, \{b, g, a, d\}, \{b, g, d, e\},$$

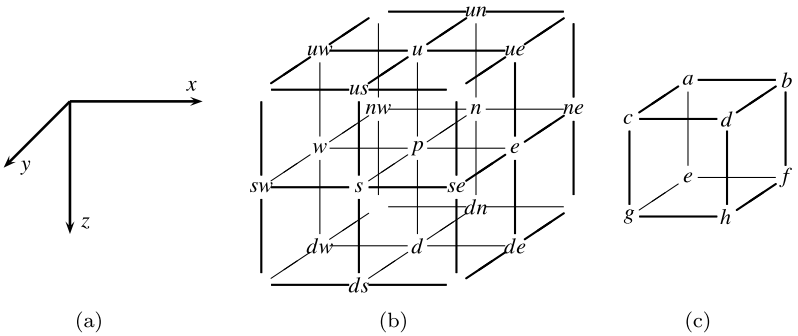


Fig. 6.2 The considered right-handed 3D coordinate system (**a**). Notation for the points in $N_{18}(p)$ (**b**). Notation for the points in a unit lattice cube (**c**)

$$\{b, c, e, h\}, \{b, g, e, h\}, \{c, f, a, d\}, \{c, f, d, e\}, \{c, f, e, h\}, \{d, e, b, c\},$$
$$\{d, e, f, g\}, \{a, d, f, g\}.$$

Point p is not the smallest element of any object in $\mathscr{B}(p)$.

Note that the deletable points at the remaining five subiterations can be derived from 3D-6-SI-**U**-ε-deletable points (assigned to the deletion direction **U**, see Definition 5) by reflexions and rotations.

Theorem 4 *Algorithm* 3D-6-SI-ε ($\varepsilon \in \{\mathbf{TK}, \mathbf{CE}, \mathbf{SE}\}$) *is topology preserving for* (26, 6) *pictures*.

Proof Without loss of generality, it is sufficient to prove that the first subiteration of algorithm 3D-6-SI-ε is topology preserving. To this end, we show that the parallel reduction operation $\mathscr{T}$ that deletes 3D-6-SI-**U**-ε-deletable points ($\varepsilon \in \{\mathbf{TK}, \mathbf{CE}, \mathbf{SE}\}$) satisfies all conditions of Theorem 2.

1. Operation $\mathscr{T}$ may delete simple points by Condition 1 of Definition 5. Hence Condition 1 of Theorem 2 is satisfied.
2. It is easy to see that the family $\mathscr{A}(p)$ (see Condition 2 of Definition 5 and Fig. 6.2a–b) contains all possible sets of simple **U**-border points that are considered by Condition 2 of Theorem 2. Therefore, this latter condition is satisfied.
3. It can be readily seen that the family of objects $\mathscr{B}(p)$ (see Condition 3 of Definition 5 and Fig. 6.2c) contains all possible objects of **U**-border points that are considered in Condition 3 of Theorem 2. Hence, this last condition is satisfied.

   Since objects contained in a unit lattice cube are composed of endpoints of type **SE**, Condition 3 of Definition 5 can be ignored in algorithm 3D-6-SI-**SE**. □

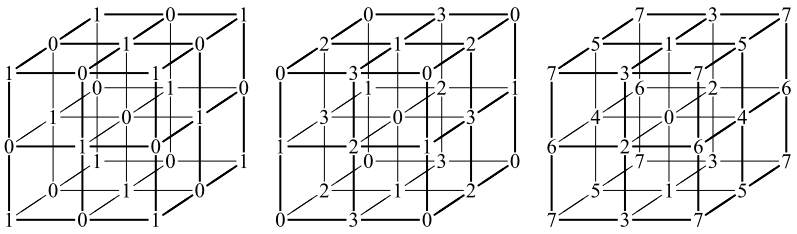


Fig. 6.3 The divisions of $\mathbb{Z}^3$ into 2 (**a**), 4 (**b**), and 8 (**c**) subfields. If partitioning into k subfields is considered, then points marked "i" are in the subfield $SF_k(i)$ ($k = 2, 4, 8$; $i = 0, 1, \ldots, k-1$)

6.3.3 Subfield-Based Algorithms

The third type of parallel thinning algorithms applies subfield-based technique [8]. In existing subfield-based parallel 3D thinning algorithms, the digital space $\mathbb{Z}^3$ is partitioned into two [21, 22, 26], four [23, 27], and eight [3, 27] subfields which are alternatively activated. At a given iteration step of a k-subfield algorithm, k successive parallel reduction operations associated to the k subfields are performed. In each of them, some border points in the active subfield can be designated for deletion.

Let us denote $SF_k(i)$ the i-th subfield if $\mathbb{Z}^3$ is partitioned into k subfields ($k = 2, 4, 8$; $i = 0, \ldots, k-1$). $SF_k(i)$ is defined formally as follows:

$$SF_2(i) = \big\{(p_x, p_y, p_z) \mid (p_x + p_y + p_z \bmod 2) = i\big\},$$

$$SF_4(i) = \big\{(p_x, p_y, p_z) \mid (p_x + 1 \bmod 2) \cdot \big[2 \cdot (p_y \bmod 2) + (p_z \bmod 2)\big] + (p_x \bmod 2) \cdot \big[2 \cdot (p_y + 1 \bmod 2) + (p_z + 1 \bmod 2)\big] = i\big\},$$

$$SF_8(i) = \big\{(p_x, p_y, p_z) \mid 4 \cdot (p_x \bmod 2) + 2 \cdot (p_y \bmod 2) + (p_z \bmod 2) = i\big\}$$

The considered divisions are illustrated in Fig. 6.3.

Proposition 1 *For the* 2*-subfield case* (*see Fig.* 6.3a), *two points* p *and* $q \in N_{26}(p)$ *are in the same subfield, if* $q \in N_{18}(p) \backslash N_6(p)$.

Proposition 2 *For the* 4*-subfield case* (*see Fig.* 6.3b), *two points* p *and* $q \in N_{26}(p)$ *are in the same subfield, if* $q \in N_{26}(p) \backslash N_{18}(p)$.

Proposition 3 *For the* 8*-subfield case* (*see Fig.* 6.3c), *two points* p *and* $q \in N_{26}(p)$ *are* not *in the same subfield.*

In order to reduce the noise sensitivity and the number of skeletal points (without overshrinking the objects), Németh, Kardos, and Palágyi introduced a new subfield-based thinning scheme [26]. It takes the endpoints into consideration at the beginning of iteration steps, instead of preserving them in each parallel reduction operation as it is accustomed in the conventional subfield-based thinning scheme.

Next, we present nine parallel 3D subfield-based thinning algorithms. The scheme of the subfield-based parallel thinning algorithm 3D-k-SF-ε with iteration-level endpoint checking using endpoint of type ε is sketched in Algorithm 3 (with $k = 2, 4, 8$; $\varepsilon \in \{\mathbf{TK}, \mathbf{CE}, \mathbf{SE}\}$).

Algorithm 3 Algorithm 3D-k-SF-ε

Input: picture $(\mathbb{Z}^3, 26, 6, X)$
Output: picture $(\mathbb{Z}^3, 26, 6, Y)$
$Y = X$
repeat
  // one iteration step
  $E = \{p \mid p$ is a border point, but not an endpoint of type ε in $Y\}$
  for $i = 0$ **to** $k - 1$ **do**
    // subfield $SF_k(i)$ is activated
    $D(i) = \{q \mid q$ is a 3D-SF-k-deletable point in $E \cap SF_k(i)\}$
    $Y = Y \setminus D(i)$
  end for
until $D(0) \cup D(1) \cup \ldots \cup D(k-1) = \emptyset$

The 3D-SF-k-deletable points are defined as follows ($k = 2, 4, 8$):

Definition 6 A black point p is 3D-SF-k-deletable ($k = 2, 4, 8$) in picture $(\mathbb{Z}^3, 26, 6, X)$ if all of the following conditions hold:

1. Point p is simple in $(\mathbb{Z}^3, 26, 6, X)$.
2. If $k = 2$, then point p is simple in picture $(\mathbb{Z}^3, 26, 6, X \setminus \{q\})$ for any simple point q such that $q \in N_{18}(p) \setminus N_6(p)$ and $p \prec q$.
3. - If $k = 2$, then point p is not the smallest element of the ten objects depicted in Fig. 6.4.
   - If $k = 4$, then point p is not the smallest element of the four objects depicted in Fig. 6.5.

Theorem 5 *Algorithm* 3D-k-SF-ε ($k = 2, 4, 8$; $\varepsilon \in \{\mathbf{TK}, \mathbf{CE}, \mathbf{SE}\}$) *is topology preserving for* $(26, 6)$ *pictures*.

Proof To prove it, we show that the parallel reduction operation $\mathscr{T}$ that deletes 3D-SF-k-deletable points satisfies all conditions of Theorem 2.

1. Operation $\mathscr{T}$ may delete simple points by Condition 1 of Definition 6. Hence Condition 1 of Theorem 2 is satisfied.
2. - Let $k = 2$ and let $p \in SF_2(i)$ be any black point in picture $(\mathbb{Z}^3, 26, 6, X)$ that is deleted by $\mathscr{T}$ ($i = 0, 1$).

     Let $Q \subseteq X \cap SF_2(i)$ be any set of black points in $(\mathbb{Z}^3, 26, 6, X)$ such that $p \in Q$, Q is contained in a unit lattice square, and each point in $Q \setminus \{p\}$ is simple in picture $(\mathbb{Z}^3, 26, 6, X)$.

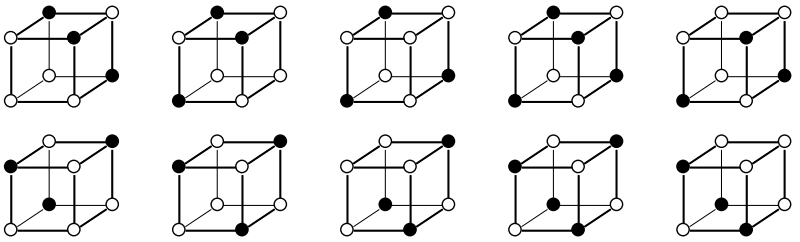


Fig. 6.4 The ten objects that are taken into consideration by 2-subfield algorithms. Notations: each point marked by "●" is a black point; each point marked by "○" is a white point. (Note that each of these objects is contained in a unit *lattice cube*)

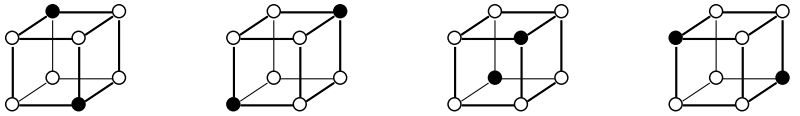


Fig. 6.5 The four objects considered by 4-subfield algorithms. Notations: each point marked "●" is a black point; each point marked "○" is a white point. (Note that each of these objects is contained in a unit *lattice cube*)

Then $Q = \emptyset$ or $Q = \{q\}$ by Proposition 1, and such kind of sets are considered by Condition 2 of Definition 6. Hence Condition 2 of Theorem 2 is satisfied.

- Let $k = 4$ and let $p \in SF_4(i)$ be any black point in picture $(\mathbb{Z}^3, 26, 6, X)$ that is deleted by $\mathscr{T}$ $(i = 0, 1, 2, 3)$.

  Let $Q \subseteq X \cap SF_4(i)$ be any set of black points in $(\mathbb{Z}^3, 26, 6, X)$ such that $p \in Q$, Q is contained in a unit lattice square, and each point in $Q \backslash \{p\}$ is simple in picture $(\mathbb{Z}^3, 26, 6, X)$.

  Then $Q = \emptyset$ by Proposition 2. Hence Condition 2 of Theorem 2 is satisfied.
- Let $k = 8$ and let $p \in SF_8(i)$ be any black point in picture $(\mathbb{Z}^3, 26, 6, X)$ that is deleted by $\mathscr{T}$ $(i = 0, 1, \ldots, 7)$.

  Let $Q \subseteq X \cap SF_8(i)$ be any set of black points in $(\mathbb{Z}^3, 26, 6, X)$ such that $p \in Q$, Q is contained in a unit lattice square, and each point in $Q \backslash \{p\}$ is simple in picture $(\mathbb{Z}^3, 26, 6, X)$.

  Then $Q = \emptyset$ by Proposition 3. Hence Condition 2 of Theorem 2 is satisfied.

3. - Let $k = 2$ and let $C \subseteq X \cap SF_2(i)$ be any object in picture $(\mathbb{Z}^3, 26, 6, X)$ that is contained in a unit lattice cube $(i = 0, 1)$.

     It can be readily seen by Proposition 1 that all the ten possible cases for such objects are depicted in Fig. 6.4, and these objects cannot be deleted completely by Condition 3 of Definition 6.

     Hence Condition 3 of Theorem 2 is satisfied.
   - Let $k = 4$ and let $C \subseteq X \cap SF_4(i)$ be any object in picture $(\mathbb{Z}^3, 26, 6, X)$ that is contained in a unit lattice cube $(i = 0, 1, 2, 3)$.

It can be readily seen by Proposition 2 that all the four possible cases for such objects are depicted in Fig. 6.5, and these objects cannot be deleted completely by Condition 3 of Definition 6.

Hence Condition 3 of Theorem 2 is satisfied.

- Let $k = 8$ and let $C \subseteq X \cap SF_8(i)$ be any object in picture $(\mathbb{Z}^3, 26, 6, X)$ that is contained in a unit lattice cube ($i = 0, 1, \ldots, 7$).

  It is easy to see that there is no such an object by Proposition 3.

  Hence Condition 3 of Theorem 2 is satisfied. □

Since objects contained in a unit lattice cube are composed of endpoints of type **SE**, Condition 3 of Definition 6 can be ignored in algorithm 3D-k-SF-**SE** ($k = 2, 4, 8$).

6.4 Implementation

One may think that the proposed algorithms are time consuming and it is rather difficult to implement them. That is why we outline a method for implementing any 3D fully parallel thinning algorithm on a conventional sequential computer. This framework is fairly general, as similar schemes can be used for the other classes of parallel algorithms and some sequential 3D thinning algorithms [33, 37, 38].

The proposed method uses a pre-calculated look-up-table to encode simple points. In addition, two lists are used to speed up the process: one for storing the border points in the current picture (since thinning can only delete border points, thus the repeated scans/traverses of the entire array storing the picture are avoided); the other list is to collect all deletable points in the current phase of the process. At each iteration, the deletable points are found and deleted, and the list of border points is updated accordingly. The algorithm terminates when no further update is required.

For simplicity, the pseudocode of the proposed 3D fully parallel thinning algorithms is given (see Algorithm 4). The subiteration-based and the subfield-based variants can be implemented in similar ways.

The two input parameters of the procedure are array A which stores the input picture to be thinned and the type of the considered endpoints ε. In input array A, the value "1" corresponds to black points and the value "0" denotes white ones. According to the proposed scheme, the input and the output pictures can be stored in the same array, hence array A will contain the resultant structure.

First, the input picture is scanned and all the border points are inserted into the list *border_list*. We should mention here that it is the only time consuming scanning. Since only a small part of points in a usual picture belong to the objects, the thinning procedure is much faster if we just deal with the set of border points in the actual picture. This subset of object points is stored in *border_list* (i.e., a dynamic data structure). The *border_list* is then updated: if a border point is deleted, then all interior points that are 6-adjacent to it become border points. These brand new

Algorithm 4 Fully parallel thinning algorithm

```
1: Input: array A and endpoint characterization ε
2: Output: array A
3: // collect border points
4: border_list = ⟨empty list⟩
5: for each p = (x, y, z) in A do
6:    if p is border point then
7:       border_list = border_list + ⟨p⟩
8:       A[x, y, z] = 2
9:    end if
10: end for
11: // thinning
12: repeat
13:    deleted = 0
14:    deletable_list = ⟨empty list⟩
15:    // checking Condition 1 of Theorem 2
16:    for each point p = (x, y, z) in border_list do
17:       if p is a simple point and not an endpoint of type ε then
18:          deletable_list = deletable_list + ⟨p⟩
19:          A[x, y, z] = 3
20:       else
21:          A[x, y, z] = 2
22:       end if
23:    end for
24:    // checking Condition 2 of Theorem 2
25:    for each point p in deletable_list do
26:       if deletion p does not satisfy Condition 2 of Theorem 2 then
27:          deletable_list = deletable_list − ⟨p⟩
28:       end if
29:    end for
30:    // checking Condition 3 of Theorem 2
31:    for each point p in deletable_list do
32:       if deletion p does not satisfy Condition 3 of Theorem 2 then
33:          deletable_list = deletable_list − ⟨p⟩
34:       end if
35:    end for
36:    // deletion
37:    for each point p = (x, y, z) in deletable_list do
38:       A[x, y, z] = 0
39:       border_list = border_list − ⟨p⟩
40:       deleted = deleted+1
41:       // update border_list
42:       for each point q = (x′, y′, z′) that is 6-adjacent to p do
43:          if A[x′, y′, z′] = 1 then
44:             A[x′, y′, z′] = 2
45:             border_list = border_list + ⟨q⟩
46:          end if
47:       end for
48:    end for
49: until deleted = 0
```

border points of the actual picture are added to the *border_list*. In order to avoid storing more than one copy of a border point in *border_list*, array A represents a four-color picture during the thinning process: the value "0" corresponds to the white points, the value "1" corresponds to (black) interior points, the value "2" is assigned to all (black) border points in the actual picture (added to *border_list*), and the value "3" corresponds to points that are added to the *deletable_list* (i.e., a sublist of *border_list*).

The kernel of the *repeat* cycle corresponds to one iteration step of the thinning process. The number of deleted points is stored in the variable called *deleted*. The thinning process terminates when no more points can be deleted (i.e., no further changes occur). After thinning, all points having a nonzero value belong to the produced skeleton-like shape feature.

Simple points in $(26, 6)$ pictures can be locally characterized; the simplicity of a point p can be decided by examining the set $N_{26}(p)$ [12]. There are 2^{26} possible configurations in the $3 \times 3 \times 3$ neighborhood if the central point is not considered. Hence we can assign an index (i.e., a non-negative integer code) for each possible configuration and address a pre-calculated (unit time access) look-up-table having 2^{26} entries of 1 bit in size, therefore, it requires only 8 megabytes storage space in memory.

By adapting this efficient implementation method, our algorithms can be well applied in practice: they are capable of extracting skeleton-like shape features from large 3D pictures containing 1 000 000 object points within one second on a standard PC.

6.5 Results

The proposed 15 algorithms were tested on objects of different shapes. Here we present some of them (see Figs. 6.6–6.12). The pairs of numbers in parentheses are the counts of object points in the produced skeleton-like structure and the parallel speed (i.e., the number of the performed parallel reduction operations [8]).

6.6 Possible Future Works and Open Problems

In this section, we will outline some possible future works and open problems concerning parallel 3D thinning.

- Conventional thinning algorithms preserve endpoints to provide important geometric information relative to the object to be represented. Bertrand and Couprie proposed an alternative strategy [4]. They developed a sequential thinning scheme based on a generalization of curve/surface interior points that are called *isthmus*es. Isthmuses are dynamically detected and accumulated in a constraint set of non-simple points.

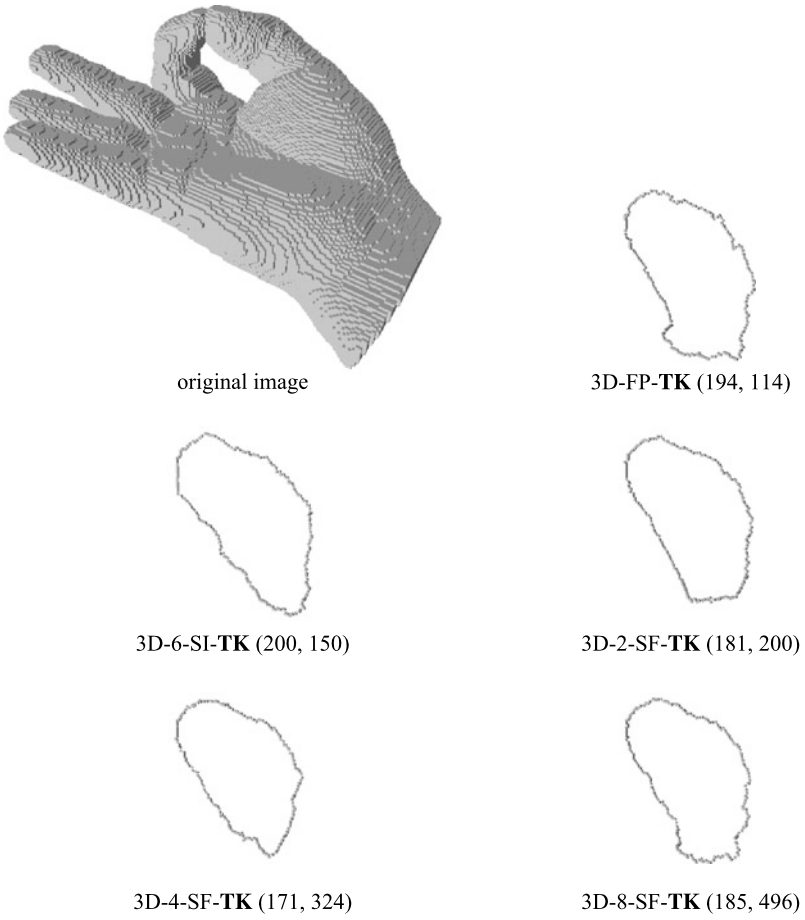


Fig. 6.6 A 191 × 96 × 114 image of a *hand* and its topological kernels produced by the five proposed parallel 3D kernel-thinning algorithms. The original image contains 455 295 black points. Since the original object contains a hole, there are holes in its topological kernels, too

The very first parallel 3D isthmus-based curve-thinning algorithm was designed by Raynal and Couprie [39]. Each iteration step of their 6-subiteration algorithm consists of two phases:

1. Updating the constraint set, by adding points detected as isthmuses;
2. Removing "deletable" points which are not in the constraint set.

Raynal and Couprie gave these "deletable" points by $3 \times 3 \times 3$ matching templates, and proved that simultaneous deletion of "deletable" points is a topology preserving reduction operation. Hence their algorithm is topology preserving.

In a forthcoming work, we are going to combine our sufficient conditions for topology preservation (see Theorem 2) with various parallel thinning strategies (i.e., fully parallel, subiteration-based, and subfield-based) and some character-

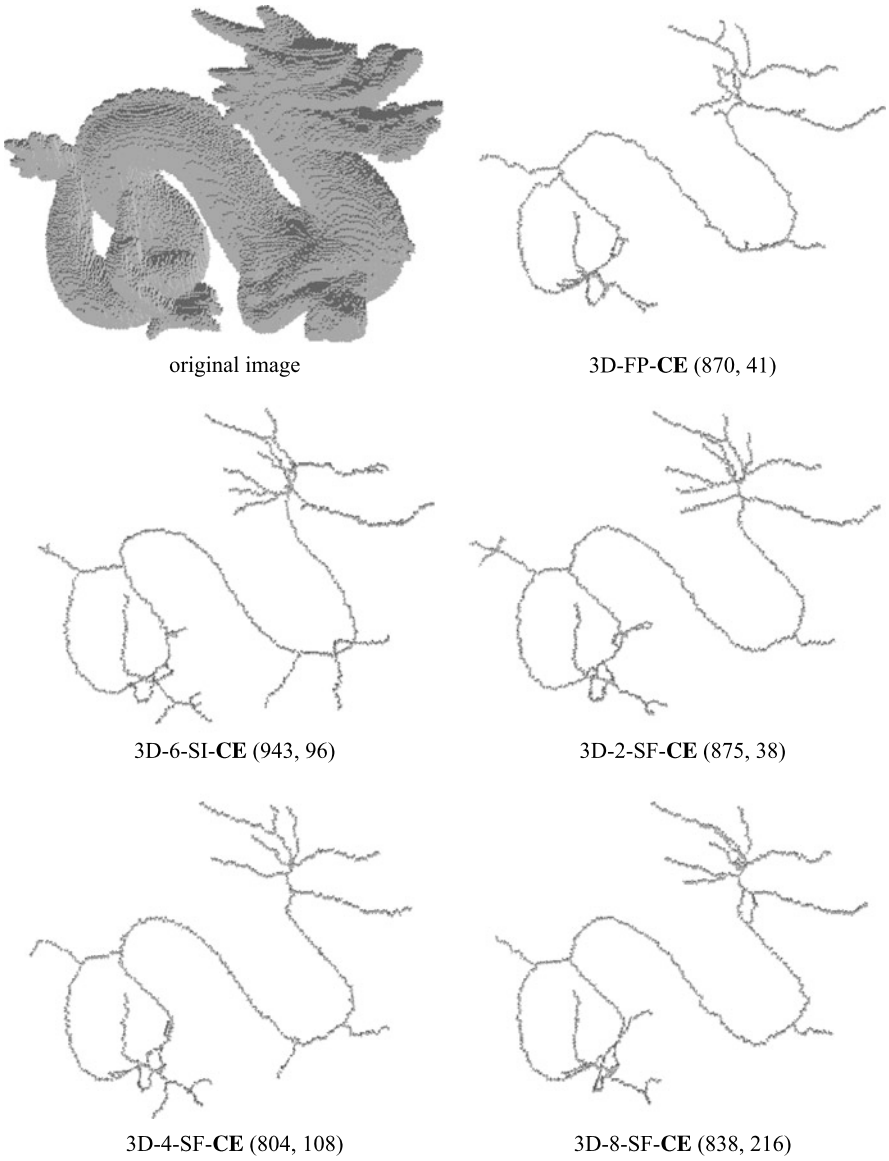


Fig. 6.7 A $135 \times 86 \times 191$ image of a *dragon* and its centerlines produced by the five proposed parallel 3D curve-thinning algorithms. The original image contains 423 059 black points

izations of isthmuses to generate new parallel 3D curve-thinning and surface-thinning algorithms.

- The 3D parallel thinning algorithms presented in this chapter are based on Theorem 2 (i.e., some sufficient conditions for topology preservation). Conditions 2 and 3 of Theorem 2 are "asymmetric", since points that are the smallest elements

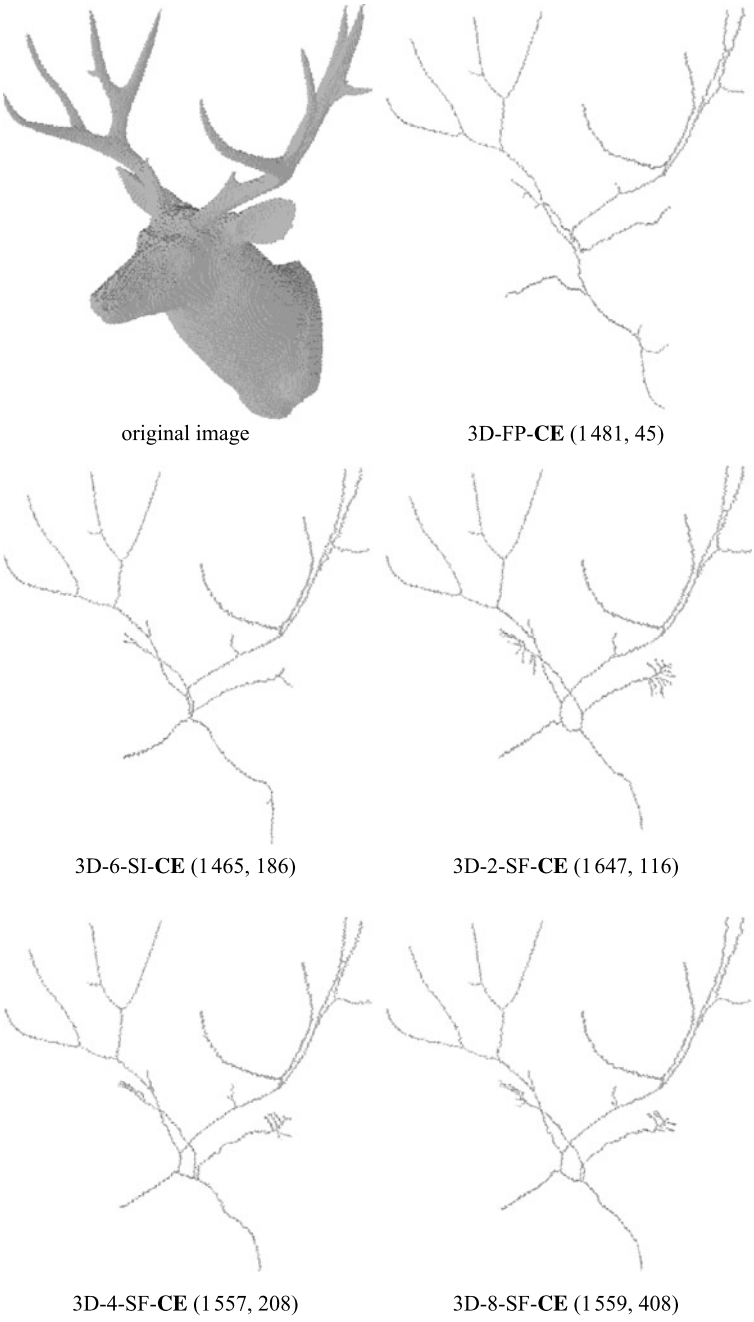


Fig. 6.8 A $380 \times 287 \times 271$ image of a *deer head* and its centerlines produced by the five proposed parallel 3D curve-thinning algorithms. The original image contains 1 658 641 black points

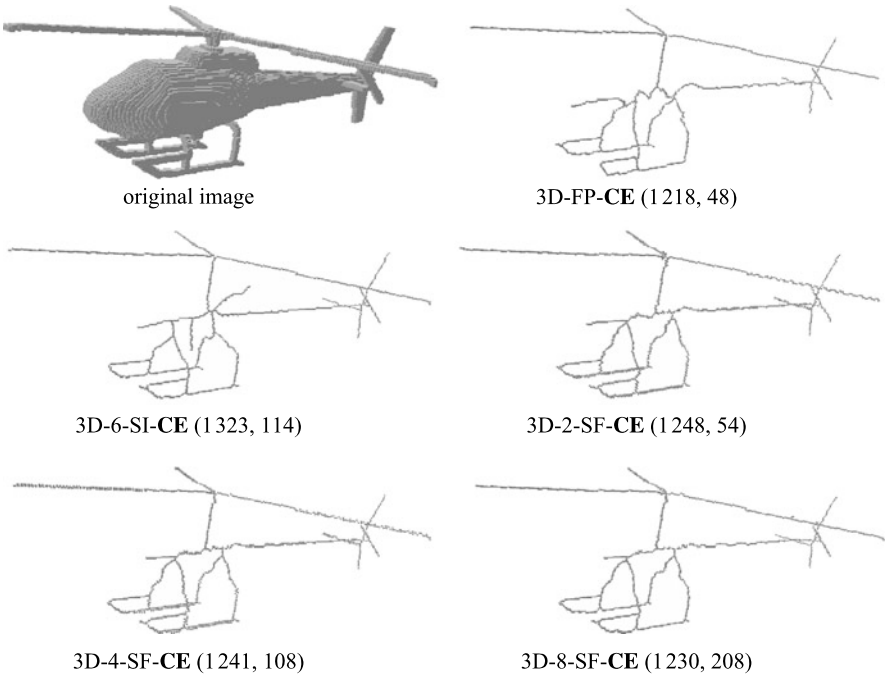


Fig. 6.9 A $103 \times 381 \times 255$ image of a *helicopter* and its centerlines produced by the five proposed parallel 3D curve-thinning algorithms. The original image contains 273 743 black points

of some sets may not be deleted. It is easy to see that the following theorem provides "symmetric" conditions for topology preservation.

Theorem 6 *The parallel reduction operation $\mathcal{O}$ is topology preserving for* $(26, 6)$ *pictures if all the following conditions hold for any black point p in any picture* $(\mathbb{Z}^3, 26, 6, B)$ *such that p is deleted by $\mathcal{O}$.*

1. *Point p is simple in* $(\mathbb{Z}^3, 26, 6, B)$.
2. *Let $Q \subseteq B$ be any set of simple points in* $(\mathbb{Z}^3, 26, 6, B)$ *such that $p \in Q$, and Q is contained in a unit lattice square.*
   Then point p is simple in picture $(\mathbb{Z}^3, 26, 6, B \backslash (Q \backslash \{p\}))$.
3. *Point p is not an element of any object $C \subseteq B$ in* $(\mathbb{Z}^3, 26, 6, B)$ *such that C is contained in a unit lattice cube.*

In a future work, we plan to combine alternative sufficient conditions for topology preservation with parallel thinning strategies to generate further classes of 3D parallel thinning algorithms.

- Unfortunately, skeletonization methods (including thinning) are rather sensitive to coarse object boundaries, hence the produced skeletons generally contain some

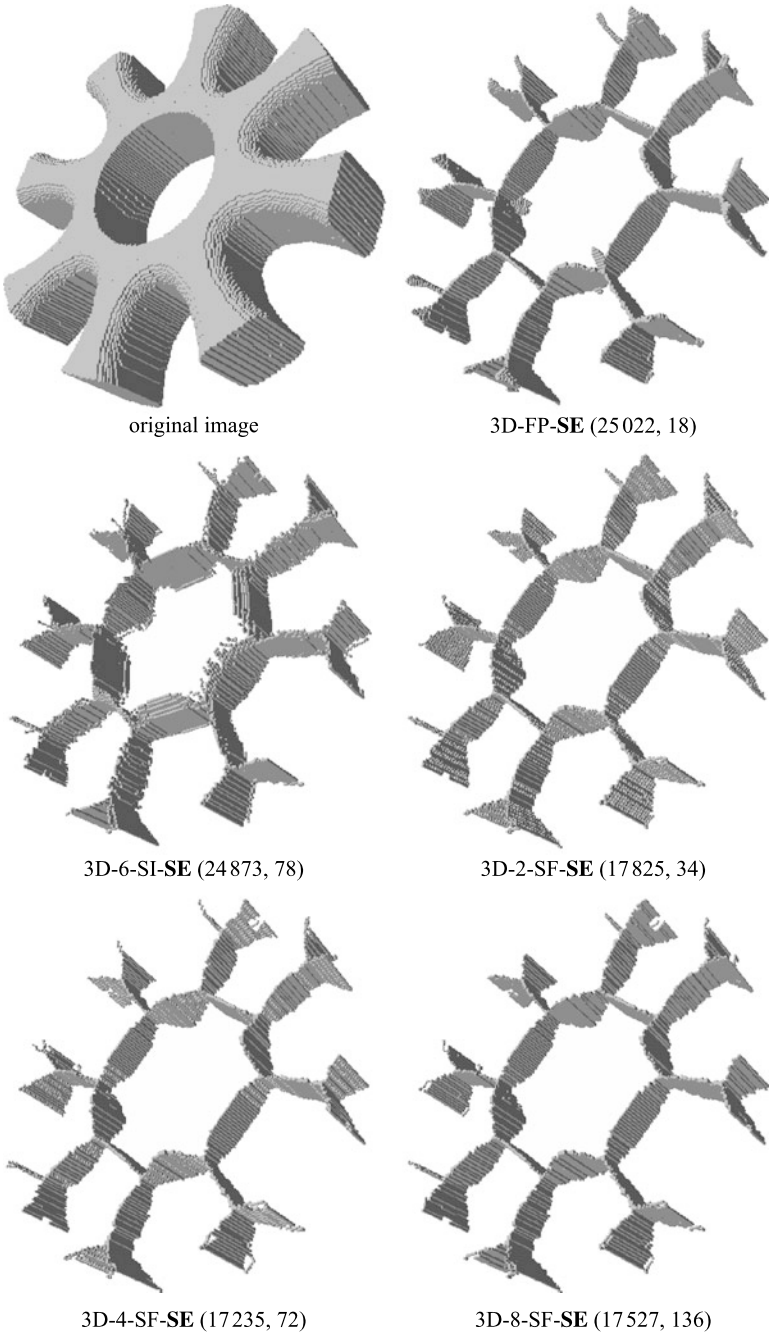


Fig. 6.10 A $45 \times 191 \times 191$ image of a *gear* and its medial surfaces produced by the five proposed parallel 3D surface-thinning algorithms. The original image contains 596 360 black points

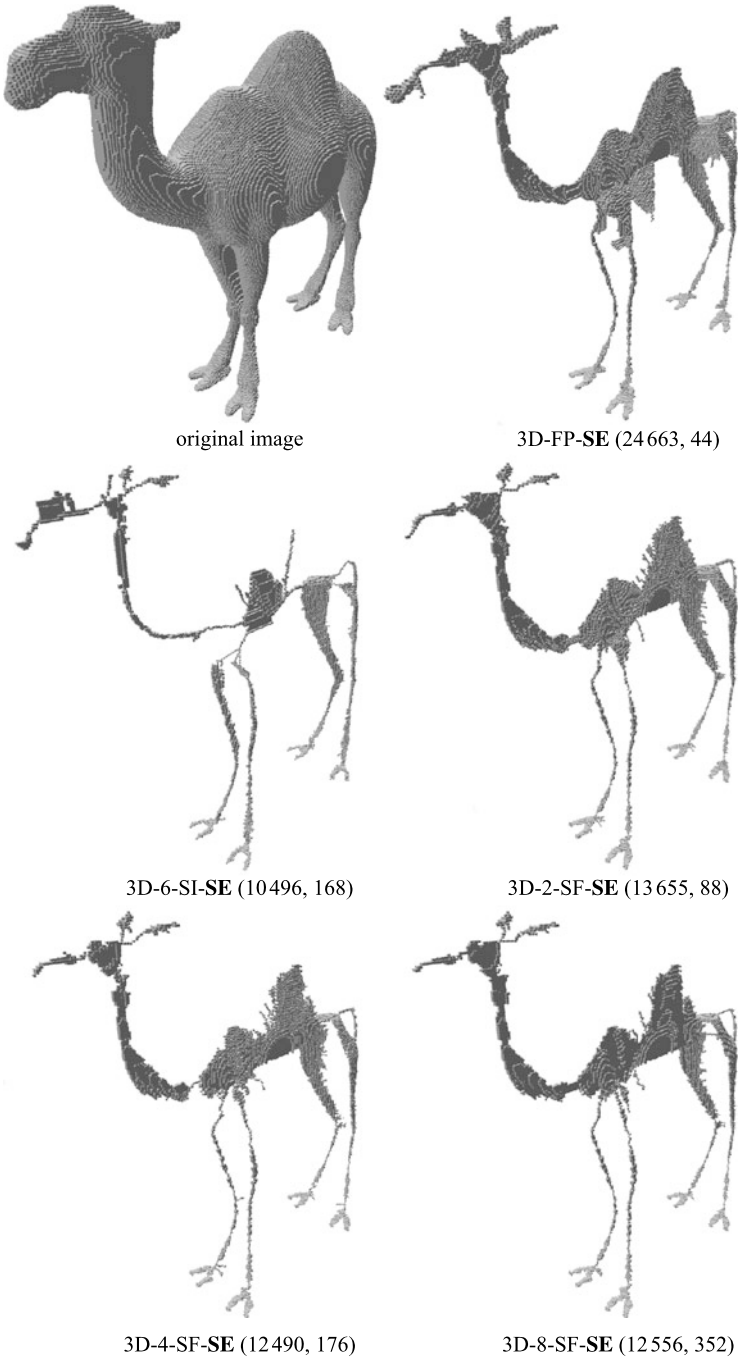
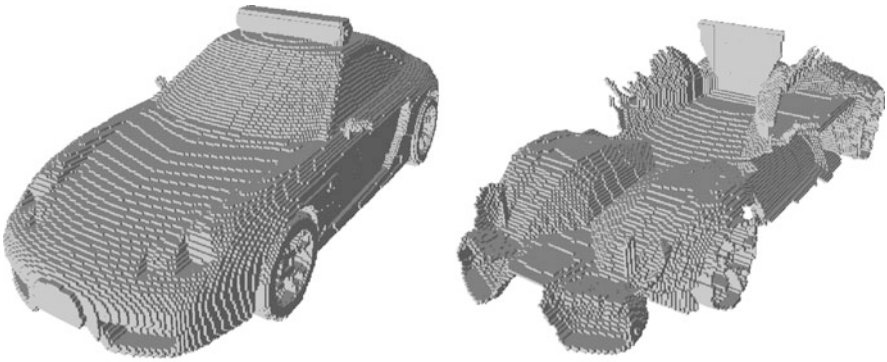
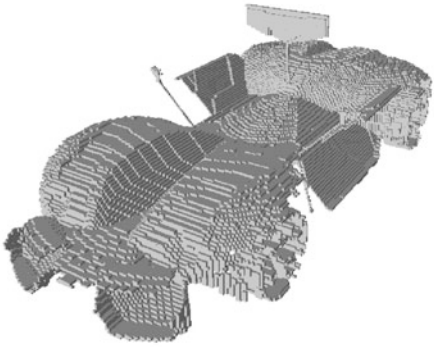


Fig. 6.11 A $285 \times 285 \times 88$ image of a *camel* and its medial surfaces produced by the five proposed parallel 3D surface-thinning algorithms. The original image contains 1 088 458 black points

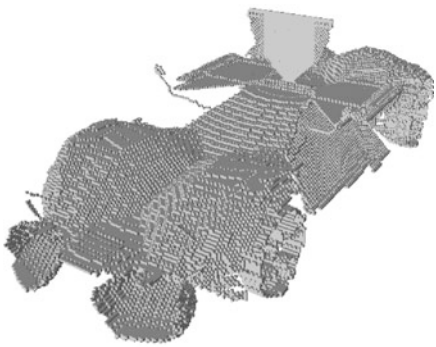


original image

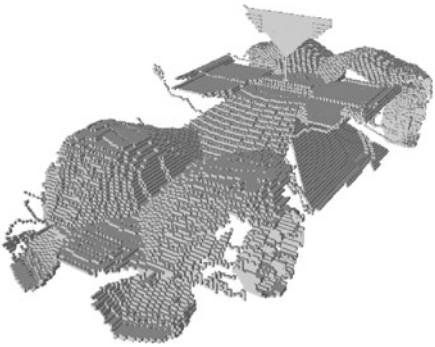
3D-FP-**SE** (76 359, 41)



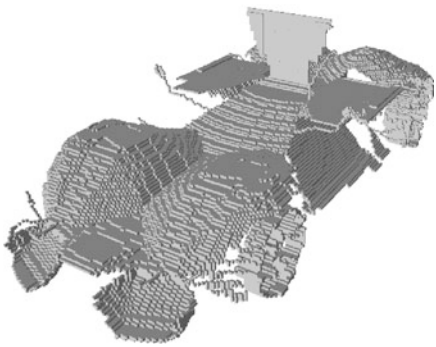
3D-6-SI-**SE** (60 000, 192)



3D-2-SF-**SE** (48 335, 82)



3D-4-SF-**SE** (44 508, 160)



3D-8-SF-**SE** (47 088, 320)

Fig. 6.12 A $122 \times 93 \times 284$ image of a *car* and its medial surfaces produced by the five proposed parallel 3D surface-thinning algorithms. The original image contains 1 321 764 black points

false segments. In order to overcome this problem, unwanted skeletal parts are usually removed by a *pruning* process as a post-processing step [40]. In [29], we presented a new thinning scheme for reducing the noise sensitivity of 3D thinning

algorithms. It uses iteration-by-iteration smoothing which removes some border points being considered as extremities.

We are going to design new topology preserving parallel contour smoothing operations, and combine our 3D parallel thinning algorithms (based on sufficient conditions for topology preservation) with iteration-by-iteration smoothing.

- It is easy to see that subiteration-based and subfield-based parallel thinning schemes are not invariant under the order of deletion directions and subfield activations, respectively. It means that choosing different orders of directions may yield various results in subiteration-based algorithms, and varieties of skeleton-like shape features can be produced by a subfield-based algorithm with diverse orders of the active subfields.

  Neither *order-independent* subiteration-based nor subfield-based parallel thinning algorithms have been proposed. We are going to deal with this unsolved problem (i.e., we plan to construct subiteration-based and subfield-based algorithms that produce the same result for any order of deletion directions and subfield activation).

6.7 Concluding Remarks

Fast and reliable extraction of skeleton-like shape features (i.e., medial surface, centerline, and topological kernel) is extremely important in numerous applications for large 3D shapes. In this chapter we presented a variety of parallel 3D thinning algorithms and their efficient implementation. They are based on some sufficient conditions for topology preserving parallel reduction operations, hence their topological correctness is guaranteed. The algorithms are based on different characterizations of endpoints. Additional types of endpoints coupled with sufficient conditions for topology preservation yield newer thinning algorithms.

Acknowledgements This research was supported by the TÁMOP-4.2.2/08/1/2008-0008 program of the Hungarian National Development Agency, the European Union and the European Regional Development Fund under the grant agreement TÁMOP-4.2.1/B-09/1/KONV-2010-0005, and the grant CNK80370 of the National Office for Research and Technology (NKTH) & the Hungarian Scientific Research Fund (OTKA).

References

1. Arcelli, C., Sanniti di Baja, G., Serino, L.: New removal operators for surface skeletonization. In: Proc. 13th Int. Conf. Discrete Geometry for Computer Imagery, DGCI 2006. Lecture Notes in Computer Science, vol. 4245, pp. 555–566. Springer, Heidelberg (2006)
2. Bertrand, G.: A parallel thinning algorithm for medial surfaces. Pattern Recognit. Lett. **16**, 979–986 (1995)
3. Bertrand, G., Aktouf, Z.: A 3D thinning algorithm using subfields. In: SPIE Proc. of Conf. on Vision Geometry, pp. 113–124 (1994)

4. Bertrand, G., Couprie, M.: Transformations topologiques discrètes. In: Coeurjolly, D., Montanvert, A., Chassery, J. (eds.) Géométrie discrète et images numériques, pp. 187–209. Hermès Science Publications-Lavoisier, Paris (2007)
5. Blum, H.: A transformation for extracting new descriptors of shape. In: Wathen-Dunn, W. (ed.) Models for the Perception of Speech and Visual Form, pp. 362–380. MIT Press, Cambridge (1967)
6. Gomberg, B.R., Saha, P.K., Song, H.K., Hwang, S.N., Wehrli, F.W.: Topological analysis of trabecular bone MR images. IEEE Trans. Med. Imaging **19**, 166–174 (2000)
7. Gong, W.X., Bertrand, G.: A simple parallel 3D thinning algorithm. In: Proc. 10th IEEE Internat. Conf. on Pattern Recognition, ICPR'90, pp. 188–190 (1990)
8. Hall, R.W.: Parallel connectivity-preserving thinning algorithms. In: Kong, T.Y., Rosenfeld, A. (eds.) Topological Algorithms for Digital Image Processing, pp. 145–179. Elsevier, Amsterdam (1996)
9. Hall, R.W., Kong, T.Y., Rosenfeld, A.: Shrinking binary images. In: Kong, T.Y., Rosenfeld, A. (eds.) Topological Algorithms for Digital Image Processing, pp. 31–98. Elsevier, Amsterdam (1996)
10. Itoh, T., Yamaguchi, Y., Koyamada, K.: Fast isosurface generation using the volume thinning algorithm. IEEE Trans. Vis. Comput. Graph. **7**, 32–46 (2001)
11. Kong, T.Y.: On topology preservation in 2-d and 3-d thinning. Int. J. Pattern Recognit. Artif. Intell. **9**, 813–844 (1995)
12. Kong, T.Y., Rosenfeld, A.: Digital topology: introduction and survey. Comput. Vis. Graph. Image Process. **48**, 357–393 (1989)
13. Lee, T., Kashyap, R.L., Chu, C.: Building skeleton models via 3-D medial surface/axis thinning algorithms. CVGIP, Graph. Models Image Process. **56**, 462–478 (1994)
14. Lohou, C.: Detection of the non-topology preservation of Ma's 3D surface-thinning algorithm, by the use of P-simple points. Pattern Recognit. Lett. **29**, 822–827 (2008)
15. Lohou, C., Dehos, J.: An automatic correction of Ma's thinning algorithm based on P-simple points. J. Math. Imaging Vis. **36**, 54–62 (2010)
16. Lohou, C., Dehos, J.: Automatic correction of Ma and Sonka's thinning algorithm using P-simple points. IEEE Trans. Pattern Anal. Mach. Intell. **32**, 1148–1152 (2010)
17. Ma, C.M.: On topology preservation in 3D thinning. CVGIP, Image Underst. **59**, 328–339 (1994)
18. Ma, C.M.: A 3D fully parallel thinning algorithm for generating medial faces. Pattern Recognit. Lett. **16**, 83–87 (1995)
19. Ma, C.M., Sonka, M.: A fully parallel 3D thinning algorithm and its applications. Comput. Vis. Image Underst. **64**, 420–433 (1996)
20. Ma, C.M., Wan, S.-Y.: Parallel thinning algorithms on 3D (18, 6) binary images. Comput. Vis. Image Underst. **80**, 364–378 (2000)
21. Ma, C.M., Wan, S.Y.: A medial-surface oriented 3-d two-subfield thinning algorithm. Pattern Recognit. Lett. **22**, 1439–1446 (2001)
22. Ma, C.M., Wan, S.Y., Chang, H.K.: Extracting medial curves on 3D images. Pattern Recognit. Lett. **23**, 895–904 (2002)
23. Ma, C.M., Wan, S.Y., Lee, J.D.: Three-dimensional topology preserving reduction on the 4-subfields. IEEE Trans. Pattern Anal. Mach. Intell. **24**, 1594–1605 (2002)
24. Manzanera, A., Bernard, T.M., Pretêux, F., Longuet, B.: Medial faces from a concise 3D thinning algorithm. In: Proc. 7th IEEE Int. Conf. Computer Vision, ICCV'99, pp. 337–343 (1999)
25. Mukherjee, J., Das, P.P., Chatterjee, B.N.: On connectivity issues of ESPTA. Pattern Recognit. Lett. **11**, 643–648 (1990)
26. Németh, G., Kardos, P., Palágyi, K.: Topology preserving 2-subfield 3D thinning algorithms. In: Proc. 7th IASTED Int. Conf. Signal Processing, Pattern Recognition and Applications, pp. 310–316 (2010)
27. Németh, G., Kardos, P., Palágyi, K.: Topology preserving 3D thinning algorithms using four and eight subfields. In: Proc. International Conference on Image Analysis and Recognition,

ICIAR 2010. Lecture Notes in Computer Science, vol. 6111, pp. 316–325. Springer, Heidelberg (2010)
28. Németh, G., Kardos, P., Palágyi, K.: A family of topology-preserving 3D parallel 6-subiteration thinning algorithms. In: Proc. 14th International Workshop on Combinatorial Image Analysis, IWCIA'2011, Madrid, Spain. Lecture Notes in Computer Science, vol. 6636, pp. 17–30. Springer, Heidelberg (2011)
29. Németh, G., Kardos, P., Palágyi, K.: Thinning combined with iteration-by-iteration smoothing for 3D binary images. Graph. Models **73**, 335–345 (2011)
30. Palágyi, K.: A 3-subiteration 3D thinning algorithm for extracting medial surfaces. Pattern Recognit. Lett. **23**, 663–675 (2002)
31. Palágyi, K.: A 3-subiteration surface-thinning algorithm. In: Proc. 12th Int. Conf. Computer Analysis of Images and Patterns, CAIP 2007, Vienna, Austria. Lecture Notes in Computer Science, vol. 4673, pp. 628–635. Springer, Heidelberg (2007)
32. Palágyi, K.: A subiteration-based surface-thinning algorithm with a period of three. In: Hamprecht, F., Jähne, B., Schnörr, C. (eds.) Lecture Notes in Computer Science, vol. 4713, pp. 294–303. Springer, Heidelberg (2007)
33. Palágyi, K.: A 3D fully parallel surface-thinning algorithm. Theor. Comput. Sci. **406**, 119–135 (2008)
34. Palágyi, K., Kuba, A.: A 3D 6-subiteration thinning algorithm for extracting medial lines. Pattern Recognit. Lett. **19**, 613–627 (1998)
35. Palágyi, K., Kuba, A.: Directional 3D thinning using 8 subiterations. In: Proc. 8th Int. Conf. on Discrete Geometry for Computer Imagery, DGCI'99, Marne-la-Valle, France. Lecture Notes in Computer Science, vol. 1568, pp. 325–336. Springer, Heidelberg (1999)
36. Palágyi, K., Kuba, A.: A parallel 3D 12-subiteration thinning algorithm. Graph. Models Image Process. **61**, 199–221 (1999)
37. Palágyi, K., Németh, G.: Fully parallel 3D thinning algorithms based on sufficient conditions for topology preservation. In: Proc. 15th IAPR International Conference on Discrete Geometry for Computer Imagery, DGCI 2009. Lecture Notes in Computer Science, vol. 5810, pp. 481–492. Springer, Heidelberg (2009)
38. Palágyi, K., Tschirren, J., Hoffman, E.A., Sonka, M.: Quantitative analysis of pulmonary airway tree structures. Comput. Biol. Med. **36**, 974–996 (2006)
39. Raynal, B., Couprie, M.: Directional 3D thinning using 8 subiterations. In: Proc. 16th Int. Conf. on Discrete Geometry for Computer Imagery, DGCI 2011, Nancy, France. Lecture Notes in Computer Science, vol. 6607, pp. 175–186. Springer, Heidelberg (2011)
40. Shaked, D., Bruckstein, A.: Pruning medial axes. Comput. Vis. Image Underst. **69**, 156–169 (1998)
41. Siddiqi, K., Pizer, S. (eds.): Medial Representations—Mathematics, Algorithms and Applications. Computational Imaging and Vision, vol. 37. Springer, New York (2008)
42. Sundar, H., Silver, D., Gagvani, N., Dickinson, S.: Skeleton based shape matching and retrieval. In: Proc. Int. Conf. Shape Modeling and Applications, pp. 130–139. IEEE Press, New York (2003)
43. Tsao, Y.F., Fu, K.S.: A parallel thinning algorithm for 3-D pictures. Comput. Graph. Image Process. **17**, 315–331 (1981)
44. Wan, M., Liang, Z., Ke, Q., Hong, L., Bitter, I., Kaufman, A.: Automatic centerline extraction for virtual colonoscopy. IEEE Trans. Med. Imaging **21**, 1450–1460 (2002)
45. Wang, T., Basu, A.: A note on 'A fully parallel 3D thinning algorithm and its applications'. Pattern Recognit. Lett. **28**, 501–506 (2007)
46. Xie, W., Thompson, R.P., Perucchio, R.: A topology-preserving parallel 3D thinning algorithm for extracting the curve skeleton. Pattern Recognit. **36**, 1529–1544 (2003)

Chapter 7
Separable Distance Transformation and Its Applications

David Coeurjolly and Antoine Vacavant

Abstract In binary shape analysis, the distance transformation (DT) and its by-products are fundamental in many applications since they provide volumetric and metric information about the input shape. In this chapter, we present a survey on a specific approach (the dimension by dimension techniques) for the Euclidean metric and with discuss its performances and its generalizations to higher dimension or to specific grid models.

7.1 Introduction

Metric based description and analysis of shapes are fundamental notions in image analysis and processing. Among classical tools, the distance transformation (DT) [38, 53] of a binary image $I : \mathbb{Z}^d \to \{0, 1\}$ consists in labeling each point of a digital object E, defined as pixels with value 1 for instance on I, with its shortest distance to the complement of E. In the literature, DT has been widely used as a powerful tool in computer vision applications such as shape matching [20], pedestrian detection [26], human tracking [13], action recognition [33], robot motion planning [39, 71], or computation of morphological operators [19].

Another application of the DT is the computation of the medial axis (MA) of a digital shape [5, 43, 54], which is defined as the set of the centers of the largest balls contained in the processed object. MA is a very interesting shape representation because it is reversible, *i.e.* one can reconstruct the original object from its MA balls. Again, MA can be found in the literature in various applications such as surface reconstruction [1], shape simplification [62], volume representation [9] or smoothing [48].

D. Coeurjolly (✉)
LIRIS, UMR CNRS 5205, Université de Lyon, 69676 Villeurbanne, France
e-mail: david.coeurjolly@liris.cnrs.fr

A. Vacavant
Clermont Université, Université d'Auvergne, ISIT, CNRS, UMR6284, 63001 Clermont-Ferrand, France
e-mail: antoine.vacavant@iut.u-clermont1.fr

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_7,

Many techniques have been proposed to compute the DT and then the MA given a metric with a trade-off between algorithmic performances and the *accuracy* of the metric compared to the Euclidean one. Hence, we can consider distances based on chamfer masks [7, 29, 50, 53] or sequences of chamfer distances [44, 46, 53, 60]; the vector displacement based Euclidean distance [20, 22, 45, 49]; the Voronoi diagram based Euclidean distance [8, 32, 34, 40] or the square of the Euclidean distance [36, 41, 55]. From a computational point of view, several of these methods lead to time optimal algorithms to compute the error-free Euclidean Distance Transformation (EDT) for d-dimensional binary images [8, 34, 36, 40, 41]. The extension of these algorithms is straightforward since they use separable techniques to compute the DT; d one-dimensional operations—one per direction of the coordinate axis—are performed.

In this chapter, we focus on separable techniques for the Euclidean metric and its applications. First of all, we define the DT and the medial axis (MA) extraction thanks to discrete geometry concepts (Sect. 7.2). We present the formulations in d dimensions (or d-D), but for a sake of clarity, we generally illustrate the principles in the 2-D case. In Sect. 7.3, we present several extensions of these algorithms on special spaces, like toric plane or irregular isothetic grids. To deal with the processing of voluminous image data, high performance versions of DT (multithreaded, GPU) are finally discussed in Sect. 7.4.

7.2 Distance Transformation and Discrete Medial Axis Extraction

7.2.1 Distances

A distance (or *metric*) is a relation that is defined by four axioms:

Definition 1 (*Distance*) Let E be a non empty set and F be a sub-group of $\mathbb{R}$. A distance from E into F, denoted by (d, E, F), is a relation $d : E \times E \to F$ such that:

$$\text{Non-negativity} \quad \forall x, y \in E,\ d(x, y) \geq 0; \tag{7.1}$$

$$\text{Identity of indiscernibles} \quad \forall x, y \in E,\ d(x, y) = 0 \Leftrightarrow x = y; \tag{7.2}$$

$$\text{Symmetry} \quad \forall x, y \in E,\ d(x, y) = d(y, x); \tag{7.3}$$

$$\text{Triangle inequality} \quad \forall x, y, z \in E,\ d(x, y) \leq d(x, z) + d(z, y). \tag{7.4}$$

The first distances that have been used in image analysis are d_1 and d_∞ [54], respectively defined as

$$d_1(\mathbf{x}, \mathbf{y}) = |y_1 - x_1| + |y_2 - x_2| + \cdots + |y_d - x_d|; \tag{7.5}$$

$$d_\infty(\mathbf{x}, \mathbf{y}) = \max\left\{|y_1 - x_1|, |y_2 - x_2|, \ldots, |y_d - x_d|\right\}. \tag{7.6}$$

In the literature, for the $\mathbb{Z}^d$ case, d_1 distance is also called ℓ_1-metric or grid metric, and for $d = 2$, Manhattan, city block, taxi-cab or rectilinear metric. The d_∞ distance

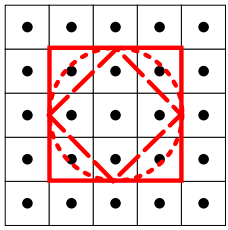


Fig. 7.1 Balls of distance 1, with different distances: d_8 in *plain lines*, d_E in *dotted lines*, and d_4 in *dashed lines*

is named ℓ_∞-metric, Chebyshev or uniform metric, and for $d = 2$, chessboard or square metric (cf. Fig. 7.1). Furthermore, d_1 and d_∞ are often called d_4 and d_8 in the 2-D case, d_6 and d_{26} in 3-D, because of the number of digital points at distance 1 of a given grid points (see further Definition 2).

The *Euclidean distance* is certainly the most employed distance, and is defined for two points $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_d)$ that belong to $\mathbb{R}^d$ by

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + \cdots + (y_d - x_d)^2}. \tag{7.7}$$

The squared Euclidean distance d_E^2 is another interesting application for image analysis, since $d_E^2(\mathbf{x}, \mathbf{y})$ has values in $\mathbb{Z}$, for all $\mathbf{x}, \mathbf{y} \in \mathbb{Z}^d$. Hence, even if d_E^2 is not a metric, it is a convenient exact representation of the Euclidean metric for which efficient algorithms can be designed.

Both d_1 and d_E metrics are in fact special cases of weighted ℓ_p metrics defined by

$$d_{\ell_p}(\mathbf{u}, \mathbf{v}) = \left(\sum_{i=0}^{d} w_i |u_i - v_i|^p \right)^{\frac{1}{p}} \tag{7.8}$$

with $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^d$ and $p \in \mathbb{R}^*$. Weights w_i can be set to represent anisotropic grids widely used in medical imaging for instance ($p = 2$ and $\{w_i = 1\}$ leads to the classical Euclidean distance on the regular square grid). From a computational point of view, algorithms presented here are illustrated with the Euclidean $d_{\ell_2} = d_E$ distance (cf. Fig. 7.2 for illustrations). However, many of them are valid for any ℓ_p metric.

From a given distance, one can also define a ball, which is an important notion for medial axis extraction:

Definition 2 (*Ball*) Let (d, E, F) be a distance, $\mathbf{p} \in E$ and $r \in F$. The ball B_d of center $\mathbf{p}$ and radius r is

$$B_d(\mathbf{p}, r) = \left\{ \mathbf{q} \in E \colon d(\mathbf{p}, \mathbf{q}) \leq r \right\}. \tag{7.9}$$

To consider an open ball, we just have to use a strict inequality in the previous equation.

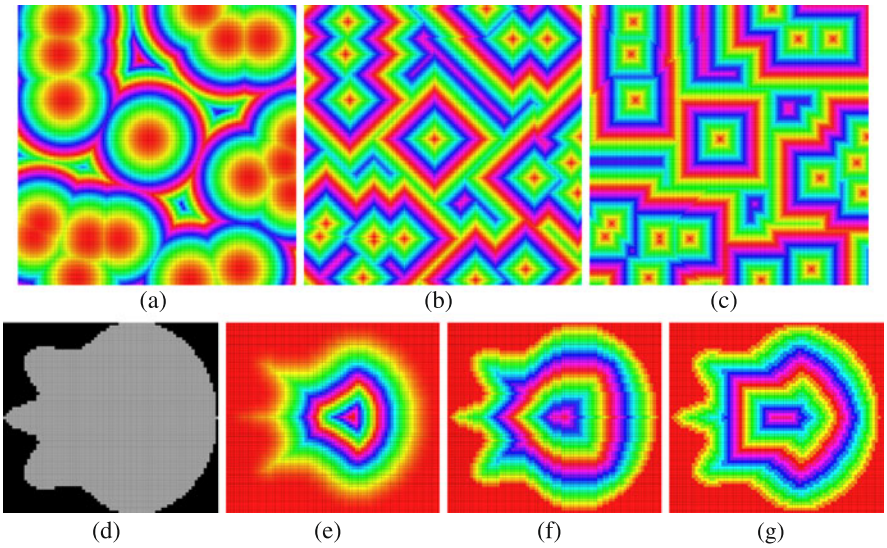


Fig. 7.2 Distance transformation of point sets (**a**–**c**) or of a 2-D shape (**d**–**g**) for the Euclidean metric, the ℓ_1 metric, and the ℓ_∞ metric, respectively

7.2.2 Distance Transformation

The distance transformation (DT) aims to compute the distance of each point of a discrete object E to the boundary of E. Let I be a binary image where value 1 pixels define the digital object E and value 0 pixels define its background $\overline{E}$. One defines the DT of $\mathbf{x} \in E$, based on a given distance d, as follows:

$$DT(\mathbf{x}) = \min\{d(\mathbf{x}, \mathbf{y}),\ \mathbf{y} \in \overline{E}\}. \tag{7.10}$$

In the rest of the chapter, we will focus on the DT based on the squared Euclidean distance d_E^2, that we denote by E^2DT. The first algorithms that were devoted to compute the E^2DT [22, 49] were inspired from by chamfer distance transformation [7], and unfortunately induced errors in the computation of the distance.

To correctly compute the E^2DT, various authors have chosen to construct a *separable algorithm* [27, 69]. The main idea of such technique is to compute the E^2DT by scanning a 2-D image with independent set of 1-D processes; one per dimension. This principle is thus easily extendable to higher dimensions. Let I be a binary image of size $n \times n$. A separable E^2DT algorithm consists in computing the image H:

$$\begin{aligned} H(i_1, i_2) = \min\{&(i_1 - x_1)^2 + (i_2 - x_2)^2: \\ &0 \leq x_1, x_2 < n \text{ and } I(x_1, x_2) \in \overline{E}\}, \end{aligned} \tag{7.11}$$

and this process is decomposed into two independent steps (see also Fig. 7.3):

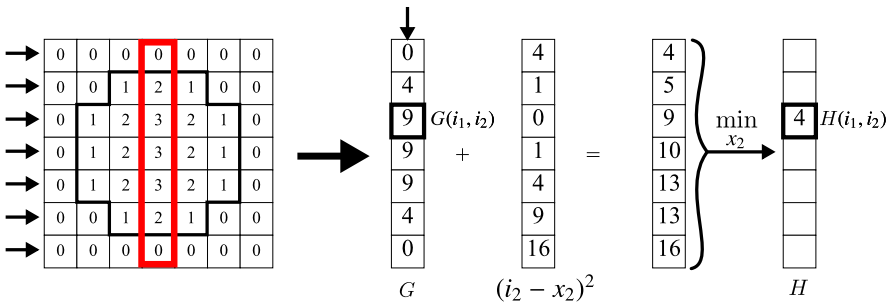


Fig. 7.3 Global overview of the two-dimensional E^2DT process. The first step of the algorithm computes G (**a**), representing minimal distances along the x-axis. If we choose one column of G, the second phase consists in computing $G(i_1, i_2)^2 + (i_2 - x_2)^2$ for each pixel of this column (**b**). The minimization of this column leads to the computation of the E^2DT stored in H

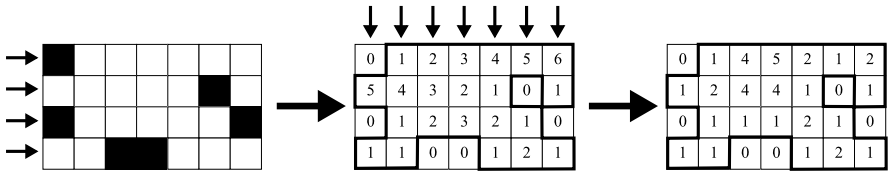


Fig. 7.4 Example of the two-dimensional E^2DT process over a small binary image

1. From image I, compute the G image with a one-dimensional minimization process over every row i_1:

$$G(i_1, i_2) = \min_{x_1} \{|i_1 - x_1|:\ 0 \le x_1 < n \text{ and } I(x_1, i_2) \in \overline{E}\}; \tag{7.12}$$

2. Then, compute the final result H, for any column i_2:

$$H(i_1, i_2) = \min_{x_2} \{G(i_1, x_2)^2 + (i_2 - x_2)^2:\ 0 \le x_2 < n\}. \tag{7.13}$$

In higher dimension, Eq. (7.11) can be generalized and decomposed into 1-D minimization steps defined in Eq. (7.13). The time complexity of the first step is $O(n^2)$, and $O(d \cdot n^d)$ in the general case, since it consists in a scan of the rows of image I (Fig. 7.5). Next step is also in $O(d \cdot n^d)$ complexity, and aims to compute the lower envelope of the set of parabolas (see Fig. 7.6)

$$\{\mathscr{F}_{x_2}^{i_1}(i_2) = G(i_1, x_2)^2 + (i_2 - x_2)^2\}, \quad \text{with } 0 \le x_2 < n. \tag{7.14}$$

This idea was first introduced by Saito and Toriwaki [55], but their algorithm was not linear in time. It was improved in [36] where using a stack based technique, the lower envelope of the set of parabolas of Eq. (7.14) is obtained in linear time (see Fig. 7.5). Hence, the overall DT computation is optimal in $O(d \cdot n^d)$ algorithm [36, 41].

Algorithm 1: First step along the X axis

Data: E a binary shape inside I, an $n \times n$ image.
Result: G the E^{2}DT along X axis.

```
1  for i2 = 0 to n − 1 do
2      if I(0, i2) ∈ Ē then
3          G(0, i2) ← 0;
4      else
5          G(0, i2) ← ∞;
6      for i1 = 0 to n − 1 do
7          if I(i1, i2) ∈ Ē then
8              G(i1, i2) ← 0;
9          else
10             G(i1, i2) ←
               1 + G(i1 − 1, i2);
11     for i1 = n − 2 to 0 do
12         if G(i1 + 1, i2) < G(i1, i2)
           then
13             G(i1, i2) ←
               1 + G(i1 + 1, i2);
```

Algorithm 2: Second step along the Y axis

Data: G obtained from step 1.
Result: H the E^{2}DT map.

```
1  for i1 = 0 to n − 1 do
2      q ← 0, s[0] ← 0, t[0] ← 0;
3      for i2 = 0 to n − 1 do
4          while
           q ≥ 0 ∧ F_{s[q]}(t[q]) >
           F_{i2}(t[q]) do
5              q ← q − 1;
6          if q < 0 then
7              q ← 0, s[0] ← i2;
8          else
9              w ← 1 + Sep(s[q], i2);
10             if w < n then
11                 q ← q + 1,
                   s[q] ← i2;
12                 t[q] ← w;
13     for i2 = n − 1 to 0 do
14         H(i1, i2) ← F_{s[q]}(i2);
15         if i2 = t[q] then
16             q ← q − 1;
```

Fig. 7.5 Separable algorithm for E^{2}DT of a 2-D binary image

In Fig. 7.5, we detail the pseudo-code of the E^{2}DT computation in dimension 2. The first step (Fig. 7.5-Alg. 1) corresponds to the distance propagation defined in Eq. (7.12) implemented with a 2-scan process. To compute the lower envelope of parabolas $\mathscr{F}^{i_1}_{x_2}(i_2)$, we define a function $Sep(u_1, u_2)$ which corresponds to the y-coordinate of the intersection point between two consecutive parabolas. For a given column i_1, it is given by:

$$Sep(u_1, u_2) = \big(u_2^2 - u_1^2 + G(i_1, u_2)^2 - G(i_1, u_1)^2\big)\mathrm{div}\big(2(u_1 - u_2)\big). \tag{7.15}$$

In Fig. 7.4, we show a small example of the E^{2}DT computation.

In [41], the authors present a way to change the $\mathscr{F}()$ and $Sep()$ functions in order to compute DT with other metrics. More precisely, for each l_p metric defined above, there exist functions $\mathscr{F}()$ and $Sep()$ for which algorithms in Fig. 7.5 produce exact l_p DT of binary images in dimension d.

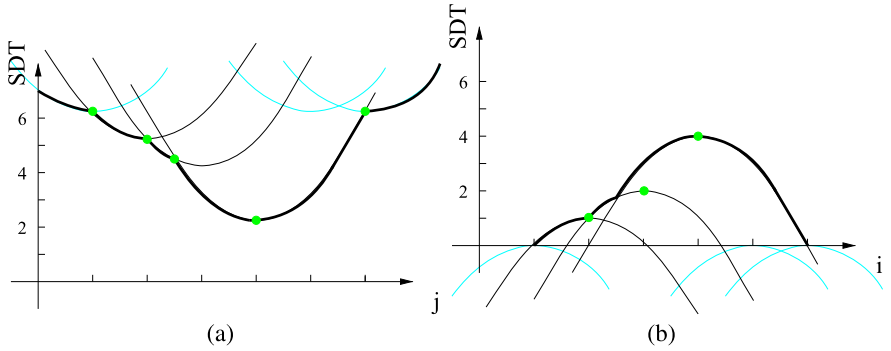


Fig. 7.6 Lower and upper envelope computations in E^2DT and the REDT problems

7.2.3 Reverse Distance Transformation

In this section, we discuss about the reconstruction problem of an object E thanks to a set of balls. Let us consider L, a set of l points $\{\mathbf{x}^m\}_{1\leq m\leq l} = \{(x_1^m, x_2^m, \ldots, x_d^m)\}_{1\leq m\leq l}$, and $r(\mathbf{x}^m)$ the associated squared Euclidean distance. A point $\mathbf{p}$ belongs to E if it belongs to at least one disk whose center is a point m of L, with radius $\sqrt{r(\mathbf{x}^m)}$. Hence, the reverse distance transformation (REDT) of L consists in obtaining the set of points P such that:

$$P = \big\{(i_1, i_2) \;\big|\; (i_1 - x_1)^2 + (i_2 - x_2)^2 < r(x_1, x_2),\ (x_1, x_2) \in L\big\}. \tag{7.16}$$

Let now F be an image of size $n \times n$ such that $F(i_1, i_2)$ is set to $r(i_1, i_2)$ if (i_1, i_2) belongs to L and to 0 otherwise. To obtain the REDT of L, we have to compute the image H:

$$\begin{aligned} H(i_1, i_2) = \max\big\{&F(x_1, x_2) - (i_1 - x_1)^2 - (i_2 - x_2)^2; \\ &0 \leq x_1, x_2 < n \text{ and } (x_1, x_2) \in F\big\}, \end{aligned} \tag{7.17}$$

then we extract the positive values of H. As in the case of the E^2DT, we can decompose this process into two main steps:

1. From image F, compute the G image with a one-dimensional maximization process

$$G(i_1, i_2) = \max_{x_1}\big\{F(x_1, i_2) - (i_1 - x_1)^2,\ 0 \leq x_1 < n\big\}; \tag{7.18}$$

2. Then, compute the final result H:

$$H(i_1, i_2) = \max_{x_2}\big\{G(i_1, x_2) - (i_2 - x_2)^2\colon\ 0 \leq x_2 < n\big\}. \tag{7.19}$$

Similarly to the previous section, Eqs. (7.18) and (7.19) correspond to the computation of the upper envelope of a set of parabolas $\{\mathscr{F}^*(i) = h - (i - x)^2\}$ with some $h \in \mathbb{Z}^+$ (see Fig. 7.6). Hence, we can use a similar algorithmic tool with specific *Sep*() functions. For example, for the first step, we have

$$Sep^*(u_1, u_2) = \big(u_1^2 - u_2^2 - F(u_1, i_2) + F(u_2, i_2)\big)\mathrm{div}\big(2(u_1 - u_2)\big). \tag{7.20}$$

Algorithm 3: REDT one-dimensional process (here, along x-axis)

Data: F the image of balls centers, of size $n \times n$.
Result: G the REDT result along X axis.

```
1  for i2 = 0 to n − 1 do
2      q ← 0, s[0] ← 0, t[0] ← 0;
3      for i1 = 1 to n − 1 do
4          while q ≥ 0 ∧ F*_{s[q]}(t[q]) < F*_{i1}(t[q]) do
5              q ← q − 1;
6          if q < 0 then
7              q ← 0, s[0] ← i1;
8          else
9              w ← 1 + Sep*(s[q], i1);
10             if w < n then
11                 q ← q + 1, s[q] ← i1;
12                 t[q] ← w;
13     for i1 = n − 1 to 0 do
14         G(i1, i2) ← F*_{s[q]}(i1);
15         if i1 = t[q] then
16             q ← q − 1;
```

Algorithm 3 details the 1-D upper envelope computation in linear time. To compute the REDT in dimension 2 or in higher dimensions d, we have to repeat this 1-D process dimension by dimension. The complexity of the overall algorithm in dimension d is $O(d \cdot n^d)$.

Figure 7.7 illustrates an example of the 2-D REDT process on a small image containing three Euclidean balls.

7.2.4 Medial Axis Extraction

In this section, we study the medial axis of a digital shape by considering the first definition of a maximal ball. Note that we always consider open balls in the rest of the chapter (see Definition 2).

Definition 3 (*Maximal ball*) A maximal ball is an open ball contained in the shape not entirely covered by another ball contained in the shape.

We now define the medial axis as follows (cf. Fig. 7.8 for an illustration in the continuous domain).

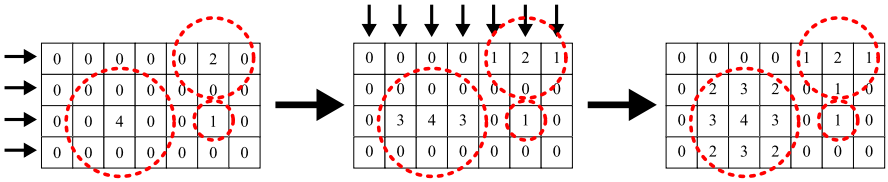


Fig. 7.7 Example of the two-dimensional REDT process over a small image with three Euclidean balls

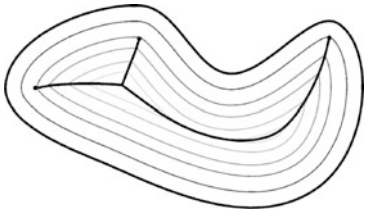


Fig. 7.8 Medial axis of a shape (*external contour*), illustration from [2]

Definition 4 (*Medial axis* [5, 54]) The medial axis of a shape is the set of maximal ball centers contained in the shape.

If we label each point $\mathbf{x}$ of the medial axis (MA) with the radius $\delta(\mathbf{x})$ of the maximal ball centered in $\mathbf{x}$, any shape $F \subset \mathbb{R}^d$ may be represented by the union of the maximal balls as follows:

$$F = \bigcup_{\mathbf{x} \in MA(F)} B\big(\mathbf{x}, \delta(\mathbf{x})\big), \quad \text{where } B(\mathbf{x}, r) = \big\{\mathbf{y} \in \mathbb{R}^d \colon d_E(\mathbf{x}, \mathbf{y}) < r\big\}. \tag{7.21}$$

An interesting property of balls and their maximality can be given in $d+1$ dimension, with the notion of *elliptic paraboloids*. For a sake of clarity, we consider a ball $B \subseteq \mathbb{R}^2$ of center $\mathbf{p} = (p_1, p_2)$ and radius r, and we denote

$$h_B(x_1, x_2) = r^2 - (x_1 - p_1)^2 - (x_2 - p_2)^2. \tag{7.22}$$

The ball B is thus the set of points $(x_1, x_2) \in \mathbb{R}^2$ such that $h_B(x_1, x_2) > 0$. We associate the ball B with the elliptic paraboloid $\widehat{B} \in \mathbb{R}^3$:

$$\widehat{B} = \big\{(x_1, x_2, x_3) \in \mathbb{R}^3 \colon 0 \leq x_3 < h_B(x_1, x_2)\big\}, \tag{7.23}$$

and the intersection between $\widehat{B}$ and the plane $x_3 = 0$ is the disk of center (p_1, p_2) and radius r. In [3], a similar object has been defined under the name of *ghost sphere*. If we now consider the Euclidean shape $F \subset \mathbb{R}^2$, Eq. (7.21) may be interpreted as:

$$\widehat{F} = \bigcup_{B \subset F} \widehat{B}, \quad \text{where } \widehat{F} \in \mathbb{R}^3. \tag{7.24}$$

As for the concept of maximal ball in $\mathbb{R}^2$ (Definition 3), we say that an elliptic paraboloid is maximal in $\widehat{F}$ if it is not entirely included in any other paraboloid in $\widehat{F}$. We have the following property (see also Fig. 7.9):

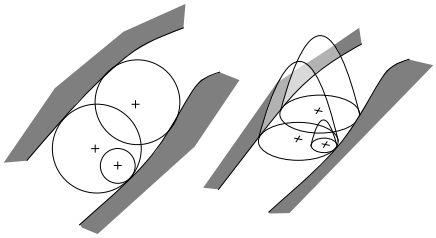


Fig. 7.9 Equivalence between inclusion of balls in 2-D and inclusion of elliptic paraboloids in 3-D

Lemma 1 (Maximality of a paraboloid [16]) *A ball B is maximal in F if and only if its associated elliptic paraboloid is maximal in $\widehat{F}$.*

When F is a finite union of balls, we can notice that the boundary of $\widehat{F}$ coincides with the upper envelope of the elliptic paraboloids associated with the balls B included in F. This envelope is defined by:

$$h(x_1, x_2) = \max_{B \subset F} h_B(x_1, x_2). \tag{7.25}$$

We also recall the following lemma, which permits to say that the elliptic paraboloid associated with a maximal ball reaches the upper envelope of paraboloids:

Lemma 2 (Ball maximality and upper envelope of paraboloids [16]) *Let $F \subset \mathbb{R}^2$ be a finite union of open balls. Let B be an open ball in $\mathbb{R}^2$. Then*:

$$B \text{ is maximal in } F \iff \exists (x_1, x_2) \in B,\ h_B(x_1, x_2) = h(x_1, x_2). \tag{7.26}$$

Note that a proof of this lemma follows properties of maximal balls [2].

We now propose to show how to extract a *discrete medial axis* of a digital shape E. This MA, denoted by $MA_{d_E}(E)$ is the set of maximal balls centers, generated with the Euclidean distance. $MA_{d_E}(E)$ has the property to reconstruct exactly the original object E (but it is generally not homotopic to E). This reconstruction process consists in computing the set of l balls $(\mathbf{x}_k, \delta(\mathbf{x}_k)) \in \mathbb{Z}^d \times \mathbb{N}$:

$$E = \bigcup_{1 \leq k \leq l} B\big(\mathbf{x}_k, \delta(\mathbf{x}_k)\big), \quad \text{where } B(\mathbf{x}, r) = \big\{\mathbf{y} \in \mathbb{Z}^d \colon d_E(\mathbf{x}, \mathbf{y}) < r\big\}. \tag{7.27}$$

Let $B(\mathbf{p})$ be the largest ball centered in $\mathbf{p}$ and contained in E. From the previous lemma, the set

$$MA_0(E) = \big\{\mathbf{p} \in \mathbb{Z}^2 \,\big|\, \exists (x_1, x_2) \in E,\ h_{B(\mathbf{p})}(x_1, x_2) = h(x_1, x_2)\big\} \tag{7.28}$$

is a subset of the MA of E, but is not included into the discrete MA of E. In fact, a ball may be maximal in E, although the Euclidean ball with the same center and radius may not be, as shown in Fig. 7.10. Indeed, in the $MA_0(E)$ set of Fig. 7.10(a), all balls are maximal in the continuous plane. However, the digital ball centered in the middle of the shape with radius $\sqrt{2}$ contains the digitizations of the four balls of radius 1. In [16], we describe a technique to filter balls in $MA_0(E)$ in order to keep maximal balls in the digital domain.

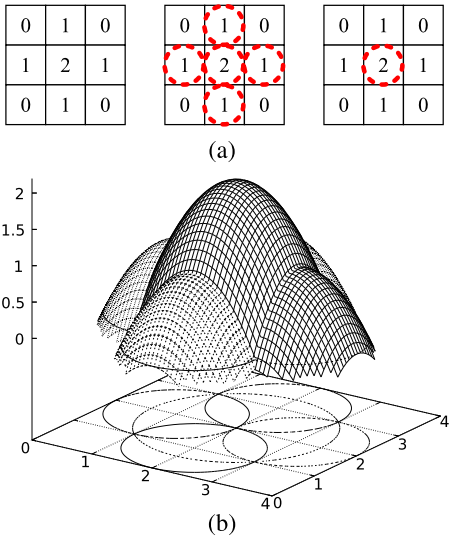


Fig. 7.10 (**a**—from *left* to *right*) A simple binary shape with its E^2DT, the points in $MA_0(E)$ and its discrete medial axis. (**b**) Illustration of the elliptic paraboloids involved in $MA_0(E)$

We can now sketch the algorithm for obtaining the *reduced discrete medial axis* (RDMA) of a digital shape E. As discussed above, balls in $RDMA(E)$ have the following properties: they are maximal in the digital domain and the union of RDMA balls is equal to E. In an algorithmic point of view, one can see that Eq. (7.22) corresponds to the upper envelope computation process defined in the REDT algorithm (see Sect. 7.2.3). Hence, from a binary shape E defined in an image I, the RDMA can be obtained as follows (see [16] for details):

1. Compute the E^2DT of E;
2. Use the REDT algorithm on the E^2DT map to mark balls belonging to the upper envelope of elliptic paraboloids (Eq. (7.28));
3. Remove maximal continuous balls which are not maximal in the digital case.

As discussed above, the first two steps can be done in $O(c \cdot n^d)$. To filter non maximal digital balls, we have defined a simple algorithm which filters the balls in the 1-D process during the REDT computation [16]. Hence, we finally obtain a RDMA extraction algorithm in optimal $O(c \cdot n^d)$ time (cf. Fig. 7.11 for a result).

7.2.5 Voronoi Diagrams and Power Diagrams

7.2.5.1 E^2DT and Voronoi Diagram

The resolution of the DT formulation given in Eq. (7.10) in the discrete space can be addressed by computing *the Voronoi diagram* (VD) of the background pixels (*i.e.*

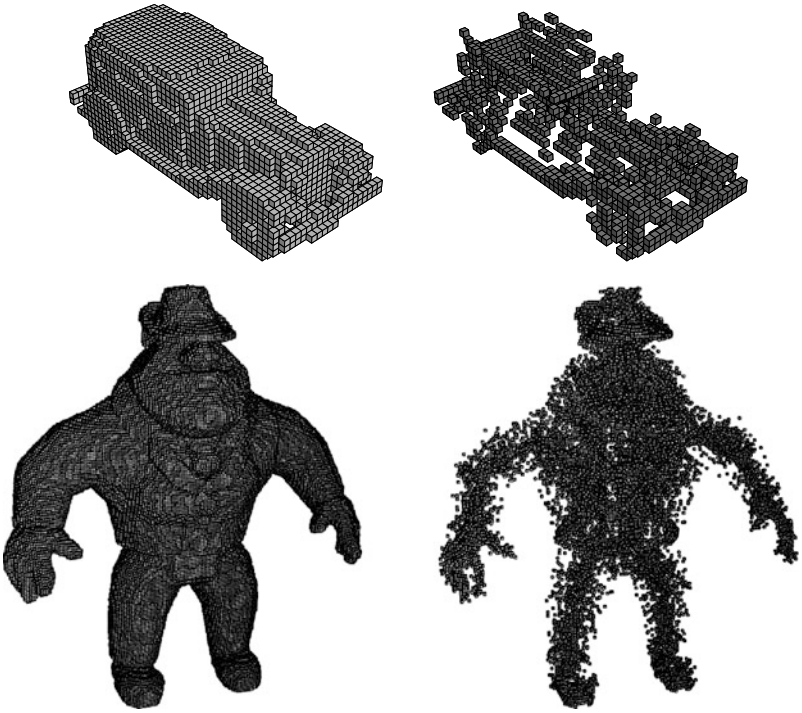


Fig. 7.11 Examples of RDMA extraction on 3-D shapes

Voronoi sites). We recall that the VD of a set of N points $P = \{\mathbf{p}_i\}_{i=1,\dots,N}$ is a tiling of the plane into *Voronoi cells* (or *VD cells*) $\{C_{\mathbf{p}_i}\}_{i=1,\dots,N}$ such that [23, 70]:

$$C_{\mathbf{p}_i} = \left\{\mathbf{x} \in \mathbb{Z}^d \colon d_E(\mathbf{x}, \mathbf{p}_i) \leq d_E(\mathbf{x}, \mathbf{p}_j),\ \forall j \neq i\right\}. \tag{7.29}$$

Hence, a simple approach to compute the E^{2}DT (presented in Fig. 7.12 and Algorithm 4) is to pre-compute the complete VD of the background points, and to locate foreground points in the VD. In the 2-D case, the E^{2}DT computation has a $O(n \times n \log n_B)$ time complexity, where $n \times n$ is the total number of pixels, and $n_B = \#\overline{E}$ is the number of background pixels. More precisely, the Voronoi Diagram V can be computed thanks to the CGAL library [37],[1] and has an optimal time complexity $O(n_B \log n_B)$ and requires $O(n_B)$ space [25]. Then, the main loop of this algorithm consists in locating each foreground point $\mathbf{p}$ in V, and in searching for its nearest Voronoi site $\mathbf{s}$ in V_P. The location query is proved to have a $O(\log n_B)$ complexity [25]. Thus, we perform this loop in $O(n_F \log n_B)$ time, where $n_F = \#E$. Indeed, searching for the nearest site $\mathbf{s}$ is processed in constant time in the three cases: $\mathbf{p}$ is inside a Voronoi cell, $\mathbf{p}$ stands on a Voronoi edge, and $\mathbf{p}$ is on a Voronoi

[1] CGAL, Computational Geometry Algorithms Library, http://www.cgal.org.

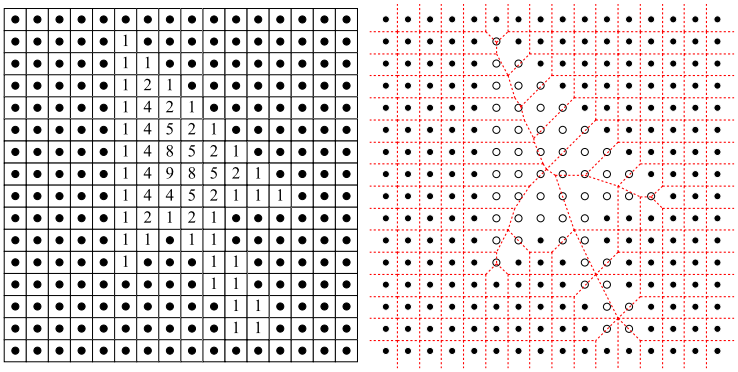


Fig. 7.12 The computation of the E^2DT (**a**) can be obtained by constructing the VD of the background pixels (*black points*). The VD is illustrated by *dotted lines* in (**b**)

vertex. When **p** stands on a vertex or on an edge of V, the choice of the Voronoi cell containing s is arbitrary.

This technique is obviously not computationally efficient for large images, and the extension of this transformation to d-D is a hard work. A VD can be computed in d-D with a $O(n_B \log n_B + n_B^{\lceil d/2 \rceil})$ time complexity (thanks to a gift-wrapping approach [23], for example). However, localizing a point in the VD is an arduous task, and an additional structure like subdivision grids [47] should be constructed to handle this operation.

Algorithm 4: Computation of the E^2DT thanks to a VD-based process

Data: E a binary shape inside I, a $n \times n$ image.
Result: The E^2DT of each point of E, stored in H.

```
1  compute the Voronoi Diagram V_P of the points P = {q: q ∈ Ē};
2  for each point p ∈ E do
3      locate p in V_P;
4      if p is the Voronoi vertex v of V_P then
5          s ← Voronoi site of an adjacent cell of v;
6      else if p belongs to a Voronoi edge e in V_P then
7          s ← Voronoi site of an adjacent cell of e;
8      else                                   {p belongs to a Voronoi cell C of V_P}
9          s ← Voronoi site of C;
10     H(p) ← d²_E(s, p);
11 for each point q ∈ Ē do
12     H(q) ← 0;
```

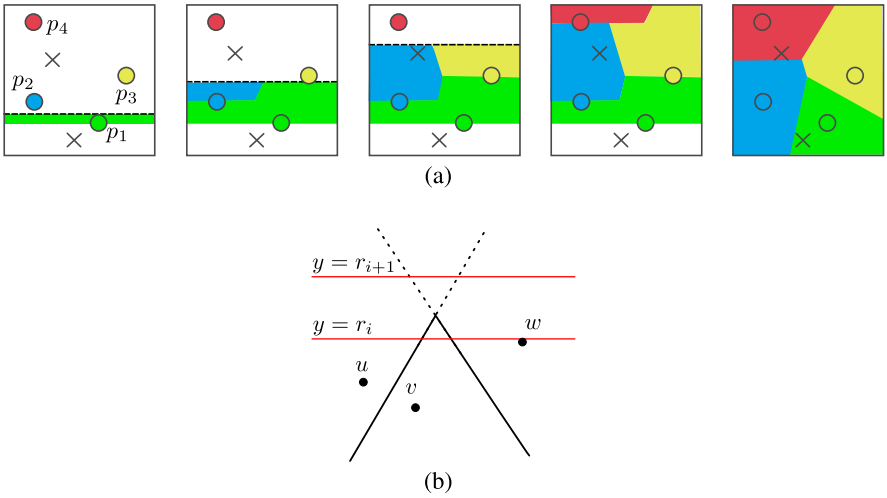


Fig. 7.13 (**a**) Several stages of the algorithm of Breu et al. [8] in an example with four background pixels $p_1, \ldots, p_4$ (*circles*) and two foreground pixels (*crosses*). During the first sweeping stage, they build the discrete VD, intersection between the $\mathbb{Z}^d$ and the complete VD of the background points. In the last step, the point p_1 is not considered anymore, because it is hidden by p_2 and p_3 (see also (**b**)). Last figure is the result of the second stage, where the foreground points are affected by the correct E^2DT

As a consequence, many algorithms devoted to compute the E^2DT of binary d-D images only compute a *partial discrete VD of the background pixels* [8, 40, 58] in order to be computationally efficient in any dimension. A first approach developed by Breu et al. [8] aims to compute the E^2DT of a binary image by using a sweep-line process. During the two scans of the image, a predicate decides if a Voronoi site u is *hidden by* two others sites v and w (see Fig. 7.13). In the 2-D case, at each Y-coordinate $y = r_i$ of the input image I, this predicate is true if

$$\begin{aligned}(w_1 - v_1) \times \left(v_1^2 - u_1^2 - 2r_i(v_2 - u_2) + v_2^2 - u_2^2\right)\\ \geq (v_1 - u_1) \times \left(w_1^2 - v_1^2 - 2r_i(w_2 - v_2) + w_2^2 - v_2^2\right).\end{aligned} \tag{7.30}$$

Maurer, Qi and Raghavan [40] propose to adapt the separable scheme proposed in [36, 41, 55] to extend this methodology to d-D, with an algorithm that prunes the VD in each dimension. Such an algorithm is now considered as a reference. As an example, the E^2DT algorithm coded in the famous `ITK` (Insight ToolKit) library is inspired from this work.[2]

[2]More details can be found in [63] and on the page http://www.itk.org/Doxygen/html/classitk_1_1SignedMaurerDistanceMapImageFilter.html.

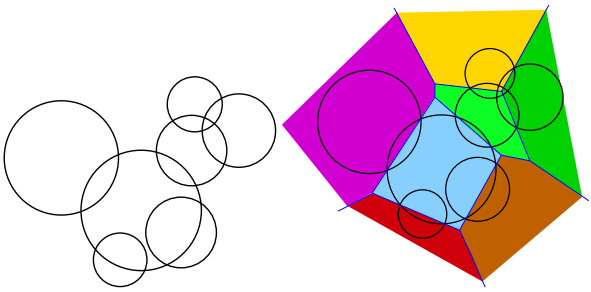


Fig. 7.14 Power diagram construction: the input set of circles in the plane (**a**), and its power diagram (**b**)

7.2.5.2 REDT and Power Diagram

We can describe similar links between the REDT computation and the computation of a specific diagram: the *power diagram* (also known as the *Laguerre diagram*) [4]. First, let us consider a set of N sites $S = \{\mathbf{c}_i\}_{i=1,\ldots,N}$ such that each point $\mathbf{c}_i$ is associated with a radius r_i (see Fig. 7.14(a)). The power $\sigma_i(\mathbf{x})$ of a point $\mathbf{x}$ in $\mathbb{R}^d$ according to the site $\mathbf{c}_i$ is given by $\sigma_i(x) = \|\mathbf{x} - \mathbf{c}_i\|^2 - r_i^2$. If $\sigma_i(\mathbf{x}) < 0$, $\mathbf{x}$ belongs to the disk of center $\mathbf{c}_i$ and radius r_i. If $\sigma_i(\mathbf{x}) > 0$, $\mathbf{x}$ is outside the ball. The *power diagram* is a kind of Voronoi diagram based on the metric σ. Hence, the power diagram V_S is a decomposition of the space into open cells $F = \{C_{\mathbf{c}_i}\}_{i=1,\ldots,N}$ associated with each site $\mathbf{c}_i$ such that

$$C_{\mathbf{c}_i} = \left\{\mathbf{x} \in \mathbb{R}^d \colon \sigma_i(\mathbf{x}) < \sigma_j(\mathbf{x}),\ \forall j \neq i\right\}. \tag{7.31}$$

Note that cell $C_{\mathbf{c}_i}$ associated with site $\mathbf{c}_i$ may be empty, otherwise, $C_{\mathbf{c}_i}$ is a convex polytope (see Fig. 7.14 in dimension 2). The power diagram is a common tool in computational geometry when a geometry of spheres or hyper-spheres must be taken into account [1, 4, 6].

In the REDT computation described in Sect. 7.2.3, Eq. (7.17) corresponds to the set of points (i_1, i_2) for which there exists a ball j with power distance $\sigma_j(i_1, i_2) < 0$.

Similarly to the previous section and since faces $C_{\mathbf{c}_i}$ of the power diagram are obtained by minimizing all power distances, a simple algorithm to solve the REDT problem can be sketched as follows:

- Given an input set of points with radii, we construct the power diagram.
- For each grid point $\mathbf{p} \in \mathbb{Z}^d$, we locate $\mathbf{p}$ in the power diagram.
- The reconstructed binary shape P corresponds to points $\mathbf{p}$ with negative power distance.

In dimension d, the construction of the power diagram has gotten the same computational cost as the Voronoi diagram construction. Hence, the overall the discussion on computation detailed in Sect. 7.2.5.1 is still valid in the power diagram case. Furthermore, algorithms that compute Voronoi diagram mappings in the digital grid can be adapted to be able to construct digital power diagram mappings [16].

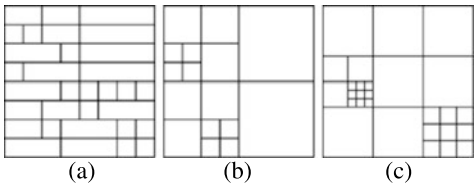


Fig. 7.15 Examples of $\mathbb{I}$-grids: grid obtained with a RLE (Run Length Encoding) scheme (**a**), quadtree-based grid (**b**) and an adaptive grid (**c**)

7.3 Extensions and Generalizations

7.3.1 Irregular Grids

As discussed in Sect. 7.2, the DT and RDMA extraction processes are widely studied and developed on regular grids. Some specific extensions of the DT to non-regular grids also exist, such as elongated grids [12, 29, 59], quadtrees/octrees [57, 71], Face-Centered Cubic (FCC)/Body-Centered Cubic (BCC) grids [31], *etc*. In a similar way, some works aimed to extend the computation of a skeleton to other non-standard regular grids (triangular, hexagonal, rectangular grids) [14, 24], and to quadtree/octree representations [56, 72]. In this section, we present generic DT and RDMA extraction processes, designed on every kind of image representations in two or higher dimensions.

We first recall the $\mathbb{I}$-grid model [17, 64, 68]:

Definition 5 (*d-D* $\mathbb{I}$*-grid*) Let P be a closed rectangular subset of $\mathbb{R}^d$. A d-D $\mathbb{I}$-grid G is a tiling of P with non overlapping hyper-rectangles (or cells) whose $d-1$ faces are parallel to the successive axes of the chosen space. The position of each cell is characterized by its center point (its position) $(p_1^R, p_2^R, \ldots, p_d^R) \in \mathbb{R}^d$ and its length along every axes $(\ell_1^R, \ell_2^R, \ldots, \ell_d^R) \in \mathbb{R}_+^{*\,d}$.

In the rest of the section, we will use for illustration mainly 2-D grids, and we will consider them binary, *i.e.* they contain cells that are labeled background or foreground (for a given grid G, denoted G_B and G_F, respectively). Some examples of classic grids in imagery are given in Fig. 7.15. The $\mathbb{I}$-grid model is able to represent these grids in any dimension. Even if the distance between discrete points in $\mathbb{Z}^d$ is a very natural notion, according to $\mathbb{R}^d$, distance in an $\mathbb{I}$-grid should be precisely addressed.

7.3.1.1 E^{2}DT Computation on $\mathbb{I}$-Grids

Here, we first recall the two extensions of the E^{2}DT proposed on $\mathbb{I}$-grids, for each cell $R \in G_F$:

$$\mathbb{I}\text{-CDT}(R) = \min_{R'} \left\{ d_E^2\left(\mathbf{p}^{\mathbf{R}}, \mathbf{p}^{\mathbf{R}'}\right) :\ R' \in G_B \right\}, \tag{7.32}$$

$$\mathbb{I}\text{-BDT}(R) = \min_{s} \left\{ d_E^2\left(\mathbf{p}^{\mathbf{R}}, s\right) :\ s \in S \right\}, \tag{7.33}$$

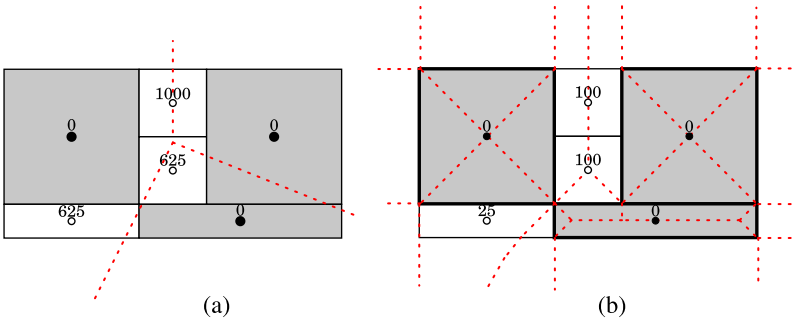


Fig. 7.16 Example of the VD of the background points to obtain the $\mathbb{I}$-CDT of a simple $\mathbb{I}$-grid (**a**) and the background/foreground frontier (dark segments) to obtain the $\mathbb{I}$-BDT (**b**)

where S is the set of segments contained in the background/foreground frontier of the grid. The first extension, the Center-based DT (Eq. (7.32)) [66] is based on the cell centers of the grid. The Border-based DT (Eq. (7.33)) [67] employs cell borders, and computes the shortest distance between a foreground cell center and the background/foreground boundary.

We can compute these extensions thanks to the construction of two kinds of VD (see Sect. 7.2.5). The $\mathbb{I}$-CDT may be computed thanks to a similar algorithm to the one illustrated in Algorithm 4. If we now consider the background/foreground frontier to compute the $\mathbb{I}$-BDT, Eq. (7.33) implies that we compute a VD of segments, and not a classical VD of points (see Fig. 7.16 for an example of these diagrams computed on a simple $\mathbb{I}$-grid). As in the discrete case, a complete VD-based $\mathbb{I}$-CDT technique is obviously not computationally efficient for every grids, and not adapted to dense grids [66]. To compute the $\mathbb{I}$-BDT, a similar process can be drawn, where the computation of the VD of segments can be handled in $O(n_S \log^2 n_S)$, where $n_S = 4n_B$ is the total number of segments belonging to the background/foreground frontier [37]. The extension of both VD-based transformations ($\mathbb{I}$-CDT and $\mathbb{I}$-BDT) to d-D $\mathbb{I}$-grids is difficult. In the following, we show how to use separable techniques in order to compute them.

To develop a separable process on $\mathbb{I}$-grids, we denote by $\mathbf{A}$ the *irregular matrix* associated with a labeled $\mathbb{I}$-grid G, introduced in [66]. This data structure aims to organize the cells of the grid along the X and Y axis by adding virtual cells centers (see Fig. 7.17 for an example).

The nodes and the border attributes are used to propagate the $\mathbb{I}$-CDT and $\mathbb{I}$-BDT distance values through the irregular matrix and then compute a correct distance value for each cell center. Thanks to the irregular matrix, we are able to compute the two extensions of the DT on $\mathbb{I}$-grids from Eq. (7.32) and Eq. (7.33), in linear time, according to the irregular matrix size (see [68] for more details). In Fig. 7.18, we present the result of the computation of the $\mathbb{I}$-CDT on various digitizations of images *ghost* and *lena*. Even if the hexagonal grid is not properly an $\mathbb{I}$-grid, the result of the $\mathbb{I}$-CDT is the DT based on the hexagonal tiles centers.

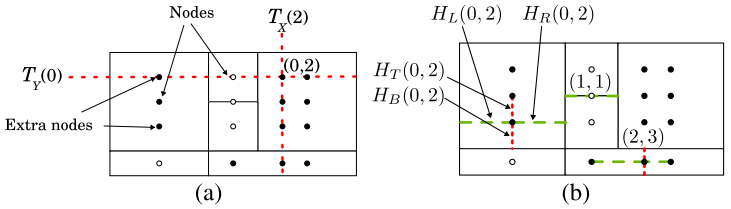


Fig. 7.17 Construction of the irregular matrix associated with the simple $\mathbb{I}$-grid illustrated in Fig. 7.16. In (**a**), the extra node $\mathbf{A}(0,2)$ of the matrix is depicted at the intersection of the *dotted lines*. In (**b**), examples of values for border attributes are also given along X (*dashed lines*) and Y (*dotted lines*). For instance, we have $H_R(0,2) = H_L(0,2) = 20$, $H_T(0,2) = H_B(0,2) = 10$, $H_R(1,1) = H_L(1,1) = 10$, and $H_T(1,1) = H_B(1,1) = 0$ since this node coincides with a cell horizontal border. We can also notice that $H_L(2,3) \neq H_R(2,3)$ while $H_T(2,3) = H_B(2,3) = 5$

It is also possible to extend the notion of RDMA on $\mathbb{I}$-grids [65]. In Fig. 7.19, we also show an application of $\mathbb{I}$-RDMA for computing an adaptive MA. We consider a quadtree decomposition of a binary image, with an increasing level of subdivision.

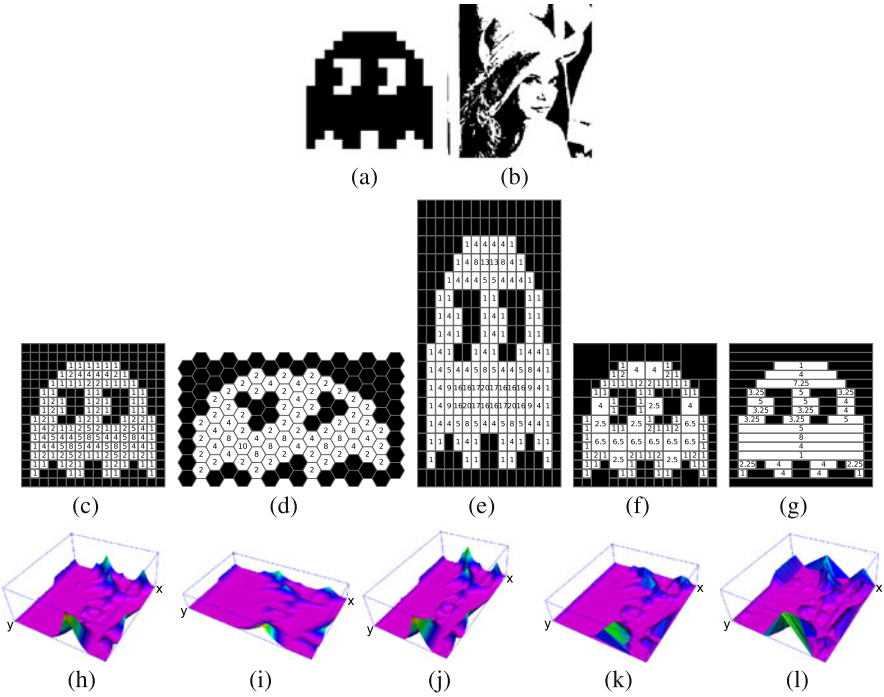


Fig. 7.18 From several digitizations of the binary images *ghost* and *lena* (**a**–**b**), we propose to compute the $\mathbb{I}$-CDT, viewed as a distance map for *ghost* (**c**–**g**). In the bottom, we also display the elevation map associated with each case of the image *lena*: (**h**) regular square grid, (**i**) hexagonal grid, (**j**) rectangular grid, (**k**) quadtree decomposition, (**l**) RLE

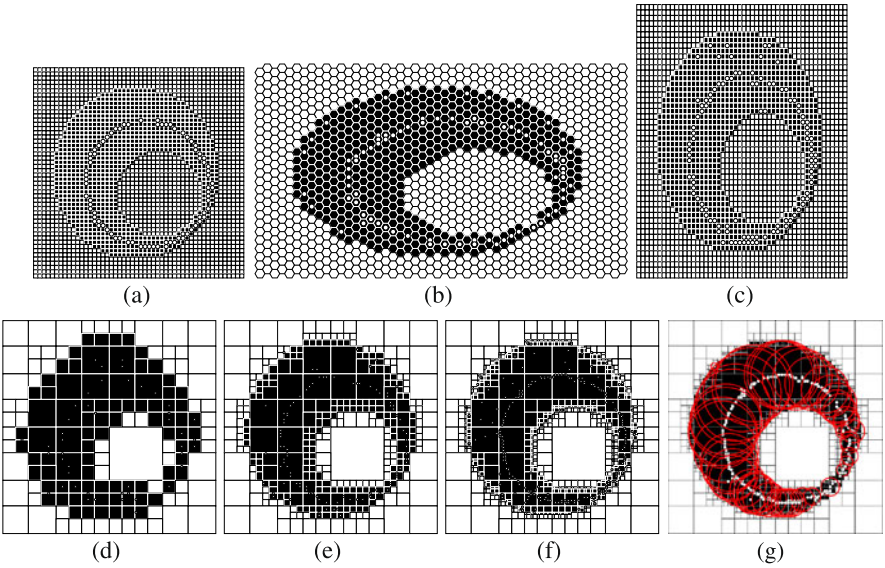


Fig. 7.19 *Top*: Extraction of the $\mathbb{I}$-RDMA on several regular image structures (square, hexagonal, rectangular). *Bottom*: Progressive extraction of the $\mathbb{I}$-RDMA for three levels of quadtree decomposition of the same image. We present in (**g**) a quarter of the balls associated with the final decomposition

We finally present a part of the balls associated with the MA points for the final decomposition.

In this irregular case, we have increased the size of the input with the help of the irregular matrix construction. In some situations such as irregular anisotropic domains, no overload exists. However, pathological cases may lead to an irregular matrix with quadratic size (compared to the input set of irregular cells). However, experimental evaluation shows that the separable algorithms still outperform Voronoi diagram based construction for instance.

7.3.2 Toric Domains

In many material sciences applications, discrete toric domains are widely used to model and analyze structures based on a sample of the material and assuming that the overall material can be approximated as a regular tiling of this sample. In order to make the measurements consistent through tiling, we have to consider that the sample is embedded in a toric space. Discrete toric spaces in higher dimension can be defined as direct products of 1-D cyclic domains [11, 15]. In order to perform volumetric analysis such as distance transformation, we have to handle the boundary propagation correctly. Since the 1-D envelope computations involved in the DT, REDT and RDMA algorithms are independent, we just have to define 1-D toric

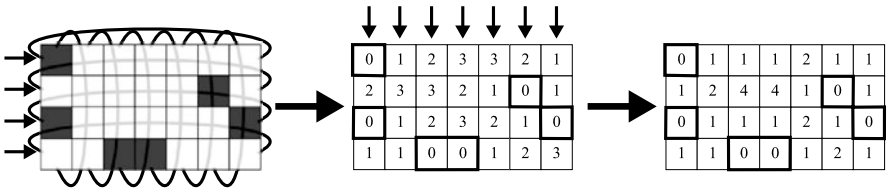


Fig. 7.20 Example of E^2DT computation on a toric domain

envelope computation in order to make all the algorithms remain consistent in the toric model (see Fig. 7.20).

In [15], we have detailed a modification of the envelope parabola computations defined earlier in order to take into consideration cyclic 1-D domains. Using these modified 1-D process, linear in time separable volumetric algorithms can be defined on d-D toric domains.

7.4 High Performance Computation

In many applications, efficient volumetric analysis is required in order to achieve real-time computations, or to be able to deal with large data-sets. For the first point, we focus on parallel implementation of separable algorithms on both CPU and GPU. In Sect. 7.4.2, we briefly discuss about out-of-core distance transformation computation.

7.4.1 Parallel Computation

Let us first consider a simple multithreaded environment. As illustrated in Fig. 7.4, for each dimensional step, we have independent 1-D processes. Hence we have a straightforward but optimal multithread implementation that can be sketched as follows in dimension 2. Let us consider p processors (or threads) in share memory model. During the first step, we decompose the loop in line 1 of Fig. 7.5-Alg. 1 into a parallel loop with p chunks of n/p rows attached to each processor. At the end of the first step, we have to synchronize the processors and continue with the second step decomposing loop 1 of Fig. 7.5-Alg. 2 in a similar way.

In dimension d, we can repeat the loop decomposition to obtain an optimal $O(d \cdot \frac{n^d}{p})$ parallel algorithm with p threads.

Graphical Processing Unit (GPU) can be considered as a specific parallel computing device with fine grain parallelism. Beside the fact that the 1-D envelope processes can be computed in parallel, the stack structure involved in the computation is not well-adapted to GPU computing. Existing techniques either consider approximated solution with errors [21, 51, 52] or may not be optimal in terms of parallelism and work-load [61].

In parallel computing, the work-load corresponds to the sum of atomic operations for all processors. For example, the *jump flooding algorithm* consists in propagating minimum distance information for each pixel (x, y) according to neighbors $(x + \{-k, 0, k\}, y + \{-k, 0, k\})$. The step length k ranges for $n/2, n/4, \ldots, 1$. Even if this algorithm may introduce errors in the distance computation, every pixel has exactly the same microprogram to execute which makes the overall process efficient. Given an $n \times n$ image, the jump flooding algorithm has $O(\log n)$ steps, where each step takes $O(n^2/p)$ for p processors. Hence, the total work amounts to $O(n^2 \log n)$ which is not optimal since the optimal sequential algorithm is $O(n^2)$.

Recently, [10] have proposed a banding approach that splits the 1-D envelope computations into chunks in order to improve the parallel efficiency. The work-load is still not optimal but we can thus obtain a fast and error-free Euclidean DT on GPU.

Optimal in time and work-load algorithms exist in the Common-CRCW and EREW parallel models [35]. Authors describe an algorithm for the distance transformation which runs in $O(\log n)$ time in the EREW model with $O(\frac{n^2}{\log n})$ processors (resp. $O(\log \log n)$ time with $O(\frac{n^2}{\log \log n})$ processors for the CRCW model), leading to an optimal linear in time work. Note that modern GPUs have a specific parallel model in-between the CRCW and EREW ones. As far as we know, these optimal theoretical algorithms have been implemented and experimented on GPU.

7.4.2 Out-of-Core Approaches

The previous section focuses on the decomposition of the volumetric computation into independent (and thus parallel) processes. In some applications, one could be more concerned with the size of the input shape to analyze rather than with the speed of the computation. However, we are still facing similar propagation issues. If the input shape cannot fit into memory, the general framework can be sketched as follows: First, decompose the volumetric shape into blocks, load and process each block, and write the final or intermediate result to an external storage. If propagation between blocks occurs, we may have to process several time the same block. Additionally, we may also need a way to detect that propagation has to be handled. In the literature, this is called *out-of-core algorithms*.

The first approach to the solution to this problem could be to exploit 1-D decomposition (illustrated for the DT problem in dimension 2):

- Decomposing the input shape into 1-D rows, we group the rows into chunks depending on the available memory.
- For each chunk, load it, compute the 1-D distance transformation on independent rows and write the result into a temporary volume on the external storage.
- Decompose the intermediate map into 1-D columns and group them into chunks.
- Process each chunk similarly with independent 1-D column computation and write the result.

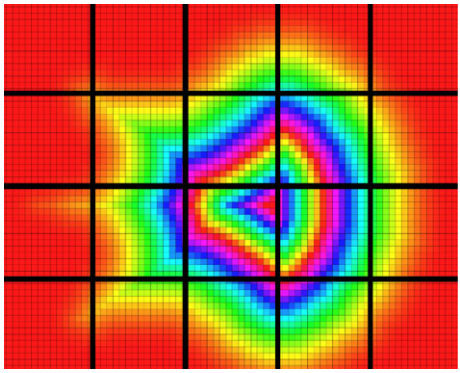


Fig. 7.21 Block decomposition for out-of-core distance transformation computation

The main advantage of this technique is that there is no distance propagation between 1-D processes. Each block is thus processed once per dimension. The main drawback corresponds to the row/column decomposition of both the input shape or the temporary one. Indeed, since the 2-D image is linearized on the file system (without loss of generality, we assume that rows are concatenated), the extraction of a set of columns could be inefficient.

In many situations, we usually prefer to decompose the input object domain into hyper-rectangular blocks (Fig. 7.21). However, we have to deal in that case with distance propagation across the boundaries. Some studies have proposed different techniques to control this propagation (see for example [28, 30, 42]). On the latter proposal, even if the internal distance transformation on each block introduces errors, they use a very interesting structure, so called sparse grid, which only contains distance information on the boundary of each block in the decomposition. Hence, once such boundary conditions are computed offline, the complete distance transformation on a block can be obtained without any propagation through the volume. In [42], the proposed algorithm is not optimal and may contain errors. However, correcting such drawbacks with separable technique is an interesting challenge.

7.5 Discussion and Open Problems

In this chapter, we have detailed separable techniques to perform volumetric quantization on digital shapes with the help of metric information. These techniques offer several computational advantages and extensions to specific metrics or domains have been presented.

At this point, we would like to focus on two challenging future works for high performance computation. The first one deals with fast GPU implementation. Indeed, as far as we know, no efficient implementation of the workload optimal DT algorithm has been proposed on GPU. The main motivation here is driven by the fact that DT is widely used as a core tool in many computer vision and image synthesis real time applications.

The second challenge corresponds to the out-of-core implementation of the optimal and exact volumetric tools. Indeed, in many material science or medical applications, it becomes common to deal with 2048^3 binary shapes to analyze [18]. In this context, specific strategies have to be investigated.

Many figures of this chapter have been generated by the DGtal library[3] which provides an implementation of these separable techniques on d-D images.

References

1. Amenta, N., Choi, S., Kolluri, R.K.: The power crust, unions of balls, and the medial axis transform. Comput. Geom. **19**(2–3), 127–153 (2001)
2. Attali, D., Boissonnat, J.-D., Edelsbrunner, H.: Stability and computation of medial axes—a state-of-the-art report. In: Mathematical Foundations of Scientific Visualization, Computer Graphics, and Massive Data Exploration. Mathematics and Visualization, pp. 109–125. Springer, Berlin (2009)
3. Attali, D., Edelsbrunner, E.: Inclusion-exclusion formulas for independent complexes. Discrete Comput. Geom. **37**(1), 59–77 (2007)
4. Aurenhammer, F.: Power diagrams: properties, algorithms, and applications. SIAM J. Comput. **16**, 78–96 (1987)
5. Blum, H.: A transformation for extracting descriptors of shape. In: Models for the Perception of Speech and Visual Forms (1967)
6. Boissonnat, J.D., Cerezo, A., Devillers, O., Duquesne, J., Yvinec, M.: An algorithm for constructing the convex hull of a set of spheres in dimension d. Comput. Geom. **6**(2), 123–130 (1996)
7. Borgefors, G.: Distance transformations in digital images. Comput. Vis. Graph. Image Process. **34**(3), 344–371 (1986)
8. Breu, H., Kirkpatrick, D., Werman, M.: Linear time Euclidean distance transform algorithms. IEEE Trans. Pattern Anal. Mach. Intell. **17**(5), 529–533 (1995)
9. Broutta, A., Coeurjolly, D., Sivignon, I.: Hierarchical discrete medial axis for sphere-tree construction. In: International Workshop on Combinatorial Image Analysis, IWCIA 2009. Lecture Notes in Computer Science, vol. 5852, pp. 56–67. Springer, Berlin (2009)
10. Cao, T.T., Tang, K., Mohamed, A., Tan, T.S.: Parallel banding algorithm to compute exact distance transform with the GPU. In: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, SI3D 2010, pp. 83–90 (2010)
11. Chaussard, J., Bertrand, G., Couprie, M.: Characterizing and detecting toric loops in n-dimensional discrete toric spaces. In: International Conference on Discrete Geometry for Computer Imagery, DGCI 2008. Lecture Notes in Computer Science, vol. 4992. Springer, Berlin (2008)
12. Chehadeh, Y., Coquin, D., Bolon, P.: A skeletonization algorithm using chamfer distance transformation adapted to rectangular grids. In: IEEE International Conference on Pattern Recognition, ICPR 1996, vol. 2, pp. 131–135 (1996)
13. Chen, L., Panin, G., Knoll, A.: Multi-camera people tracking with hierarchical likelihood grids. In: International Conference on Computer Vision Theory and Applications, VISAPP 2011 (2011)
14. Ciuc, M., Coquin, D., Bolon, P.: Quantitative assessment of two skeletonization algorithms adapted to rectangular grids. In: International Conference on Image Analysis and Processing, CIAP 1997, vol. 1, pp. 588–595 (1997)

[3] Digital Geometry tools and algorithms, http://liris.cnrs.fr/dgtal.

15. Coeurjolly, D.: Distance transformation, reverse distance transformation and discrete medial axis on toric spaces. In: IEEE International Conference on Pattern Recognition, ICPR 2008, pp. 1–4 (2008)
16. Coeurjolly, D., Montanvert, A.: Optimal separable algorithms to compute the reverse Euclidean distance transformation and discrete medial axis in arbitrary dimension. IEEE Trans. Pattern Anal. Mach. Intell. **29**(3), 437–448 (2007)
17. Coeurjolly, D., Zerarga, L.: Supercover model, digital straight line recognition and curve reconstruction on the irregular isothetic grids. Comput. Graph. **30**(1), 46–53 (2006)
18. Coléou, C., Lesaffre, B., Brzoska, J.-B., Ludwig, W., Boller, E.: Three-dimensional snow images by X-ray microtomography. Ann. Glaciol. (2001). doi:10.3189/172756401781819418
19. Cuisenaire, O.: Locally adaptable mathematical morphology using distance transformations. Pattern Recognit. **39**(3), 405–416 (2006)
20. Cuisenaire, O., Macq, B.: Fast Euclidean distance transformation by propagation using multiple neighborhoods. Comput. Vis. Image Underst. **76**(2), 163–172 (1999)
21. Culver, T., Keyser, J., Lin, M., Manocha, D.: Fast computation of generalized Voronoi diagrams using graphics hardware. In: International Conference on Computer Graphics and Interactive Techniques, pp. 277–286 (1999)
22. Danielsson, P.E.: Euclidean distance mapping. Comput. Graph. Image Process. **14**, 227–248 (1980)
23. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications. Springer, Berlin (2000)
24. Deutsch, E.S.: Thinning algorithms on rectangular, hexagonal, and triangular arrays. Commun. ACM **15**(9), 827–837 (1972)
25. Devillers, O.: Improved incremental randomized Delaunay triangulation. In: Annual ACM Symposium on Computational Geometry, pp. 106–115 (1998)
26. Enzweiler, M., Gavrila, D.M.: Monocular pedestrian detection: survey and experiments. IEEE Trans. Pattern Anal. Mach. Intell. **31**(12), 2179–2195 (2009)
27. Fabbri, R., Costa, L.D.F., Torelli, J.C., Bruno, O.M.: 2D Euclidean distance transform algorithms: a comparative survey. ACM Comput. Surv. **40**(1), 1–44 (2008)
28. Fouard, C.: Extraction de paramètres morphométriques pour l'étude du réseau microvasculaire cérébral. Ph.D. thesis, Université de Nice Sophia Antipolis (2005)
29. Fouard, C., Malandain, G.: 3-D chamfer distances and norms in anisotropic grids. Image Vis. Comput. **23**(2), 143–158 (2005)
30. Fouard, C., Malandain, G., Prohaska, S., Westerhoff, M.: Blockwise processing applied to brain microvascular network study. IEEE Trans. Med. Imaging **25**(10), 1319–1328 (2006)
31. Fouard, C., Strand, R., Borgefors, G.: Weighted distance transforms generalized to modules and their computation on point lattices. Pattern Recognit. **40**(9), 2453–2474 (2007)
32. Gotsman, C., Lindenbaum, M.: Euclidean Voronoi labelling on the multidimensional grid. Pattern Recognit. Lett. **16**, 409–415 (1995)
33. Grundmann, M., Meier, F., Essa, I.: 3D shape context and distance transform for action recognition. In: IEEE International Conference on Pattern Recognition, ICPR 2008 (2008)
34. Guan, W., Ma, S.: A list-processing approach to compute Voronoi diagrams and the euclidean distance transform. IEEE Trans. Pattern Anal. Mach. Intell. **20**(7), 757–761 (1998)
35. Hayashi, T., Nakano, K., Olariu, S.: Optimal parallel algorithms for finding proximate points, with applications. IEEE Trans. Parallel Distrib. Syst. **9**(12), 1153–1166 (1998)
36. Hirata, T.: A unified linear-time algorithm for computing distance maps. Inf. Process. Lett. **58**(3), 129–133 (1996)
37. Karavelas, M.I.: Voronoi diagrams in CGAL. In: European Workshop on Computational Geometry, EWCG 2006, pp. 229–232 (2006)
38. Klette, R., Rosenfeld, A.: Digital Geometry: Geometric Methods for Digital Picture Analysis. Morgan Kaufmann, San Mateo (2004)
39. Lau, B., Sprunk, C., Burgard, W.: Improved updating of Euclidean distance maps and Voronoi diagrams. In: IEEE International Conference on Intelligent Robots and Systems, IROS 2010 (2010)

40. Maurer, C.R., Qi, R., Raghavan, V.: A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions. IEEE Trans. Pattern Anal. Mach. Intell. **25**(2), 265–270 (2003)
41. Meijster, A., Roerdink, J.B.T.M., Hesselink, W.H.: A general algorithm for computing distance transforms in linear time. In: International Symposium on Mathematical Morphology, ISMM 2000, pp. 331–340 (2000)
42. Michikawa, T., Suzuki, H.: Sparse grid distance transforms. Graph. Models **72**(4), 35–45 (2010)
43. Montanari, U.: Continuous skeletons from digitized images. J. ACM **16**(1), 534–549 (1969)
44. Mukherjee, J., Das, P.P., Kumarb, M.A., Chatterjib, B.N.: On approximating euclidean metrics by digital distances in 2D and 3D. Pattern Recognit. Lett. **21**(6–7), 573–582 (2000)
45. Mullikin, J.C.: The vector distance transform in two and three dimensions. CVGIP, Graph. Models Image Process. **54**(6), 526–535 (1992)
46. Nagy, B.: A comparison among distances based on neighborhood sequences in regular grids. In: Image Analysis, 14th Scandinavian Conference. Lecture Notes in Computer Science, vol. 3540, pp. 1027–1036. Springer, Berlin (2005)
47. Park, S.H., Lee, S.S., Kim, J.H.: The Delaunay triangulation by grid subdivision. In: Computational Science and Its Applications, pp. 1033–1042 (2005)
48. Prevost, S., Lucas, L., Bittar, E.: Multiresolution and shape optimization of implicit skeletal model. In: Winter School in Computer Graphics and Visualization, WSCG 2001 (2001)
49. Ragnemalm, I.: The Euclidean distance transform. Ph.D. thesis, Linköping University (1993)
50. Remy, E., Thiel, E.: Optimizing 3D chamfer masks with norm constraints. In: International Workshop on Combinatorial Image Analysis, Caen, pp. 39–56 (2000)
51. Rong, G., Tan, T.-S.: Jump flooding in GPU with applications to Voronoi diagram and distance transform. In: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, SI3D 2006, p. 109 (2006)
52. Rong, G., Tan, T.-S.: Variants of jump flooding algorithm for computing discrete Voronoi diagrams. In: IEEE International Symposium on Voronoi Diagrams in Science and Engineering, ISVD 2007, pp. 176–181 (2007)
53. Rosenfeld, A., Pfaltz, J.L.: Sequential operations in digital picture processing. J. ACM **13**(4), 471–494 (1966)
54. Rosenfeld, A., Pfaltz, J.L.: Distance functions on digital pictures. Pattern Recognit. **1**(1), 33–61 (1968)
55. Saito, T., Toriwaki, J.-I.: New algorithms for Euclidean distance transformation of an n-dimensional digitized picture with applications. Pattern Recognit. **27**(11), 1551–1565 (1994)
56. Samet, H.: A quadtree medial axis transform. Commun. ACM **26**(9), 680–693 (1983)
57. Samet, H.: Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS. Addison-Wesley, Reading (1990)
58. Schouten, T., Broek, E.V.: Fast exact Euclidean distance (FEED) transformation. In: IEEE International Conference on Pattern Recognition, ICPR 2004, vol. 3, pp. 594–597 (2004)
59. Sintorn, I.-M., Borgefors, G.: Weighted distance transforms for volume images digitized in elongated voxel grids. Pattern Recognit. Lett. **25**(5), 571–580 (2004)
60. Strand, R.: Distance functions and image processing on point-lattices with focus on the 3D face- and body-centered cubic grids. Ph.D. thesis, Uppsala University (2008)
61. Sud, A., Otaduy, M.A., Manocha, D.: DiFi: fast 3D distance field computation using graphics hardware. Comput. Graph. Forum **23**(3), 557–566 (2004)
62. Tam, R., Heidrich, W.: Shape simplification based on the medial axis transform. In: IEEE Visualization, VIS 2003 (2003)
63. Tustison, N.J., Suquiera, M., Gee, J.C.: N-D linear time exact signed Euclidean distance transform. Insight J. (January–June 2006). http://hdl.handle.net/1926/171
64. Vacavant, A.: Fast distance transformation on two-dimensional irregular grids. Pattern Recognit. **43**(10), 3348–3358 (2010)
65. Vacavant, A., Coeurjolly, D.: Medial axis extraction on irregular isothetic grids. In: International Workshop on Combinatorial Image Analysis, IWCIA 2009. Progress in Combinatorial

Image Analysis, pp. 207–220. Research Publishing Service, Singapore (2009)
66. Vacavant, A., Coeurjolly, D., Tougne, L.: Distance transformation on two-dimensional irregular isothetic grids. In: International Conference on Discrete Geometry for Computer Imagery, DGCI 2008. Lecture Notes in Computer Science, vol. 4292, pp. 238–249. Springer, Berlin (2008)
67. Vacavant, A., Coeurjolly, D., Tougne, L.: A novel algorithm for distance transformation on irregular isothetic grids. In: International Conference on Discrete Geometry for Computer Imagery, DGCI 2009. Lecture Notes in Computer Science, vol. 5810, pp. 469–480. Springer, Berlin (2009)
68. Vacavant, A., Coeurjolly, D., Tougne, L.: Separable algorithms for distance transformations on irregular grids. Pattern Recognit. Lett. **32**(9), 1356–1364 (2011)
69. van den Broek, E.L., Schouten, T.E.: Distance transforms: academics versus industry. Recent Patents Comput. Sci. **4**(1), 1–15 (2011)
70. Voronoi, G.: Nouvelles applications des paramtres continus la théorie des formes quadratiques. Deuxième mémoire : Recherches sur les parallélloèdres primitifs. J. Reine Angew. Math. **134**, 198–287 (1908)
71. Vörös, J.: Low-cost implementation of distance maps for path planning using matrix quadtrees and octrees. Robot. Comput.-Integr. Manuf. **17**(13), 447–459 (2001)
72. Wong, W.T., Shih, F.Y., Su, T.F.: Thinning algorithms based on quadtree and octree representations. Inf. Sci. **176**(10), 1379–1394 (2006)

Chapter 8
Separability and Tight Enclosure of Point Sets

Peter Veelaert

Abstract In this chapter we focus on the separation and enclosure of finite sets of points. When a surface separates or encloses a point set as tightly as possible, it will touch the set in a small number of points. The results in this chapter center on how the surfaces touch the set, and on the side of a surface at which a point lies. The subject is closely related to theory of oriented matroids, where oriented matroids are used to describe hyperplane arrangements, but there are some differences in focus, as well. Oriented matroids are very useful to prove that an abstract configuration of lines and points can or cannot be realized in real space. In digital geometry, however, the realization is given, for example, as a set of edge points in a digital image. The emphasis is on finding models that explain how the digitized points and lines could arise. We give a general overview of the use of preimages (or domains) and elemental subsets in digital geometry and we also present some new results on the relation between elemental subsets and separability.

8.1 Introduction

Digital geometry arose as the study of the geometric properties of points that lie on a regular grid. The primary goal was to derive properties of digital objects, i.e., sets of grid points, that mimicked Euclidean geometry. Notable examples were digital straight lines and digital planes [17], both satisfying chord properties [23, 27], and digital disks [16]. Formal properties of sets of grid points are widely applicable in digital image processing, digital tomography, or more recently, depth images. For example, once an object has been recognized as a digitized circle, we have at hand a compact representation from which we can reconstruct the digitized version without error.

Since its initial conception the development of the digital geometry of grid points has taken several directions. From a practical viewpoint, a recurring theme is to formalize the digitization process, depending on the application. A digitization may

P. Veelaert (✉)
Ghent University, Schoonmeersstraat 52, 9000 Ghent, Belgium
e-mail: Peter.Veelaert@ugent.be

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_8,

be viewed as a rounding scheme, e.g., the grid intersect rounding scheme uses the crossings of an algebraic surface with grid lines. Obviously, the digitization scheme determines the properties of the digitized object [5]. In particular, since connectivity on a regular grid is easy to define, we can introduce schemes that preserve the connectedness of an object. Once a digitization scheme has been chosen, we can look at the inverse recognition problem. To recognize a digital object we have to show that there is a continuous surface that has given rise to the digitized object. Much effort has been spent on finding efficient algorithms for recognizing the most common digital primitives, e.g., digitized lines, circles or planes.

Since grid points have integral coordinates, a natural consequence of the digitization process is to look for properties that are related to integral solutions of algebraic equations. For example, when a digital annulus is defined as the set of integral points that satisfy $k \leq x^2 + y^2 \leq k + m$, a natural question is to look for pairs (k, m) for which the annulus is an 8-connected set. Except for the most simple cases, however, the mathematics that is involved is notoriously sophisticated.

In this work the focus is not on the digital objects themselves, but on the separation and enclosure of finite sets of points. When a surface separates or encloses a point set as tightly as possible, it will touch the set in a small number of points. The results in this chapter center on how the surfaces touch the set, and on the side of a surface at which a point lies. The subject is closely related to theory of oriented matroids [6], in particular, where oriented matroids are used to describe hyperplane arrangements. There are some differences in focus, though. Oriented matroids are very useful to prove that an abstract configuration of lines and points can or cannot be realized in real space. In digital geometry, however, the realization is given, for example, as a set of edge points in a digital image. The focus is on finding models that explain how the digitized points and lines could arise.

We will give a general overview of the use of preimages (or domains) and elemental subsets in digital geometry. Most of the material is known, but we also present some new results on the relation between elemental subsets and separability.

The chapter is structured as follows. In Sect. 8.2 we introduce domains in an affine space of separating functions. In Sect. 8.3 we consider the lattice structure of a domain. In Sect. 8.4 we look at special types of separating functions and we describe an algorithm that finds the supporting surfaces that enclose a set. Section 8.5 deals with classification of domains, and finally, in the last section we outline some open problems.

8.2 Separation Maps and Parameter Domains

In this section we introduce domains in an affine space of separating functions. Each function separates a point set by attributing function values to the points that are either positive, negative or zero. A domain encompasses all those functions that separate a given point set in the same way. One of the objectives is to enumerate

all the domains that can be defined on a finite set. For this we introduce minimal separations as a way to represent a domain as concisely as possible.

8.2.1 Affine Function Spaces

Let $h_0, h_1, \ldots, h_n$ be functions in the variables $x_1, \ldots, x_d$. We define an n-dimensional function space as the affine span of the $n+1$ functions h_i. For $p \in \mathbb{R}^d$, consider the affine space $\mathcal{F}$ of functions f_a of the form

$$f_a(p) = a_0 h_0(p) + a_1 h_1(p) + \cdots + a_n h_n(p), \quad a_i \in \mathbb{R},$$

where the parameters a_i have to satisfy the condition $a_0 + \cdots + a_n = 1$, which makes the space affine. We may rewrite this as

$$f_a(p) = \big(1 - (a_1 + \cdots + a_n)\big)h_0(p) + a_1 h_1(p) + \cdots + a_n h_n(p)$$

or

$$f_a(p) = g_0(p) + a_1 g_1(p) + \cdots + a_n g_n(p) \tag{8.1}$$

where we have set $g_0(p) = h_0(p)$, and $g_i(p) = h_i(p) - h_0(p)$, for $i > 1$. Thus, in (8.1) there is always a term, $g_0(p)$, that has no parameter. It is this form of the *affine parameterization* that we will mostly use.

To state the results that follow in a simple form, we will need subsets of points that are in general position with respect to the affine function space. For our purposes, for a subset of n points p_i to be in general position means that there is a unique surface $f_a(p) = 0$, $f_a \in \mathcal{F}$ that passes through the n points. It therefore suffices that $g_0(p_i) \neq 0$, for at least one point, and that the determinant of the matrix $(g_i(p_j))$, $1 \leq i, j \leq n$ is not zero. For example, if we define the functions as $f_a : (x, y, z) \to z - a_1 x - a_2 y - a_3$, this excludes a configuration of 3 collinear points in $\mathbb{R}^3$. When there are more than n points, they will be in general position if at least n of these points are in general position.

In what follows the functions g_i will be defined by polynomials. This is not a prerequisite, however. The only restriction is that we cannot have linear dependencies between the n functions, i.e., $g_n = \alpha_1 g_1 + \cdots + \alpha_{n-1} g_{n-1}$, not all α_i vanishing. Linear dependencies would not allow us to have n points in general position.

Example 1 In the following we will explore several affine function spaces. The functions

$$f(x, y) = y - ax - b$$

represent linear separations. Alternative forms are $f(x, y) = x - ay - b$ and $f(x, y) = ax + by - 1$. We also define circular separations

$$f(x, y) = x^2 + y^2 - 2ax - 2by + c. \tag{8.2}$$

Although this form differs from the more common form $(x-a)^2+(y-b)^2=r^2$, (8.2) has the advantage that the parameterization is affine. Circular separations can be generalized to conic separations, for example,

$$f(x,y)=x^2+dy^2-2ax-2by+c. \tag{8.3}$$

8.2.2 Separation by Sign Maps

Each function in the function space separates the points of a set by their signs. The sign map is defined as $\text{sign}(x)=1$, if $x>0$, $\text{sign}(x)=-1$, $x<0$ and $\text{sign}(0)=0$. We will often use $\{+,-,0\}$ as a shorthand for $\{+1,-1,0\}$. Let $f_a(p)$ be a function in $\mathcal{F}$ and let S be a finite subset of $\mathbb{R}^d$. All points receive a sign sign $f_a(p)$, which can be $+,-,0$. A function separates the points of S unambiguously if it only attributes the signs $+,-$.

Conversely, we can define a map on S that attributes a sign to each point in an arbitrary way. In most cases such a map will not be covered by a function.

Definition 1 Let $\text{sign}_S : S \to \{+1,-1\}$ be a map that attributes a sign to each point of S. We say that sign_S is consistent with the affine function space $\mathcal{F}$ if there is at least one function f_a in $\mathcal{F}$, such that $\text{sign}(f_a(p))=\text{sign}_S(p)$ for each point in S.

Given a sign map sign_S, we will use the shorthands $S^+ := \text{sign}_S^{-1}(+)$ and $S^- := \text{sign}_S^{-1}(-)$. Points of S^+ are sometimes denoted as p_i^+, points of S^- as p_i^-. To find out whether a sign map is consistent it is sufficient to verify the feasibility of a linear programming problem. The sign map is consistent if the linear program

$$\begin{aligned} g_0(p)+a_1g_1(p)+\cdots+a_ng_n(p)>0, &\quad p\in S^+,\\ g_0(p)+a_1g_1(p)+\cdots+a_ng_n(p)<0, &\quad p\in S^- \end{aligned} \tag{8.4}$$

in the variables a_i is feasible.

One of our objectives is to find subsets of functions that separate a set of finite points in a similar fashion, or, in other words, that attribute signs in a similar way. Therefore we want to specify a consistent sign map as concisely as possible. We are interested in attributing signs to a small subset R of S such that the sign map sign_R can be extended to a map sign_S in a unique way. A first step is to introduce surface separations, where we limit the sign map to the points that lie on a common surface $f_a(p)=0$.

Lemma 1 *Let R be a subset of S of at least n points in general position. that lie on a common surface of the form $f_a(p)=0$, $f_a\in\mathcal{F}$. Assume furthermore that R is maximal in S, that is, there are no points in $S\setminus R$ that satisfy $f_a(p)=0$. Then any sign map defined on R that is consistent with $\mathcal{F}$, can always be extended to a unique sign map defined on S.*

Proof First we note that f_a is uniquely determined because the points are in general position. The set R encompasses all the points of S that satisfy $f_a(p) = 0$. Therefore $|f_a(p)| > 0$ for all points in $S \setminus R$. Let p_m be the point in $S \setminus R$ for which $|f_a(p)|$ is minimal.

Second, since the sign map is consistent on R there is a function f_b for which sign $f_b(p)$ coincides with the sign map defined on R. Let p_M be the point in $S \setminus R$ for which $|f_b(p)|$ is maximal. Now we choose $\alpha > |f_b(p_M)|/|f_a(p_m)|$, and let

$$f_c = \alpha f_a(p) + f_b(p).$$

Then we have $\text{sign}(f_c(p)) = \text{sign}(f_a(p))$, for $p \in S \setminus R$, because α was chosen such that $|\alpha f_a(p)| > |f_b(p)|$ for all $p \in S \setminus R$. On the other hand, $\text{sign}(f_c(p)) = \text{sign}(f_b(p))$, for $p \in R$, since $f_a(p) = 0$ for the points in R.

Hence, it is always possible to attribute signs that coincide with the signs given to the points in R, and that coincide with the signs of f_a for the points not in R. Furthermore, the function f_a is uniquely determined by the points in R and the signs are consistent with a function f_c in the function space. □

Later, after defining domains, we will see that it is possible to restrict the set R even further. A domain encompasses all the functions of $\mathcal{F}$ that separate the points of S in the same way.

8.2.3 Domains of Functions

The parameters of the functions that separate a set in the same way define a subset of $\mathbb{R}^n$. Since this set of parameters is either a polyhedron or a polytope we introduce the following terminology mostly based on [30]. A polyhedron is an intersection of finitely many closes halfspaces. A polytope is a polyhedron that is bounded. A face of a polytope is an intersection of the polytope with a halfspace in which the polytope is entirely contained. Vertices are faces that are single points. Facets are faces of dimension $d - 1$. An edge is a 1-dimensional face that is the convex hull of two vertices. The vertices and edges of a polytope form an undirected graph. A polytope is *simplicial* if all its facets have the minimal possible number of d points. A polytope is *simple* if each vertex is adjacent to the minimal possible number of d facets. An n-dimensional simplex is the convex hull of $n + 1$ affinely independent vertices. A simplex is both simple and simplicial. A supporting hyperplane is a hyperplane that contains a facet.

Definition 2 Let S^+, S^- be a partitioning of S that is consistent with the function class $\mathcal{F}$. Then the polyhedron D in R^n defined by the inequalities

$$\begin{aligned} g_0(p) + a_1 g_1(p) + \cdots + a_n g_n(p) &\geq 0, \quad p \in S^+, \\ g_0(p) + a_1 g_1(p) + \cdots + a_n g_n(p) &\leq 0, \quad p \in S^- \end{aligned} \tag{8.5}$$

is called the domain of functions that separate S into S^+ and S^-.

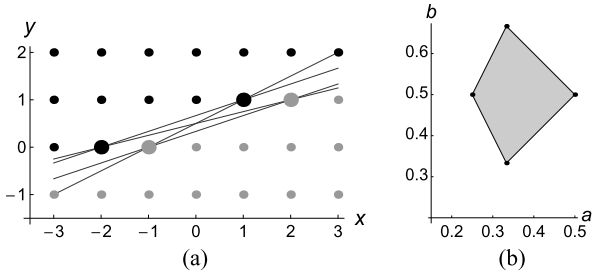


Fig. 8.1 (**a**) A separation of a set of grid points, and (**b**) the resulting domain of separating lines

A domain has also been called a preimage as it encompasses surfaces that produce the same set after digitization [2]. Each parameter point $(a_1, \ldots, a_n)$ that lies in the interior $\text{int}(D)$ of a domain defines a function $f_a(p) = g_0(p) + a_1 g_1(p) + \cdots + a_n g_n(p)$ that is consistent with the signed separation. Functions that lie on the boundary ∂D of the domain pass through one or more points of S. Therefore, they do not separate S unambiguously. Nonetheless, it is convenient to include them also. In fact, as we will see, the vertices of the domain are of special importance because they lead to a very concise way to represent a separation.

Furthermore, we explicitly exclude the case where S^+ and S^- have points in common, which would mean that (8.5) also includes pairs of inequalities that can be replaced by an equality. This would reduce the dimension of the polyhedron. For example, we could have a triangle embedded in $\mathbb{R}^3$.

Example 2 Figure 8.1(a) shows a finite set S of gridpoints. The signed separation $(-2, 0)^+$, $(1, 1)^+$ yields the domain in Fig. 8.1(b). The domain is a polygon with 4 vertices. Each of these vertices represents the parameters (a, b) of one of the 4 straight lines shown in Fig. 8.1(a). The same domain could have been specified as well by the signed pair $(-2, 0)^+$, $(2, 1)^-$, or several other pairs.

The pair $(1, 1)^+, (3, 2)^+$ does not represent a valid separation, however. Since the straight line that passes through these two points, also passes through $(-1, 0)$ and $(-3, 0)$, we must attribute signs to these points that coincide with the domain, i.e., $(-1, 0)^-$ and $(-3, 0)^-$.

Figure 8.1(b) shows only one of the many domains that can arise from a separation of S. Figure 8.2 shows more domains of separating lines for the set of gridpoints shown in Fig. 8.1(a). Only domains are shown that are bounded and that contain lines passing through the rectangle $(-1, 0)$, $(1, 0)$, $(-1, 1)$, $(1, 1)$. In this example, each domain is either a triangle or a quadrangle. This is true in general when S is a connected subset of $\mathbb{Z}^2$ [10]. When the points of S do not have to lie on a regular grid, we can have domains with an arbitrary large number of vertices.

There is a direct relation between the domains in Fig. 8.2 and the covectors of an oriented matroid. We refer to [22] for more details.

Each parameter point $a \in \text{int}(D)$ defines a function $f_a(p)$ that is consistent with the signed separation. This does not mean, however, that $f_a(p) = 0$ is always a real surface. For example, the function $x^2 + y^2 + 10$ is consistent with the separation $S^+ = \{(0, 0)\}$, $S^- = \emptyset$, but $x^2 + y^2 + 10 = 0$ does not represent a real curve in $\mathbb{R}^2$.

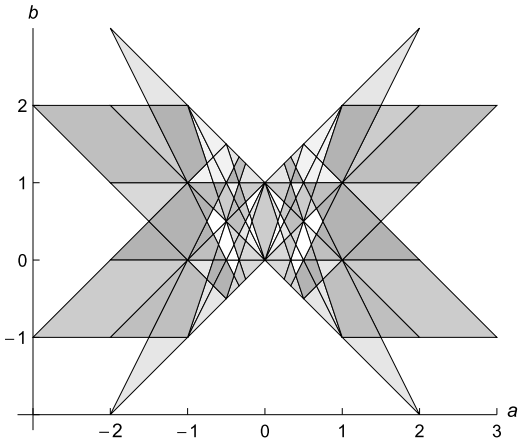


Fig. 8.2 Some of the domains that can arise from separations of the set shown in Fig. 8.1

In [29] it was shown that this anomaly does not occur as long as $S^- \neq \emptyset$. More generally, if $S^- \neq \emptyset$ and $S^+ \neq \emptyset$, there is always a real curve that separates a point $p_1 \in S^+$ from a point $p_2 \in S^-$ in the following way. Consider any continuous path $p(t)$: $t \in [0, 1]$ in $\mathbb{R}$ that starts at p_1 and ends at p_2. The restriction of a function f_a to the points on the path $p(t)$, yields a continuous function that is positive at p_1, and negative at p_2. Therefore, $f_a(p(t)) = 0$ for some point $p(t)$. This proves that we cannot go from S^+ to S^- without passing through a point of the surface $f_a(p) = 0$.

Also the points on the boundary of the domain are interesting. A parameter point $a \in \partial D$ defines a surface $f_a(p) = 0$ that will pass through one or more points of S. However, we can still specify signs for the points that satisfy $f_a(p) = 0$ in a consistent way to obtain a separation on a surface. In fact, if we specify consistent signs for the vertices of the polyhedron this will lead to minimal separations.

8.2.4 Minimal Separations

Let V_D denote the vertex set of a polyhedral domain. Each vertex corresponds to a surface separation. The vertex $v_j = (v_{1j}, \ldots, v_{nj})$ defines the surface $f_{v_j}(p) = 0$. To the points of S that lie on this surface we can attribute signs that are consistent with the separation. Each point p_i of S that lies on $f_{v_j}(p) = 0$ defines a hyperplane,

$$g_0(p_i) + a_1 g_1(p_i) + \cdots + a_n g_n(p_i) = 0$$

that passes through the vertex v_j. If the domain lies in the halfspace

$$g_0(p_i) + a_1 g_1(p_i) + \cdots + a_n g_n(p_i) \geq 0$$

we attribute the sign $+$ to p_i, else we attribute the sign $-$. In this way we can attribute signs to all the points of S that lie on the surface $f_{v_j}(p) = 0$.

However, not every point on the surface will give rise to a supporting hyperplane for the domain. Or in other words, not every hyperplane contains a facet adjacent to v_j. This allows us to reduce the surface separation to a minimal separation.

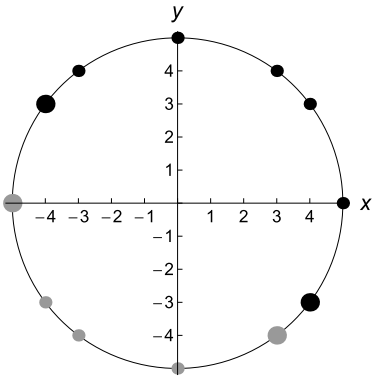


Fig. 8.3 The *big points* form a minimal separation in which *dark points* are positive, and *light points* are negative

Definition 3 Let the signed set R be a surface separation. A subset of R is called a minimal signed separation if each of its points corresponds to a supporting hyperplane of $D(R)$ and if this subset cannot be extended.

In Fig. 8.3 the dark points are positive points, and the light points are negative. All the points lie on a common circle. The set of big points, with dark as well as light points, indicates a minimal signed separation. To attribute signs to all points on the circle, and by extension also in the plane, it is sufficient to specify the signs of the points of the minimal signed separation.

Although it is minimal and cannot be further reduced, a minimal separation may have an arbitrarily large number of points.

Example 3 Figure 8.4(a) shows a minimal separation for a set S with 13 points, one point at the origin and 12 points on a common circle $x^2 + y^2 = 25$. There is a minimal separation that consists of 12 positive points: $(5, 0)^+$, $(4, 3)^+$, $(3, 4)^+$, ..., $(-4, -3)^+$. Figure 8.4(b) shows the domain, which is a 12-gon pyramid. The apex of the pyramid lies on 12 facets. The domain can be specified, however, by any of its vertices. Each of the 12 vertices at the base of the pyramid lies at the intersection of 3 half-spaces. Thus, the domain can be specified as well by the minimal separation $(0, 0)^-$, $(5, 0)^+$, $(4, 3)^+$.

Likewise, each vertex of a domain in $\mathbb{R}^n$ must lie on at least n facets. This provides a means to enumerate all possible separations of a given set.

Example 4 Consider the domains shown in Fig. 8.2, which describe domains of straight lines $y = ax + b$. Since a domain is a polygon, each vertex lies on two edges (facets). Therefore, all minimal separations consist of two points, e.g., p_i^+, p_j^+. Conversely, we can find all minimal separations by listing all pairs of points of S, and then attributing either the sign $+$ or $-$ to each point of them. A signed pair will unambiguously define a consistent separation, provided S does not contain any point that lies in between them, since the sign of this point would still be unspecified. Figure 8.2 was generated by considering only signed pairs without points in between.

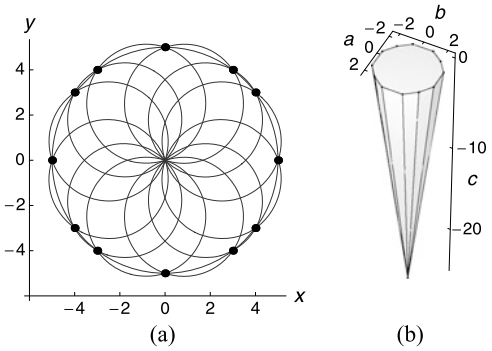


Fig. 8.4 The *black points* form a minimal separation

8.3 Lattice Structure of a Domain

The incidence relations between the faces of a domain play an important role in digital object recognition algorithms [12]. The combinatorial structure of a polytopal domain, i.e., its face lattice, is entirely determined by the meet and join relations of its faces. Each face represents a subfamily of separating functions, where a vertex corresponds to a single function, an edge to a 1-dimensional family of surfaces, and so on. The meet and join relations in the parameter space correspond to join and meet relations in the original space. For example, the join of two vertices of the parameter domain, i.e., an edge between vertices, corresponds to the meet or common intersection of a family of surfaces.

To illustrate how the combinatorial structure of the polytope is inherited by the separating functions we consider a domain of separating circles. Several authors have proposed algorithms to determine circular separability [8, 9, 11, 18, 19, 24].

Example 5 Figure 8.5 shows an example. The circular separation with $R^- = \{(1, 1), (2, 1)\}$, and $R^+ = \{(1, 0), (2, 0)\}$, induces the signs shown in Fig. 8.5(a). The gray points belong to S^- and lie inside the separating circles, the black points belong to S^+. There are infinitely many circles that separate S into S^+ and S^-. Figure 8.5(a) shows one of these circles. Figure 8.5(b) shows the polytopal domain, which has 5 vertices.

Each *vertex* of the domain corresponds to one circle. Figure 8.5(c) shows 5 circles, one for each vertex. Each circle, however, also determines a circular separation that is consistent with the partitioning S^+, S^-.

To be precise, a vertex lies at the intersection of three or more facets of the polytope, where each facet lies on a plane of the form

$$\begin{aligned} &x_1^2 + y_1^2 - 2ax_1 - 2by_1 + c = 0, \\ &\dots \\ &x_n^2 + y_n^2 - 2ax_n - 2by_n + c = 0, \end{aligned} \tag{8.6}$$

where $(x_1, y_1), \dots, (x_n, y_n)$ lie on a common circle. Each point (x_i, y_i) corresponds to one of these facets. By attributing signs to the points we can establish a consistent

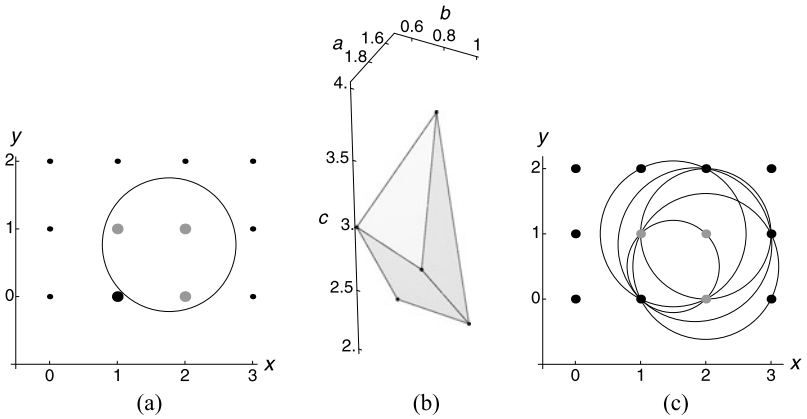


Fig. 8.5 Family of separating circles: one separating circle, the polytopal domain of parameters, and the circles that correspond to the vertices of the domain

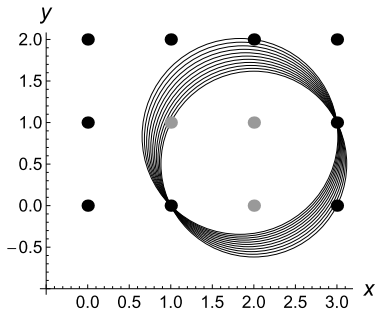


Fig. 8.6 Circles that correspond to a polytope edge. All the circles pass through two common points. The circles that correspond to the two vertices on the edge, pass through three points

circular separation. The relative position of the vertex with respect to the polytope determines the sign of each point. If the polytope is contained in the halfspace $x_1^2 + y_1^2 - 2ax_1 - 2by_1 + c \geq 0$, then (x_1, y_1) receives a positive sign, otherwise (x_1, y_1) receives a negative sign. If we limit ourselves to supporting planes at the vertex, the circular separation will be minimal.

An *edge* of the polytope corresponds to a 1-dimensional family of circles. Figure 8.6 shows the circles determined by one of the edges. All these circles have the same two points in common. The two points correspond to the two facets that are incident to the edge. Finally, each *facet* of the polytope corresponds to a 2-dimensional family of circles, which all have one point in common. This is the point (x_i, y_i) in S that determines the plane

$$x_i^2 + y_i^2 - 2ax_i - 2by_i + c = 0$$

in which the facet lies.

8.3.1 Leaning Points and Surfaces

Since the vertices and facets play an important role in the separation, we give explicit names to their counterparts.

Definition 4 Let D be a domain. Each vertex v of D gives rise to a *leaning surface* defined by $f_v(p) = 0$. Let $v_1, \ldots, v_k$, denote the k vertices of a facet of D, where we must have $k \geq n$ for a domain in $\mathbb{R}^n$. The point p that lies at the intersection of the k surfaces

$$f_{v_j}(p) = 0, \quad j = 1, \ldots, k \tag{8.7}$$

is called a *leaning point* of D.

Thus each vertex corresponds to a leaning surface. A facet corresponds to a leaning point, that is, the point that lies at the intersection of the leaning surfaces defined by the vertices of the facet. Although a facet of a domain always corresponds to a leaning point, this is not true for polytopes in general. A facet of an arbitrary polytope does not have to correspond to a leaning point. In general, given k vertices that span a facet of an arbitrary polytope, the system (8.7) will not have a real solution for p.

Example 6 Consider the space of conics defined by

$$x^2 + dy^2 - 2ax - 2by + c = 0.$$

Let P be an arbitrary polytope for the parameters a, b, c, d, and let $u_0, \ldots, u_4$ denote the coefficients of one of its supporting hyperplanes

$$u_0 + du_3 + au_1 + bu_2 + c = 0.$$

This hyperplane corresponds to a leaning point provided

$$\begin{cases} x^2 = u_0 \\ y^2 = u_3 \\ -2x = u_1 \\ -2y = u_2 \end{cases}$$

has a solution for (x, y). This is only possible if the coefficients satisfy the relations $u_1^2 = 4u_0$, and $u_2^2 = 4u_3$.

However, if we start from a separation S^+, S^- and use it to define a domain, there are no difficulties. Each facet corresponds to a real leaning point in $S^+ \cup S^-$, because all the supporting planes correspond to points of $S^+ \cup S^-$.

Example 7 Figure 8.7 shows, for a given separation S^+, S^-, the leaning points and leaning surfaces of separating circles. The separation was specified by the surface separation $(1, 1)^-$, $(2, 1)^-$, $(1, 0)^+$, $(2, 0)^-$.

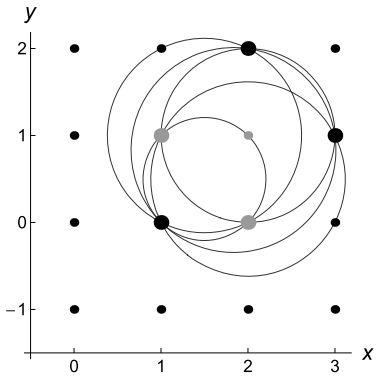


Fig. 8.7 Leaning points for circular separations

We now take the same separation S^+, S^-, but we look at a richer class of separating functions of the form

$$x^2 + dy^2 - 2ax - 2by + c,$$

whose level sets are conics. For conics the points $(1,1)$, $(2,1)$, $(1,0)$, $(2,0)$ are not in general position. There are infinitely many ellipses passing through these 4 points. Hence we cannot use this set to define a surface separation. The points $(1,0)$, $(2,0)$, $(0,1)$, $(3,1)$, $(1,2)$, $(2,2)$ are in general position. We obtain the same separation S^+, S^- if we attribute signs to these points that coincide with the original separation: $(1,0)^+$, $(2,0)^-$, $(0,1)^+$, $(3,1)^+$, $(1,2)^+$, $(2,2)^+$.

The resulting domain is a polyhedron, but not a polytope. By increasing the parameter d, we can squeeze the ellipse vertically so that it becomes more and more like a pair of horizontal parallel lines, as shown in Fig. 8.8(a). To prevent this from happening we add the point $(0, 1/2)$ to S^+, which makes the polyhedron a bounded polytope. For the same reason, we also add the point $(3/2, -1)$ to S^+, to prevent unlimited squeezing in the horizontal direction. Figure 8.8(b) shows the leaning points and leaning conics, which are all ellipses. The resulting domain has 11 vertices and 7 facets, corresponding to 11 ellipses and 7 leaning points. Since ellipses are more flexible than circles, there are more leaning points.

The ellipse configuration inherits its incidence relations from the lattice structure of the domain. Each ellipse (vertex) passes through 4 leaning points (facets). Three of the leaning points have 8 ellipses that pass through them. The domain is simple, but not simplicial.

Figure 8.9 shows the face lattice of the ellipse configuration in Fig. 8.8(b). On the top layer is the domain itself. On the second layer are 7 facets, and on the lowest, or fifth, layer 11 vertices. Each node on the lattice represents a family of ellipses. The dimension of the families decreases from top to bottom. For the domain at the top there are 4 degrees of freedom. For the second layer, the layer of facets, there are 3 degrees of freedom. A facet represents the ellipses that pass through a common leaning point. The third layer, with 2 degrees of freedom, corresponds to ellipses that pass through two common leaning points. Finally, at the fifth layer, no degree

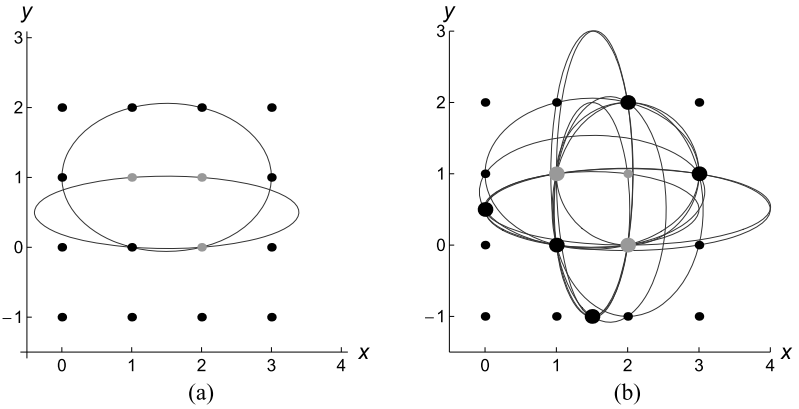


Fig. 8.8 (**a**) Squeezed ellipse; (**b**) Leaning points for elliptical separations

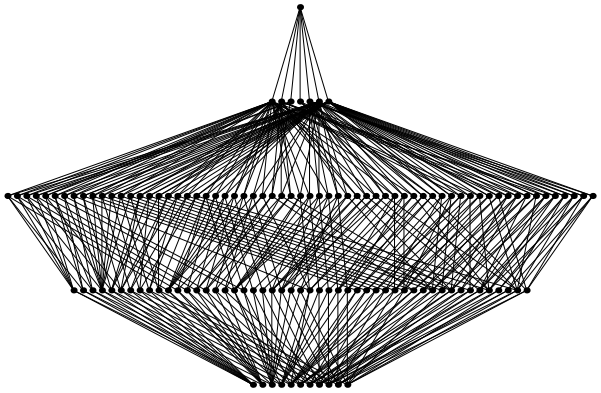


Fig. 8.9 Face lattice of ellipses

of freedom is left. Each node corresponds to a unique leaning ellipse that has to pass through 4 leaning points.

8.3.2 Lifting Map

To gain more insight in the separability problem, we will lift the points of S onto a surface in $\mathbb{R}^n$. Consider the map

$$\psi : (p_1, \ldots, p_d) \rightarrow \big(g_0(p), g_1(p), \ldots, g_n(p)\big)$$

where usually $n > d$. The map ψ lifts the points of $\mathbb{R}^d$ onto a d-dimensional surface in $\mathbb{R}^n$. We let $\psi(S^+) := T^+$, and $\psi(S^-) := T^-$. Then finding a separating function of the form (8.4) is equivalent to finding a separation of the form

$$\begin{aligned} z_0 + a_1 z_1 + \cdots + a_n z_n > 0, \quad & p \in T^+ \\ z_0 + a_1 z_1 + \cdots + a_n z_n < 0, \quad & p \in T^- \end{aligned} \tag{8.8}$$

which separates the points $p = (z_0, \ldots, z_n)$ of T^+ and T^- in $\mathbb{R}^n$. The separation problem is now the problem of separating the two sets T^+ and T^- by a hyperplane. In other words, by lifting the points onto the hypersurface $\psi(\mathbb{R}^d)$, the separation by a function

$$f_a(p) = g_0(p) + a_1 g_1(p) + \cdots + a_n g_n(p)$$

has been transformed into a separation by an affine function

$$f'_a(z) = z_0 + a_1 z_1 + \cdots + a_n z_n. \tag{8.9}$$

The map ψ does not change the nature of the separation problem. Any solution of (8.8) is also solution of (8.4) and vice versa. The domain defined by (8.8) is exactly the same as the original domain. Likewise, ψ transforms leaning points into leaning points, and leaning surfaces into leaning surfaces.

The lifting shows that a domain is always a domain of hyperplanes. For example, for any domain of separating circles in $\mathbb{R}^2$, we can always find a domain of separating planes in $\mathbb{R}^3$such that both domains coincide. The converse is not true, however. The lifted points lie on a surface, not at arbitrary positions. This restricts the shape in which a domain can occur. The following example clarifies this.

Example 8 Consider the space of affine functions

$$x^2 + y^2 - 2ax - 2by + c,$$

and the map

$$\psi : (x, y) \to \big(-2x, -2y, x^2 + y^2, 1\big).$$

If we let $(z_1, z_2, z_3, 1) = (-2x, -2y, x^2 + y^2, 1)$, then finding a separating surface amounts to finding a function

$$z_3 + a z_1 + b z_2 + c$$

that separates $\psi(S^+)$ from $\psi(S^-)$ in $\mathbb{R}^3$. Likewise, consider the space of conics

$$x^2 + d y^2 - 2ax - 2by + c,$$

and the map

$$\psi : (x, y) \to \big(-2x, -2y, x^2, y^2\big).$$

Finding a separating surface then amounts to finding a function

$$a z_1 + b z_2 + z_3 + d z_4 + c$$

that separates $\psi(S^+)$ from $\psi(S^-)$ in $\mathbb{R}^4$.

Thus circular separability in $\mathbb{R}^2$ is transformed into linear separability in $\mathbb{R}^3$ [6, 19], and conic separability is transformed into linear separability in $\mathbb{R}^4$. An immediate consequence is that circular separability can be decided in linear time [19], since a linear programming problem in 3D can be solved in linear time.

The map ψ maps $\mathbb{R}^n$ onto the paraboloid Q defined by $z_3 = (z_1/2)^2 + (z_2/2)^2$. Since ψ is one-to-one, the inverse ψ^{-1} is defined on Q. In [29] it was shown that a circular separation R^+, R^- is consistent if the sets R^+, R^- can be separated linearly in $\mathbb{R}^2$. That is, on a common circle, circular separability is equivalent to linear separability. The transformation ψ provides further insight on how circular separability is related to linear separability. Let $R = R^+ \cup R^-$ be points on a common circle. The transformation ψ maps the points of R onto a common plane H_1 in $\mathbb{R}^3$. In addition the points in $\psi(R)$ lie on the paraboloid Q. Let H_2 be a second plane that separates the sets $\psi(R^+)$, $\psi(R^-)$ in $\mathbb{R}^3$. Then the points of $\psi^{-1}(H_2 \cap Q)$ lie on a common circle which separates R^+ from R^-.

Clearly, since the lifted points of R lie on a common plane H_1 and on a paraboloid, the separability by a second plane H_2 coincides with the separability by a straight line in H_1. Furthermore, the straight line $H_1 \cap H_2$ intersects Q in two points q_1, q_2. The straight line that passes through $\psi^{-1}(q_1)$ and $\psi^{-1}(q_2)$ separates R^+ from R^- in $\mathbb{R}^2$.

Furthermore, the separation R^+, R^- is consistent for the functions (8.1) if the separation T^+, T^- is consistent with the affine space of functions (8.9). The functions (8.9) define hyperplanes. In fact, the map ψ transforms the surface $f(p) = 0$ into a hyperplane $f'(z) = 0$. Suppose the points of $R = R^+ \cup R^-$ lie on the common surface $f(p) = 0$. Then the transformation ψ maps these points onto a common hyperplane in $\mathbb{R}^n$. The separation T^+, T^- is consistent if the sets T^+ and T^- can be separated by a second hyperplane. More generally we have the following result.

Proposition 1 *Let S^+, S^- be a separation of S into positive and negative points. Then this separation is consistent with the affine functions* (8.1) *if* $\text{conv}(\psi(S+)) \cap \text{conv}(\psi(S-)) = \emptyset$.

Proof The proof is immediate. The proposition simply restates that two sets can be separated by an affine plane if their convex hulls do not intersect. □

8.3.3 Separation Extensions

Up to now we looked at how a domain separates a finite set of points. Each function in a domain separates the points of S in exactly the same way.

For the points not in S, however, the situation is different. Some points will receive the same sign from all the functions in the domain. But there are also ambiguous points, that may receive a positive sign for some functions and a negative sign for other functions.

The situation is illustrated in Fig. 8.6 which shows a family of circles whose parameters lie on an edge v_1v_2 of a polytopal domain. All pass through the two common points, that correspond to the facets that are adjacent to the edge v_1v_2. A point may always lie outside the circles, always inside, or both outside and inside some circles.

According to the following proposition, to find out in which category a point falls it is sufficient to check the vertices of a domain. The following proposition is based on one of the basic properties of polytopes.

Proposition 2 *Let S^+, S^- be a separation with polytopal domain D. Let p be a point not in $S^+ \cup S^-$. If for one of the parameter points b in the domain we have $f_b(p) \leq 0$, then there is at least one vertex v of D for which we have $f_v(p) \leq 0$. The statement is also true if we replace $\leq$ by $\geq$.*

Proof For a given point p, the half-space H of functions $f_a(p)$ for which $f_a(p) \leq 0$ is determined by

$$g_0(p) + a_1 g_1(p) + \cdots + a_n g_n(p) \leq 0.$$

Since there is at least one parameter point a in the domain for which this inequality holds, $H \cap D \neq \emptyset$. Since the domain is a convex polytope there is at least one vertex v of D for which $f_v(p) \leq 0$. □

Hence, the vertices not only specify the domain as their convex span, but in terms of separations, the functions associated with the vertices are also sufficient to determine all the unambiguous points outside S. Furthermore, a domain not only separates the finite set S, but also a large part of $\mathbb{R}^d$. The only part of space where the separation is undefined, are the points p for which $f_a(p) = 0$ for some function in the domain. Furthermore, any point with an unambiguous sign can be added to S (with the correct sign) without altering the consistency of the separation. Suppose p is a new point not in $S^+ \cup S^-$. If we add p to the separated sets, the domain D changes in the following way:

- if $f_{v_i}(p) > 0$ for all vertices v_i, then D does not change if we add p to S^+;
- if $f_{v_i}(p) < 0$ for all vertices v_i, then D does not change if we add p to S^-;
- if $f_{v_i}(p) > 0$ for some vertex v_i and $f_{v_j}(p) < 0$ for some other vertex v_j, then when we add p either to S^+ or to S^- the domain will change, but the separation will still be consistent;
- if $f_{v_i}(p) > 0$ for all vertices v_i, then the separation is no longer consistent when we add p to S^-;
- if $f_{v_i}(p) < 0$ for all vertices v_i, then the separation is no longer consistent when we add p to S^+.

8.4 Enclosure and Separation with Elemental Subsets

In this section we look at special types of separating functions. We will discover that a tight separation allows to reduce the separated sets so that their domain becomes

a simplex. Furthermore, we will see that the tight separation of two reduced sets is actually the same as their tight enclosure. For this reason tight separation and enclosure are handled together.

8.4.1 Tight Enclosure and Separation

Suppose we want to separate S^+, S^- with two non-intersecting surfaces that lie as far apart as possible. More precisely, we are interested in two surfaces $f_a(p) = \varepsilon$ and $f_a(p) = -\varepsilon$, such that

$$\begin{aligned} f_a(p) &\geq \varepsilon, \qquad p \in S^+ \\ f_a(p) &\leq -\varepsilon, \qquad p \in S^- \end{aligned} \tag{8.10}$$

where we want ε to be as large as possible. Equally interesting is a third surface $f_a(p) = 0$, which lies half-way the two sets S^+ and S^-, and which can be defined as the unique separating surface for which $\min_{p \in S^+ \cup S^-} |f_a(p)|$ is maximal.

Since the two surfaces (8.10) separate sets, there must be a link with a separating domain. Obviously, the function f_a itself lies inside the domain. Assume that one of the g_i is a constant function, e.g., $g_n = 1$, so that the functions take the form

$$g_0(x) + a_1 g_1(x) + \cdots + a_n.$$

In this case, the functions $f_a + \varepsilon$ and $f_a - \varepsilon$ also belong to the affine function space and to the domain. In fact, they lie at the boundary of the domain.

Example 9 For circles we use the function space

$$f(x, y) = x^2 + y^2 - 2ax - 2by + c.$$

Let D be a polytopal domain of separating circles. In the parameter space, we consider a straight line parallel to the c-axis passing through the domain. This line crosses the boundary of the domain in two points (a, b, c_M) and (a, b, c_m), where we choose $c_M > c_m$. The points of $S^+ \cup S^-$ are separated twice as follows:

$$\begin{aligned} x^2 + y^2 - 2ax - 2by + c_M &\geq 0, \quad (x, y) \in S^+ \\ x^2 + y^2 - 2ax - 2by + c_M &\leq 0, \quad (x, y) \in S^- \\ x^2 + y^2 - 2ax - 2by + c_m &\geq 0, \quad (x, y) \in S^+ \\ x^2 + y^2 - 2ax - 2by + c_m &\leq 0, \quad (x, y) \in S^- \end{aligned}$$

The third inequality can be rewritten as

$$x^2 + y^2 - 2ax - 2by + (c_M + c_m)/2 \geq (c_M - c_m)/2, \quad x \in S^+.$$

The second inequality can be rewritten as

$$x^2 + y^2 - 2ax - 2by + (c_M + c_m)/2 \leq -(c_M - c_m)/2, \quad x \in S^-.$$

This corresponds to a separation of the form (8.10), with $\varepsilon = (c_M - c_m)/2$. Clearly, $(c_M - c_m)/2$ attains a maximum when at least one of the crossed boundary points is a vertex of the domain. Since a vertex lies on at least $n = 3$ facets, and the other point lies on at least one facet, the surfaces

$$\begin{aligned} x^2 + y^2 - 2ax - 2by + (c_M + c_m)/2 &= (c_M - c_m)/2, \\ x^2 + y^2 - 2ax - 2by + (c_M + c_m)/2 &= -(c_M - c_m)/2, \end{aligned} \tag{8.11}$$

together pass through at least $n + 1$ points of S.

The above property is true in general. A pair of tightly separating surfaces always passes through $n + 1$ points. This brings us to the notion of an *elemental subset* which plays an important role in L_∞ fitting [26, 28].

8.4.2 Elemental Subsets

Elemental subsets are the basic building blocks for exploring separations and enclosures.

Definition 5 Let S be a finite set of points and let $\mathcal{F}$ be the affine space of functions of the form (8.1). An elemental subset E is a subset of S with $n + 1$ points, which has at least one n-point subset with its points in general position.

The primary importance of elemental subsets stems from the fact that the signs of the induced separation can be determined in an analytical way. For $n + 1$ points p_i, we define the $(n + 1) \times (n + 1)$ matrix

$$M := \begin{pmatrix} g_0(p_1) & g_1(p_1) & \dots & g_n(p_1) \\ \dots & & & \\ g_0(p_{1+n}) & g_1(p_{1+n}) & \dots & g_n(p_{1+n}) \end{pmatrix}.$$

Let C_i denote the cofactors of the first column of M. These cofactors play an important role with respect to the relative positions of the points in E and the enclosing surfaces $f(p) = \pm\varepsilon$. We have a first simple lemma.

Lemma 2 *Let E be a subset of $n + 1$ points p_i, and assume that $g_0(p_i) \neq 0$ for at least one of the points p_i. Then E is an elemental subset if and only if at least one of the cofactors C_i is non vanishing.*

Proof It suffices to note that if the cofactor $C_i \neq 0$, then the non-homogeneous system with n linear equations

$$a_1 g_1(p_j) + \dots + a_n g_n(p_j) = -g_0(p_j), \quad 1 \leq j \leq n + 1, \ j \neq i$$

has a unique solution, because its determinant is non zero. □

We introduce a residual $\varepsilon(E)$ that weighs the absolute value of the determinant of M against the sum of the absolute values of the cofactors of the first column of M,

$$\begin{aligned}\varepsilon(E) &:= \big|\det(M)\big| / \big(|C_1| + \cdots + |C_{n+1}|\big) \\ &= \big|C_1 g_0(p_1) + \cdots + C_{n+1} g_0(p_{1+n})\big| / \big(|C_1| + \cdots + |C_{n+1}|\big). \end{aligned} \tag{8.12}$$

Since at least one of the cofactors is non-vanishing, $\varepsilon(E)$ is always defined. Elemental subsets can be used for enclosing a set of points, or for separating two sets of points. In the case of enclosing points this amounts to L_∞ fitting and ε is called the residual. We have the following result [28].

Theorem 1 *Let E be an elemental subset. Then there is a unique pair of surfaces $f_a(p) = \varepsilon(E)$, $f_a(p) = -\varepsilon(E)$ such that each point of E lies on one of these surfaces. Furthermore, there is no surface $f_a(p) = 0$ for which $|f_a(p)| < \varepsilon$ for all $p \in E$.*

Thus, given E we may define E^+ as the set of points p in E for which $f_a(p) = \varepsilon(E)$, and similarly, E^- as the set of points for which $f_a(p) = -\varepsilon(E)$. The separation of E into E^+ and E^- entirely depends on the signs of the cofactors of (8.12), provided the sign of each cofactor is either $+$ or $-$, but never 0.

Theorem 2 *Let E be an elemental subset for which all the n-point subsets are in general position. Then there is a unique pair of surfaces $f_a(p) = \varepsilon(E)$, $f_a(p) = -\varepsilon(E)$ such that each point p_i in E lies on the surface*

$$f_a(p_i) = \operatorname{sign}(C_i).\operatorname{sign}(M).\varepsilon(E).$$

Proof Since the n points of each of the $n+1$ n-point subsets are in general position, none of the cofactors C_i is vanishing. Hence, $\operatorname{sign}(C_i)$ is either 1 or -1, but never equal to 0. According to Theorem 1 a point $p_i \in E$ either lies on the surface $f_a(p_i) = \varepsilon(E)$ or on the surface $f_a(p_i) = -\varepsilon(E)$. Hence we have

$$\begin{aligned} &g_0(p_1) + a_1 g_1(p_1) + \cdots + a_n g_n(p_1) = \lambda_1 \varepsilon(E) \\ &g_0(p_2) + a_1 g_1(p_2) + \cdots + a_n g_n(p_2) = \lambda_2 \varepsilon(E) \\ &\cdots \\ &g_0(p_{n+1}) + a_1 g_1(p_{n+1}) + \cdots + a_n g_n(p_{n+1}) = \lambda_{n+1} \varepsilon(E) \end{aligned} \tag{8.13}$$

where λ_i is either 1 or -1. We will show that $\lambda_i = \operatorname{sign}(C_i).\operatorname{sign}(\det(M))$.

Let C_i be the cofactors of the first column of the matrix M. Then,

$$\sum_{1 \le i \le n+1} g_0(p_i) C_i = \det M,$$

but also $\sum_{1 \le i \le n+1} g_j(p_i) C_i = 0$ for $j = 1, \ldots, n$, because these sums represent determinants of matrices with two identical columns. If we multiply the i-th equality of (8.13) by the cofactor C_i, and add all the left and right hand sides, we find

$$\sum_{1 \le i \le n+1} g_0(p_i) C_i = \sum_{1 \le i \le n+1} \varepsilon(E) \lambda_i C_i,$$

which can be rewritten as

$$\varepsilon(E) = \det(M) / \sum_{1 \le i \le n+1} \lambda_i C_i. \tag{8.14}$$

However, according to Theorem 2 we also have

$$\varepsilon(E) = \big|\det(M)\big| / \sum_{1 \le i \le n+1} |C_i|.$$

Hence in (8.14) the λ_i must be chosen such that $|\sum_{1 \le i \le n+1} \lambda_i C_i|$ is maximal, and $\varepsilon(E)$ is positive. This will be true if $\lambda_i = \operatorname{sign}(C_i) \operatorname{sign}(\det(M))$. □

Because of this result it is easy to find the best fit of an elemental subset. The parameters of the best fit satisfy the linear system

$$g_0(p_i) + a_1 g(p_i) + \cdots + a_n g_n(p_i) = \operatorname{sign}(C_i) \operatorname{sign}\big(\det(M)\big)\varepsilon(E),$$
$$i = 1, \ldots, n+1.$$

This system has $n+1$ equations in n variables. If $\varepsilon(E)$ is defined as in (8.12), the system always has a solution, which can be found by discarding one of the $n+1$ equations, no matter which one. Alternatively, one may consider ε as the $(n+1)$ variable. Then, after solving the system for $a_1, \ldots, a_n, \varepsilon$, one finds for ε the value defined by (8.12).

8.4.3 Tight Enclosure of Large Set

Elemental subsets allow us to construct a global separation or enclosure from elemental separations or enclosures [28]. Let S be a finite subset of points that contains at least one elemental subset. The size of S may be much larger than $n+1$. We define the residual of S as

$$\varepsilon(S) := \max_{E \subseteq S} \varepsilon(E),$$

where the maximum of $\varepsilon(E)$ is taken over all elemental subsets E in S. The following result was proven in [28].

Theorem 3 *Let S be a set that contains at least one elemental subset E. Then there is a unique surface $f_a(p) = 0$ such that $|f_a(p)| \le \varepsilon(S)$, $p \in S$.*

This theorem has an immediate corollary.

Corollary 1 *Let S be a set that contains at least one elemental subset E_1. Let $f_1(p) = 0$ be the unique surface such that $f_1(p) = \varepsilon(E_1)$ and $f_1(p) = -\varepsilon(E_1)$ contain all the points of E_1. If there is a point $p_2 \in S$ for which $|f_1(p_2)| > \varepsilon(E_1)$ then the set $E_1 \cup \{p_2\}$ contains at least one elemental subset E_2 for which $\varepsilon(E_2) > \varepsilon(E_1)$.*

Proof Let E_1 be an elemental subset in S for which there is a point p_2 with $|f(p_2)| > \varepsilon(E_1)$. According to Theorem 3 there is a surface f_2 such that

$$\big|f_2(p)\big| \le \varepsilon\big(E_1 \cup \{p_2\}\big), \quad p \in E_1 \cup \{p_2\}. \tag{8.15}$$

At the same time,

$$\big|f_1(p)\big| \le \varepsilon(E_1), \quad p \in E_1. \tag{8.16}$$

Since p_2 satisfies (8.15), but not (8.16), we must have $\varepsilon(E_1 \cup \{p_2\}) > \varepsilon(E_1)$. Since $\varepsilon(E_1 \cup \{p_2\} = \max_{E \subset (E_1 \cup p_2)} \varepsilon(E)$, there must be at least one subset E_2 in $E_1 \cup \{p_2\}$ such that $\varepsilon(E_2) > \varepsilon(E_1)$. □

Corollary 1 leads to an efficient algorithm that finds the supporting surfaces that enclose a large set S. The algorithm works for any space of affine functions. The basic idea is to construct a sequence of elemental subsets $E_1, E_2, \ldots$ with increasing costs $\varepsilon(E_1) < \varepsilon(E_2) < \cdots$. Eventually we must find an E_n of maximal cost, whose supporting surfaces $f_a(p) = \pm\varepsilon(E_n)$ enclose the entire set.

Algorithm
Input: A finite set S.
Output: An elemental subset E_n such that $\varepsilon(E_n) = \varepsilon(S)$.
1. Select an arbitrary initial elemental subset E_1 and compute $\varepsilon(E_1)$;
2. Compute the best fit f_a of E_i;
3. Process all points of S until a point p is found at a distance further than $\varepsilon(E_i)$ from the best fit;
4. If no such point is found, return the current E_i;
5. Replace E_i by an elemental subset E_{i+1} in $E_i \cup \{p\}$ for which $\varepsilon(E_{i+1}) > \varepsilon(E_i)$. According to Corollary 1 there is at least one such subset in $E_i \cup \{p\}$.
6. Proceed with step 2.

8.4.4 Tight Separation of Sets

In Sect. 8.2 we considered the domain of all functions that separate two sets. Among these functions, however, there is a separating function that maximizes the algebraic distance to both sets. Let S^+, S^- be a consistent separation. We define $\varepsilon(S^+, S^-)$ as the maximal value of ε for which the system

$$\begin{aligned} g_0(p) + a_1 g(p) + \cdots + a_n g_n(p) &\ge \varepsilon, \qquad p \in S^+ \\ g_0(p) + a_1 g(p) + \cdots + a_n g_n(p) &\le -\varepsilon, \qquad p \in S^- \end{aligned} \tag{8.17}$$

is still feasible. Again elemental subsets play a crucial part in this optimization problem.

Proposition 3 *Let S^+, S^- be a consistent separation with a polytopal domain. Then there is an elemental subset E in $S^+ \cup S^-$ for which $E \cap S^+ \neq \emptyset$, $E \cap S^- \neq \emptyset$ and $\varepsilon(E \cap S^+, E \cap S^-) = \varepsilon(S^+, S^-)$.*

Proof Consider the linear programming problem in the variables $a_1, \ldots, a_n, \varepsilon$ where we want to maximize the value of ε subject to the conditions (8.17). Since the $a_1, \ldots, a_n$ that satisfy (8.17) always belong to the domain of the separation, the polyhedron defined by the linear programming problem is also a polytope. The maximum is always attained in a vertex $v_1, \ldots, v_n, \varepsilon_m$ of the polytope, which lies on at least $n+1$ facets. Hence there must be $n+1$ points p in $S^+ \cup S^-$ for which

$$g_0(p) + v_1 g(p) + \cdots + v_n g_n(p) = \varepsilon_m.$$

These points form an elemental subset E for which $\varepsilon(E \cap S^+, E \cap S^-) = \varepsilon(S^+, S^-)$. □

Given S^+, S^- for each elemental subset E in $S^+ \cup S^-$ we let E^+, E^- denote the separation of E induced by S^+, S^-. That is, $E^+ = E \cap S^+$ and $E^- = E \cap S^-$. Clearly, if S^+, S^- then E^+, E^- is also consistent. The following theorem is the dual of Theorem 3.

Theorem 4 *Let S^+, S^- be a consistent separation. Then*

$$\varepsilon\big(S^+, S^-\big) = \min_{E \in S} \varepsilon\big(E^+, E^-\big).$$

Proof The proof is based on Helly's Theorem which states that in $\mathbb{R}^n$ the intersection of a finite collection of convex compact sets is non-empty if and only if the intersection of every $n+1$ of these sets is non-empty. Consider the system (8.17) where we want to find the maximal value for which this system still has a solution. Due to Helly's Theorem, the system is feasible if each of its subsystems with $n+1$ inequalities is feasible.

However, each subsystem with $n+1$ inequalities corresponds to a separation E^+, E^- of an elemental subset. The subsystem will have a solution provided $\varepsilon \leq \varepsilon(E^+, E^-)$. Since this must hold for any subsystem, we must have $\varepsilon(E^+, E^-) \leq \min_E \varepsilon(E)$, where the minimum is taken over elemental subsets E in $S^+ \cup S^-$. □

One of the pleasant properties of an elemental subset is that we can compute the residual cost $\varepsilon(E)$ of its tight enclosure analytically. For tight separations there is a similar result. It comes at no surprise, however, that to compute $\varepsilon(E^+, E^-)$ we also need to specify at which side a point lies. In this sense, the computation of $\varepsilon(E^+, E^-)$ is a refinement of the computation of $\varepsilon(E^+ \cup E^-)$.

Given a separation E^+, E^-, for each p_i let $\operatorname{sign}(p_i)$ denote the sign of p_i as attributed by the separation.

Theorem 5 *Let E^+, E^- be a consistent separation of an elemental subset E. Then*

$$\varepsilon\big(E^+, E^-\big) = \det(M) \Big/ \sum_{1 \leq i \leq n+1} \operatorname{sign}(p_i) C_i. \tag{8.18}$$

Proof The proof uses the same technique as the proof of Theorem 2. The points of E^+ lie on the surface $f_a(p) = \varepsilon$, and the points of E^- lie on the surface $f_a(p) = -\varepsilon$. Hence we have the following system of $n+1$ equations:

$$g_0(p_1) + a_1 g_1(p_1) + \cdots + a_n g_n(p_1) = \text{sign}(p_i).\varepsilon\big(E^+, E^-\big), \quad p \in E. \tag{8.19}$$

After multiplication with the cofactors C_i we find

$$\sum_{1 \le i \le n+1} g_0(p_i) C_i = \sum_{1 \le i \le n+1} \text{sign}(p_i) C_i \varepsilon\big(E^+, E^-\big),$$

which yields

$$\varepsilon\big(E^+, E^-\big) = \det(M) / \sum_{1 \le i \le n+1} \text{sign}(p_i) C_i . \qquad \square$$

The proof also shows that $\varepsilon(E^+ \cup E^-) = \min \varepsilon(E^+, E^-)$, where the minimum is taken over all possible ways in which we can separate the elemental subset E into E^+ and E^-. Hence, if try to separate the set E in all possible ways, the tight separation for which the residual $\varepsilon(E^+, E^-)$ becomes minimal, coincides with a tight enclosure of the elemental subset.

Example 10 Let S^+, S^- be a signed separation as shown in Fig. 8.10. The value of $\varepsilon(S^+, S^-)$ can be computed in two ways. First, the vertices of the domain are: $(3/2, 1/2, 2)$, $(3/2, 1, 2)$, $(11/6, 5/6, 8/3)$, $(2, 1/2, 3)$, $(2, 1, 4)$. By constructing a line parallel to the c-axis through each of the vertices, and by computing the intersections with opposing facets, we find that the maximal value of $c_M - c_m$ is $2/3$.

The same result can be found with elementary subsets. For circles the matrix M takes the form

$$M = \left(\begin{array}{c|ccc} x_1^2 + y_1^2 & -2x_1 & -2y_1 & 1 \\ x_2^2 + y_2^2 & -2x_2 & -2y_2 & 1 \\ x_3^2 + y_3^2 & -2x_3 & -2y_3 & 1 \\ x_4^2 + y_4^2 & -2x_4 & -2y_4 & 1 \end{array} \right)$$

for an elemental subset $(x_1, y_1), \ldots, (x_4, y_4)$. The set $(1, 0)^+$, $(2, 2)^+$, $(3, 1)^+$, $(1, 1)^-$ is an elemental subset whose supporting surfaces separate the set S into S^+, S^-. For this elemental subset (8.18) yields $\varepsilon = 1/3$. Figure 8.10 shows the pair of supporting circles of the elemental subset. Also shown is the circle that lies half-way the separating circles.

8.5 Classification of Domains

Since a bounded domain is a convex polytope, there is a wide range of results on polytopes that are immediately relevant for domains and configurations of separating surfaces. Clear examples are the lower and upper bound theorems which give

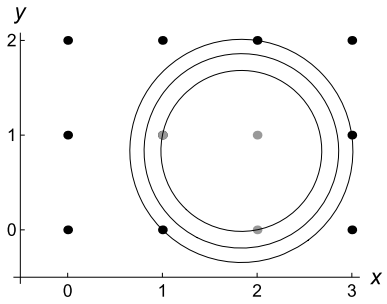


Fig. 8.10 The pair of supporting circles that separate S^+ from S^-

bounds for the number k-faces for a polytope with n facets [7, 30]. In this section we will only touch the most straightforward links between polytopes and surface configurations. In particular, we look at what kind of lattice structure a domain can have. The lattice of a domain corresponds directly to a lattice of separating surfaces and their intersections.

8.5.1 Simplices

Two polytopes are called combinatorially equivalent if there is an isomorphism that preserves the lattice structure [30]. A simplex is the most simple kind of polytope. In R^n space, a polytope with $n+1$ affinely independent vertices is a simplex. A consistent separation has at least $n+1$ leaning points and $n+1$ leaning surfaces. Therefore, a simplex represents the most basic way to separate two point sets, with the minimal number of leaning points and surfaces. Figure 8.11 shows three configurations where the domain is a simplex. In Fig. 8.11(a) the lines that separate the points have a triangular domain. There are 3 leaning lines (vertices of the triangle), and three leaning points (facets of the triangle). In Fig. 8.11(b) the circles form tetrahedron or 4-simplex. There are 4 leaning circles, and 4 leaning points. On each leaning circle there are exactly 3 leaning points, and through each leaning point pass precisely 3 circles. Figure 8.11(c) also leads to a simplex domain for circles, although the configuration is different from that in Fig. 8.11(b). In Fig. 8.11(b) the circles separate one internal point from three external points. In Fig. 8.11(c) two internal points are separated from two external points.

Theorem 6 *Let E be an elemental subset with $\varepsilon(E) > 0$, and let E^+, E^- be the separation induced by the tight enclosure of E. Then the domain of E^+, E^- is a simplex.*

Proof Each point of E defines a half-space, and the intersection of the $n+1$ half-spaces is a polyhedron P. Since there is a tight enclosure $f_a(p) = \pm\varepsilon(E)$, with $\varepsilon(E) > 0$, the interior of P is non-empty. □

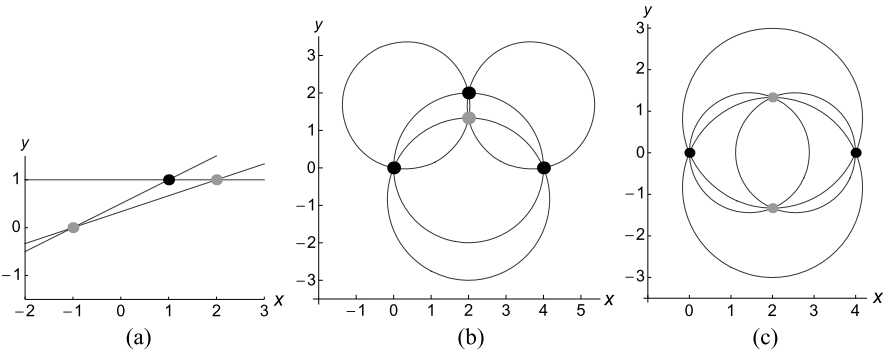


Fig. 8.11 Point configurations that lead to simplex domains

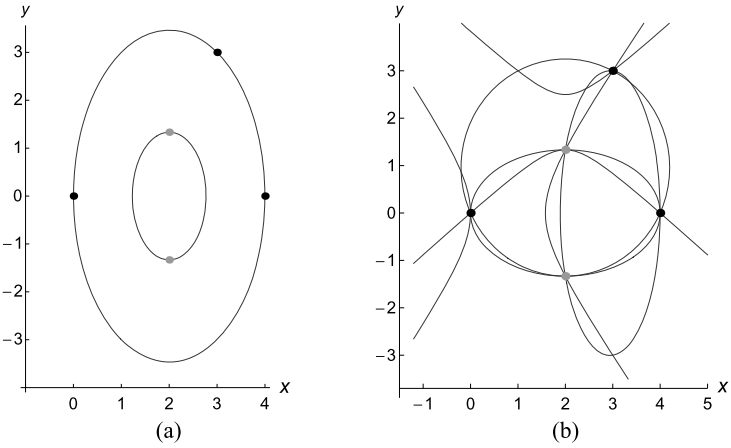


Fig. 8.12 A tight enclosure of 5 arbitrarily chosen points yields a simplex of conics

Therefore, to find a simplex domain, we can take an arbitrary subset of $n+1$ points in general position, compute a tight enclosure to find a consistent separation, which then leads to a simplex domain. Figure 8.12(a) shows 5 points and their tight enclosure by conics. The signs and separation induced by the enclosure yields a 4-simplex in the parameter space. Figure 8.12(b) shows the 5 leaning points, which coincide with the original points, and the 5 leaning conics. On each conic there are 4 points and through each point 4 conics pass.

8.5.2 More Complex Domains

Although the classification of polytopes in $\mathbb{R}^n$ is still an ongoing research topic, the classification of convex polytopes with few vertices, i.e., $n+2$ or $n+3$ vertices is known [14, 15]. We start with a general observation.

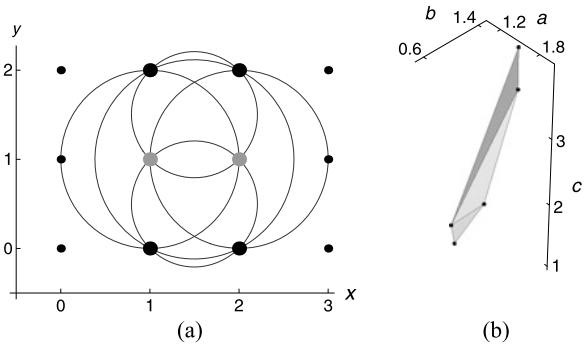


Fig. 8.13 Bipyramid as a domain and the leaning points and leaning surfaces

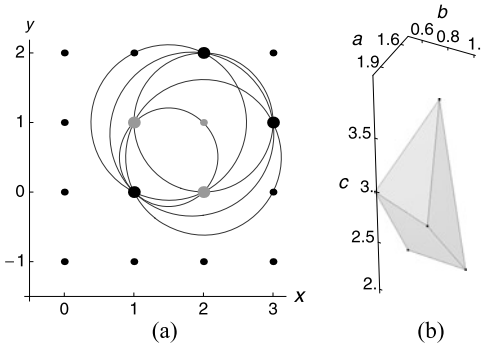


Fig. 8.14 Square pyramid as a domain of separating circles and their leaning points and leaning surfaces

Proposition 4 *Let S^+, S^- be a separation such that there is no subset $S^+ \cup S^-$ with $n+1$ points that lie on a common surface $f(p) = 0$. Then the domain is a simple polytope.*

Proof Since each leaning surface has at most n points on it, each vertex is incident to the minimal number of n facets. □

There are only two kinds of 3-polytopes with 5 vertices, the bipyramid over a triangle, and the square pyramid. In Fig. 8.13 the domain is a bipyramid over a triangle. The domain has 5 vertices and 6 facets. Hence there are 5 leaning circles and 6 leaning points. The bipyramid is a simplicial polytope since each facet has 3 vertices. Because of this, there are three leaning circles passing through each leaning point. A bipyramid is not a simple polytope. There are 3 vertices that are each adjacent to 4 facets. These vertices correspond to the 3 circles in Fig. 8.13(a) that have 4 leaning points.

In Fig. 8.14 the domain is a square pyramid. It is not simplicial and also not a simple polytope. There are 5 leaning points, 3 positive leaning points and 2 negative leaning points. The leaning point $(1, 0)$ corresponds to the square of the pyramid. There are four leaning circles that pass through it. The circle that passes through 4 distinct leaning points corresponds to the apex of the pyramid.

The number of types increases rapidly with the number of vertices. There are 7 kinds of 3-polytopes with 6 vertices [14, 15]. Two of these 7 are simplicial polytopes.

8.6 Concluding Remarks and Open Problems

The careful analysis of the structure of domains has led to efficient algorithms for digital object recognition. Since digital object recognition itself is in essence an LP problem, the main focus was on problems that go one step further and where one has to split an ordered set into segments that represent multiple objects, for example, circular arcs [8, 24].

An even greater challenge, however, may lie in problems that are related to convex programming. We list two of them. The enclosure of finite sets of data points is equivalent to L_∞ fitting. Here we addressed the basic problem of enclosing the entire data set. In practice, however, we need more robust fitting methods that can cope with outliers, e.g., in geometric vision [1]. It is known that when outliers are present, the elemental subset with the largest fitting cost contains at least one of them. It is difficult, however, to identify which of the $n+1$ points in the elemental subset is an outlier. Several approaches have been used to remove the most likely outliers while preserving the most likely inliers [1, 20, 25]. From the viewpoint of digital geometry this outlier removal problem can be formulated in two distinct ways: Given a set S find a subset T of size m such that $\varepsilon(T)$ is as small as possible. Here we assume that $n-m$ of the points are outliers. Or alternatively, given a set S, and a threshold δ find a subset T that satisfies $\delta \geq \varepsilon(T)$ where T is as large as possible. Here we assume that the outliers are those points that would make the residual larger than a given threshold. Although for both problems an optimal solution can be found by enumerating all elemental subsets, it is still an open problem to find an algorithm that makes optimal use of the information provided by tight enclosures. An efficient algorithm should evaluate as few elemental subsets as possible to determine the most likely outliers.

A similar question occurs in classification problems where we want to find a minimal set of surfaces that separate feature points in distinct classes, as is done by support vector machines. In its most basic form we have to find a surface that separates two sets S^+ and S^-, but where the given separation is in general not consistent. Here the problem can be formulated as follows. Find two subsets $T^+ \subseteq S^+$ and $T^- \subseteq S^-$ such that the separation T^+ and T^- is consistent with the affine function space, and where $T^+ \cup T^-$ is as large as possible.

Finally, one of the corner stones that still seems to be missing in digital geometry is a better integration of the analytical results on digitized surfaces with digital topology. The separation and enclosure results presented in this chapter are closely related to sets of integral points between level surfaces. Subsets of the form $S = \{p \in \mathbb{Z}^d\colon\ -\varepsilon \leq f_a(p) \leq \varepsilon\}$ have been called digital layers, digital level sets [13, 21], or also discrete analytical surfaces. It is still a major problem to find a

value for ε so that a digital layer is connected as a set. For hyperplanes and hyperspheres this question has been examined in some detail [3–5]. For the more general case, only a few limited results are known [13].

References

1. Agarwal, S., Snavely, N., Seitz, S.M.: Fast algorithms for l infinity problems in multiview geometry. In: CVPR. IEEE Comput. Soc., Los Alamitos (2008)
2. Anderson, T.A., Kim, C.E.: Representation of digital line segments and their preimages. Comput. Vis. Graph. Image Process. **30**(3), 279–288 (1985)
3. Andres, E., Acharya, R., Sibata, C.: Discrete analytical hyperplanes. Graph. Models Image Process. **59**(5), 302–309 (1997)
4. Andres, E., Jacob, M.-A.: The discrete analytical hyperspheres. IEEE Trans. Vis. Comput. Graph. **3**(1), 75–86 (1997)
5. Andres, E., Roussillon, T.: Analytical description of digital circles. In: Debled-Rennesson, I., et al. (eds.) Discrete Geometry for Computer Imagery. LNCS, vol. 6607, pp. 235–246. Springer, Berlin (2011)
6. Bjorner, A., Las Vergnas, M., Sturmfels, B., White, N., Ziegler, G.: Oriented Matroids. Cambridge University Press, Cambridge (1993)
7. Bronsted, A.: An Introduction to Convex Polytopes. Springer, Berlin (1983)
8. Coeurjolly, D., Gerard, Y., Reveilles, J.-P., Tougne, L.: An elementary algorithm for digital arc segmentation. Discrete Appl. Math. **139**, 31–50 (2004)
9. Damaschke, P.: The linear time recognition of digital arcs. Pattern Recognit. Lett. **16**, 543–548 (1995)
10. Dorst, L., Smeulders, A.: Discrete representation of straight lines. IEEE Trans. Pattern Anal. Mach. Intell. **6**, 450–462 (1984)
11. Fisk, S.: Separating point sets by circles, and the recognition of digital disks. IEEE Trans. Pattern Anal. Mach. Intell. **8**(4), 554–556 (1986)
12. Gérard, Y., Coeurjolly, D., Feschet, F.: Gift-wrapping based preimage computation algorithm. Pattern Recognit. **42**(10), 2255–2264 (2009)
13. Gérard, Y., Provot, L., Feschet, F.: Introduction to digital level layers. In: Debled-Rennesson, I., et al. (eds.) Discrete Geometry for Computer Imagery. LNCS, vol. 6607, pp. 83–94. Springer, Berlin (2011)
14. Grünbaum, B.: Polytopes, graphs, and complexes. Bull. Am. Math. Soc. **76**, 1131–1201 (1970)
15. Grünbaum, B.: Convex Polytopes, vol. 221. Springer, Berlin (2003)
16. Kim, C.E.: Digital disks. IEEE Trans. Pattern Anal. Mach. Intell. **6**, 372–374 (1984)
17. Kim, C.E.: Three-dimensional digital planes. IEEE Trans. Pattern Anal. Mach. Intell. **6**, 639–645 (1984)
18. Kim, C.E., Anderson, T.A.: Digital disks and a digital compactness measure. In: STOC, pp. 117–124. ACM, New York (1984)
19. Koraraju, S.R., Meggido, N., O'Rourke, J.: Computing circular separability. Discrete Combin. Geom. **1**, 105–113 (1986)
20. Lee, H., Seo, Y., Lee, S.W.: Removing outliers by minimizing the sum of infeasibilities. Image Vis. Comput. **28**(6), 881–889 (2010)
21. Provot, L., Gérard, Y.: Recognition of digital hyperplanes and level layers with forbidden points. In: Aggarwal, J.K., Barneva, R.P., Brimkov, V.E., Koroutchev, K., Korutcheva, E. (eds.) Combinatorial Image Analysis. LNCS, vol. 6636, pp. 144–156. Springer, Berlin (2011)
22. Richter-Gebert, J., Ziegler, G.M.: Oriented matroids. In: Handbook of Discrete and Computational Geometry, pp. 111–132. CRC Press, Boca Raton (1997)
23. Rosenfeld, A.: Digital straight line segments. IEEE Trans. Comput. **23**, 1264–1269 (1974)

24. Roussillon, T., Tougne, L., Sivignon, I.: On three constrained versions of the digital circular arc recognition problem. In: Brlek, S., Reutenauer, C., Provençal, X. (eds.) Discrete Geometry for Computer Imagery. LNCS, vol. 5810, pp. 34–45. Springer, Berlin (2009)
25. Seo, Y., Lee, H., Lee, S.W.: Outlier removal by convex optimization for l-infinity approaches. In: Wada, T., Huang, F., Lin, S. (eds.) PSIVT. LNCS, vol. 5414, pp. 203–214. Springer, Berlin (2009)
26. Stromberg, A.: Computing the exact least median of squares estimate and stability diagnostics in multiple linear regression. SIAM J. Sci. Comput. **14**, 1289–1299 (1993)
27. Veelaert, P.: On the flatness of digital hyperplanes. J. Math. Imaging Vis. **3**, 205–221 (1993)
28. Veelaert, P.: Constructive fitting and extraction of geometric primitives. Graph. Models Image Process. **59**(4), 233–251 (1997)
29. Veelaert, P.: Distance between separating circles and points. In: Debled-Rennesson, I., et al. (eds.) Discrete Geometry for Computer Imagery. LNCS, vol. 6607, pp. 346–357. Springer, Berlin (2011)
30. Ziegler, G.M.: Lectures on Polytopes. Graduate Texts in Mathematics, vol. 152. Springer, New York (1995)

Chapter 9
Digital Straightness, Circularity, and Their Applications to Image Analysis

Partha Bhowmick and Bhargab B. Bhattacharya

Abstract This chapter contains some theoretical properties of digital straightness and digital circularity, which are helpful in designing algorithms related to image analysis. These properties are obtained mainly from word-theoretic and number-theoretic analysis. Existing techniques on straight line recognition, circular arc recognition, vectorization, etc. have been discussed along with their historical connections. Some salient points that discriminate digital geometry from real geometry have also been mentioned. Relevant experimental results have been given to demonstrate the elegance of digital-geometric techniques in performing desired tasks in the digital plane. Some open problems have been given at the end to point out the challenges and prospects of digital straightness and circularity in image analysis.

9.1 Introduction

Although Euclidean geometry is often sought for—quite more than any other non-Euclidean geometry, however, with the proliferating digitization of graphical objects and visual imageries, fresh analytical studies and experimentation with geometric primitives such as straight lines, circles, curves, and planes have become indispensable for efficient real-world applications. Hence, theoretical studies on these primitives in the digital space have been an active subject of research since 1960s [20, 21, 38, 45, 53, 60, 61, 72, 90, 114, 122]. In later years, several works have come up on characterization and algorithms for generation and recognition of these geometric primitives [15, 18, 22, 25, 33, 42, 50, 73, 84, 85, 98, 99, 141, 148, 150, 153].

P. Bhowmick (✉)
Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur, India
e-mail: bhowmick@gmail.com

P. Browmick
e-mail: pb@cse.iitkgp.ernet.in

B.B. Bhattacharya
Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India
e-mail: bhargab@isical.ac.in

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_9, © Springer Science+Business Media Dordrecht 2012

The earlier works were mostly based on the notions borrowed from real/Euclidean geometry, whereas the more recent works are based on the new paradigms of *digital geometry* [6, 24, 26, 32, 51, 82, 84, 132], *digital topology* [75, 77, 83, 92, 130, 131], *computational imaging* [5, 8, 121], *combinatorial image analysis* [28–30], and *theory of words and numbers* [7, 23, 86].

Among these geometric primitives, digital straight line segments (DSS) and digital circles have drawn special attention for their challenging nature from the viewpoint of theoretical formulation, and also for their potential applications to image analysis and computer graphics. Several intriguing problems and properties related to DSS and DSL (digital straight line/ray) have been studied [20, 115, 124]. Many attributes of DSS can be interpreted in terms of continued fractions [88, 103, 144]. The most fundamental problem, which is highly relevant to pattern recognition in general, and to curve approximation in particular, is to ascertain whether or not a given digital curve segment S is a DSS. Many solutions to this problem have been reported in the literature [43, 49, 50, 89, 102, 138]. With the advent of digitization in general, and vectorization in particular, the problem on characterization and recognition is also of higher relevance today for digital circles and circular arcs [46, 52, 70, 139, 140]. Some of the contemporary works on circle detection may be seen in [12, 93, 118, 128, 129].

The chapter is organized as follows. In Sect. 9.2, we indicate the notional difference of straightness in $\mathbb{Z}^2$ with that in $\mathbb{R}^2$, and explain the digital-geometric properties of straightness. The concept of approximate straightness and the algorithm to extract approximate straight line segments are also discussed in this section. Section 9.3 contains the digital-geometric properties of circularity and a couple of algorithms to determine whether a digital curve segment is digitally circular. The usage of approximate straightness in polygonal approximation of a set of digital curve segments is explained in Sect. 9.4. Related algorithms and some experimental results have been furnished in this section to demonstrate the elegance of digital-geometric algorithms. Section 9.5 presents the usefulness of chord property and sagitta property in circular arc segmentation, along with experimental results. With some open problems given in Sect. 9.6, we conclude this chapter in Sect. 9.7.

9.2 Digital Straightness

Since a straight line—whether Euclidean or digital—essentially consists of points, a digital line (or any other digital curve) is a sequence of digital points, which satisfies certain straightness properties in a digital sense. Similarities may be found in their constitutions, but differences lie in their very definitions. In the *Elements*, Euclid defines a (real) point as that of which there is no part [19]. In digital geometry, an additional criterion is that the coordinates of a (digital) point can only be integers. Thus, the similarity between a real point and a digital point lies in their very characteristic of having "no part", and the dissimilarity is in their "coordinate constraints". Similarly, for a real and a digital straight line, the similarity is in the fact that a

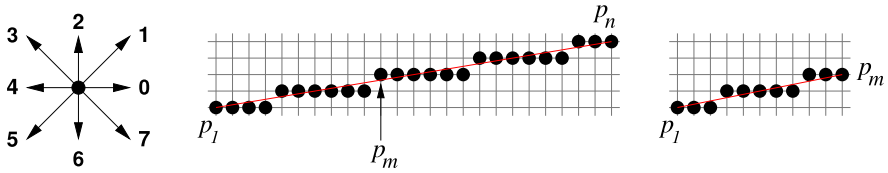


Fig. 9.1 *Left*: Chain codes in 8N. *Right*: DSS(p_1, p_n) is cut at p_m; when the real line segment p_1p_m is digitized to get DSS(p_1, p_m), it is not the exact (ordered) subset of points from p_1 to p_m in the original DSS

straight line is one that lies evenly with points on itself. Interestingly, the property of evenness of points lying on a line was stated as the definition of a real straight line by Euclid, which was reiterated by both Freeman and Rosenfeld [61, 84, 122] in 1960s–70s in the context of digital straight lines. The difference is, Euclid stated the "evenness" as a definition; whereas, the concept of "evenness" was formalized and proved in [122], which shows that "evenness" is a necessary condition in order that a digital curve segment is digitally straight.

Another important and not-so-trivial characteristic that finds some similarity and also some difference between a real straight line and a digital straight line is as follows. One would expect that, if a digital straight line segment (DSS) is cut into two parts, then each (i.e., sub-DSS) of them would still be digitally straight (and a DSS, thereof). This, in fact, is true for a real straight line segment in the Euclidean geometry, and also in accordance with our notional intuition. However, from the two subsequences of digital points representing the cut-off parts, the correspondence is not straightforward. If p_1p_n is a real line segment joining $p_1, p_n \in \mathbb{Z}^2$, then in the DSS($p_1$, p_n), which is the digitization of p_1p_n, the DSS properties are valid. But if we cut DSS(p_1, p_n) at "any" intermediate point, say p_m, then the (ordered) set of points from p_1 to p_m (and from p_{m+1} to p_n) might not be the digitization of the real line segment p_1p_m (and $p_{m+1}p_n$).

For example, in Fig. 9.1, DSS(p_1, p_n) is cut at p_m in such a way that the rightmost run-length of DSS(p_1, p_m) is too small compared to the other runs (including the leftmost run-length). In such a case, the new DSS, namely DSS(p_1, p_m), obtained by digitization of the real line segment p_1p_m, is not a subset of the original DSS, i.e., DSS(p_1, p_n). Thus, the difference is: When two points p and q are arbitrarily selected from a real straight line segment L, then the new straight line segment pq is always a part of the original one; whereas, if p and q are two arbitrarily selected points from a digital straight line segment, DSS(p_1, p_n), then DSS(p, q) may or may not be a part/subsequence of the original segment, DSS(p_1, p_n).

Nevertheless, when a DSS is cut at an arbitrary point p_m to get $\langle p_1, p_2, \ldots, p_m \rangle$ as a sub-DSS, it may not be the digitization of p_1p_m, but it remains to be a DSS in the sense that it represents a part of the digitization of some real line or line segment different from the real line segment p_1p_m. In fact, a finite set of digital points satisfying the properties of digital straightness is a subset of digitization of infinitely many real lines, which does not stop an arbitrarily cut DSS to remain a DSS. Thus, the similarity is: Whether in case of a real or a digital straight line

segment, an arbitrary part of it is always straight—taken in the respective real or digital sense.

The problems related with DSS may be categorized into two classes. One class deals with the mapping from $\mathbb{R}^2$ to $\mathbb{Z}^2$: Given a real straight line segment joining two digital points p and q, what is the sequence of digital points constituting DSS(p, q)? There exists several algorithms to find the points of DSS(p, q) [20, 22, 58, 84], and the problem is relatively simpler. If $p = (i_p, j_p)$ and $q = (i_q, j_q)$, and w.l.o.g., if $i_p < j_p$ and if the slope of the real line pq be in $[0, 1]$, then DSS(p, q) consists of the nearest digital points corresponding to those real points on pq which have integer abscissas in the interval $[i_p, i_q]$. That is,

$$\mathrm{DSS}(p,q) = \left\{ (i,j) \in \mathbb{Z}^2 \;\middle|\; i_p \leqslant i \leqslant j_p \wedge j = \left\lfloor y + \frac{1}{2} \right\rfloor \wedge \frac{y - j_q}{j_q - j_p} = \frac{i - i_q}{i_q - i_p} \right\}. \tag{9.1}$$

The other class deals with problems related with digital straightness, which mainly involves the mapping from $\mathbb{Z}^2$ to $\mathbb{R}^2$. To introduce, some of the intriguing problems from this class are listed below.

1. How many different DSLs (digital straight lines) exist passing through two given points, p and q?
2. How it can be proved that the points of DSS generated by a DSS algorithm are "placed near to" Euclidean straight line $y = ax + b$?
3. How a criterion for initialization conditions can be obtained when DSS extraction algorithm should generate exactly the Bresenham's DSS [20]?
4. Given the sequence of digital points constituting a digital curve segment, how to decide whether it is a DSS or not?
5. Is it possible to construct any algorithm that can normalize the set (space) of DSS(s), i.e., attaching the length to any DSS starting at a digital point p and ending at another digital point q?
6. How can we extract DSS(s) of maximum possible length(s) from a given digital curve segment?
7. How can we extract the minimal DSS cover from a given digital curve segment?

It may be noted that, solutions to Problem 1 and Problem 2 inherit from the close relationship between continued fractions and DSS [88, 103, 144]. Problem 3 has been addressed in [115], where a DSS has been defined without using the equation of the Euclidean straight line ($y = ax + b$). Regarding Problem 4, in which the input is a digital curve segment and output is "yes" or "no", depending on whether or not the digital curve segment is a DSS, there also exist several interesting solutions [43, 49, 50, 89, 102, 138]. As evident from their definitions, Problems 5–7 can be classified as very complicated ones. Further, besides the above problems, it is possible to formulate more problems related to DSS [115].

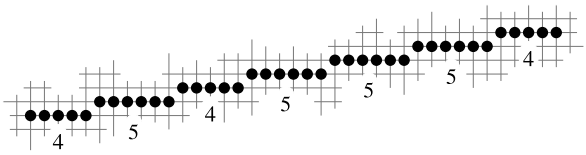


Fig. 9.2 $C = 0^4 1 0^5 1 0^4 1 0^5 1 0^5 1 0^5 1 0^4$ from p to q is not a DSS as it fails to satisfy R4. Its run length code is 4545554, in which the runs of 5 have lengths 1 and 3, which are not consecutive. However, if we split C into C' and C'', respectively from p to p' and from q' to q, then each of C' and C'' becomes a DSS

9.2.1 Properties of Digital Straightness

A (irreducible) digital curve segment S is a sequence of digital points in 8N or 4N connectivity [84]. In 8N (our consideration), $(i, j) \in C$ and $(i', j') \in C$ are neighbors of each other, provided $\max(|i - i'|, |j - j'|) = 1$, and the chain codes constituting C are from $\{0, 1, 2, \ldots, 7\}$ (Fig. 9.2). If each point in C has exactly two neighbors in C, then C is a *closed curve*; otherwise, C is an *open curve* having two points with one neighbor each, and the remaining points with two neighbors each. A self-intersecting curve C can be split into a set of open/closed curves (Fig. 9.3).

In order to determine whether (a part of) S is straight or not, there have been several studies since 1960s [84]. Interestingly, solutions to this problem inherit from the theory of continued fractions [84, 144]. In [122], it has been shown that S is the digitization of a straight line segment if and only if it has the *chord property*. (A curve S has the chord property if, for each point-pair $(p, q) \in S \times S$, $p \neq q$, for any (x, y) on the chord $\overline{pq}$ (real straight line segment joining p and q), $\exists (i, j) \in S$ such that $\max\{|i - x|, |j - y|\} < 1$.)

From the theory of words [97], it has been be shown that, if a DSL is the digitization of a real straight line with rational slope, then the chain code is periodic; otherwise, aperiodic [31]. Formulation of necessary conditions based on the properties of self-similarity w.r.t. chain codes, was first given in [61] as follows:

F1: At most two types of elements (chain codes) can be present, and these can differ only by unity, modulo 8;
F2: One of the two element values always occurs singly;
F3: Successive occurrences of the element occurring singly are as uniformly spaced as possible.

The above three properties were illustrated by examples and based on heuristic insights. Further, Property F3 is not precise enough for a formal proof [110]. Nevertheless, it explicates how the composition of a straight line in the digital space resembles that in the Euclidean space as far as the "evenness" (Sect. 9.1) in distribution of its constituting points is concerned. A few examples are given below to clarify the idea.

1. 001012100 is not a DSS, since F1 fails (three elements are present).
2. 001001100 is not a DSS, since F2 fails (none of 0 and 1 occurs singly).

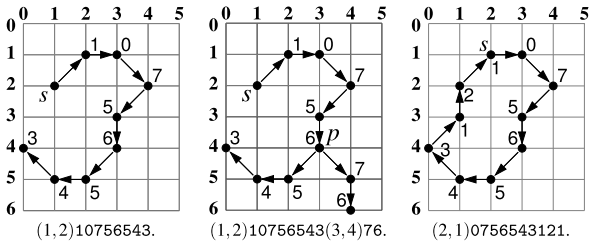


Fig. 9.3 Defining digital curve segments by chain codes. (Source: [14])

3. `010100010` is not a DSS, since F3 fails (singular element `1` is not uniformly spaced).
4. `010010010` is a DSS, since F1–F3 are true.

It may be noted that, for a chain code sequence `010100100`, the question of "uniform spacing" (as stated in F3) of the singular element `1` remains unanswered. For, there is one `0` between the first two consecutive `1`s, and there are two `0`s between the next two consecutive `1`s. The first formal characterization of DSS, which also brought in a further specification of Property F3, was however provided a few years later in [122], as stated in the following properties of DSS.

R1: The runs have at most two directions, differing by 45^0, and for one of these directions, the run length must be 1.
R2: The runs can have only two lengths, which are consecutive integers.
R3: One of the run lengths can occur only once at a time.
R4: For the run length that occurs in runs, these runs can themselves have only two lengths, which are consecutive integers; and so on.

It may be noted that the above four properties, R1–R4, still do not allow a formulation of sufficient conditions for the characterization of a DSS, but they specify F3 by a recursive argument on run lengths. Thus, `010100100` qualifies as a DSS by R1–R4, since the intermediate run-lengths (i.e., 1 and 2) of `0`s are consecutive. Another example is given in Fig. 9.2, which disqualifies Property R4, and hence is not a DSS. Once it is split into two appropriate pieces, the individual pieces pass through R1–R4 tests, and hence each of them becomes a DSS.

9.2.2 Approximate Straightness

In recent times, digital straightness has drawn special attention for their interesting periodic properties, which are found to be quite useful in applications related to digital images. For example, in a digital image containing one or more objects with fairly straight edges, the set of (approximate) digital straight line segments carries a strong geometric information of the underlying objects. Hence, the concept of approximate digital straight line segments (ADSS) has been proposed in [14]. In the ADSS, some of the most fundamental properties of DSS are preserved and some

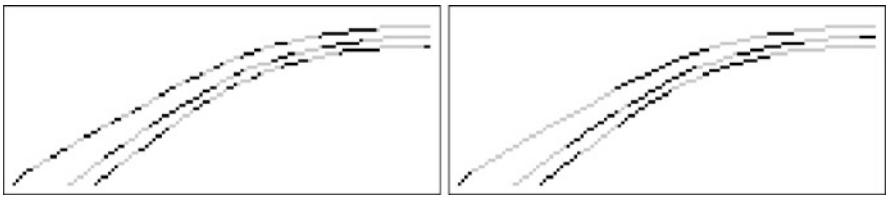


Fig. 9.4 Set of DSS (*left*) and that of ADSS (*right*), alternately colored in *black* and *gray*, extracted from a small set of digital curve segments. (Source: [14])

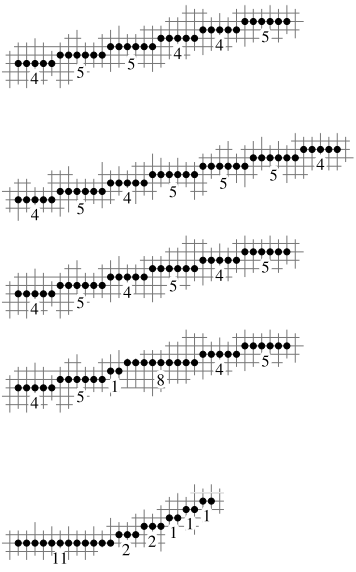
properties relaxed (see Fig. 9.5). The number of ADSS extracted from a set of digital curve segment S in a real world scenario is usually fewer than that of DSS cover, since many visually straight segments may fail to satisfy all stringent properties (R3 and R4, Sect. 9.2.1) of a DSS. For example, in Fig. 9.4 contains forty eight fragments, each of which is "exactly straight", whereas, that of ADSS contains only twenty, which look "visually straight".

The concept of ADSS can also be used for an efficient polygonal approximation. Since the set of ADSS provides an elegant and compact representation of digital curve segments, it is very effective in producing approximate polygons (or, polychains) using a single parameter. The whole process consists of two stages: extraction of ADSS and polygonal approximation. The major features are as follows.

- The detection of ADSS is based on chain-code properties; only primitive integer operations, such as comparison, increment, shift, and addition (subtraction) are required.
- Does not use any recursion, and thus saves execution time.
- To obtain the polygonal approximation, only the endpoints of ADSS are required with a few integer multiplications.
- The actual approximation of a digital curve segment never oversteps the worst-case approximation for a given value of a control parameter.

Several other methods [35, 39, 66, 152] have been proposed recently for (approximate) line detection. Most of the conventional parametric approaches are based on certain distance criteria, usage of masks, eigenvalue analysis, Hough transform, etc. In contrast, the ADSS-based method relies on utilizing some of the basic properties of DSS for extraction of ADSS. Earlier algorithms for approximating a given digital curve segment may be seen in [1, 9, 74]. Several variants of have been proposed later [16, 17, 112, 135, 137]. The class of polygonal approximation algorithms, in general, can be broadly classified into two categories: one in which the number of vertices of the approximate polygon(s) is specified, and the other where a distortion criterion (e.g. maximum Euclidean distance) is used.

Most of the existing polygonal approximation algorithms, excepting a few, have super-linear time complexities, for example, $O(N)$ in [145], $O(MN^2)$ in [112], $O(N^2)$ in [135] and [137], $O(N^3)$ in [120], where M denotes the number of segments, and N the total number of points representing the input set of digital curve segments. A comparative study of these algorithms can be found in [14, 154].



S_1 with chain code $0^4 1 0^5 1 0^5 1 0^4 1 0^4 1 0^5$ (from left to right) ($p = 4, q = 5, l = 4, r = 5$) that does not satisfy R3, since both the run lengths 4 and 5 have non-singular occurrences in the code of run lengths: 455445. S_1 is not a DSS but an ADSS.
S_2 with chain code $0^4 1 0^5 1 0^4 1 0^5 1 0^5 1 0^5 1 0^4$ ($p = 4, q = 5, l = 4, r = 4$) does not satisfy R4, since in the run length code 4545554, the runs of 5 have lengths 1 and 3 that are not consecutive. S_2 is not a DSS but an ADSS.

S_3 with chain code $0^4 1 0^5 1 0^4 1 0^5 1 0^4 1 0^5$ ($p = 4, q = 5, l = 4, r = 5$) that satisfies (R1–R4) and (c1, c2). S_3 is an ADSS as well as a DSS.

S_4 with chain code $0^4 1 0^5 1 0 1 0^8 1 0^4 1 0^5$ ($p = 1, q = 8, l = 4, r = 5$) that does not satisfy R2 and conditions (c1, c2). S_4 is neither a DSS nor an ADSS.
S_5 with chain code $0^{11} 1 0^2 1 0^2 1 0 1 0 1 0$ ($p = 1, q = 2, l = 11, r = 1$) that violates R2 and is, therefore, not a DSS. Further, although it satisfies (c1) (as $q - p (= 1) \leqslant d (= 1)$), but since $l - p (= 10) \not\leqslant e (= 1)$, it fails to satisfy (c2); and therefore, it is not even recognized as an ADSS.

Fig. 9.5 Instances of digital curve segments showing the significance of properties and conditions related with DSS and ADSS recognition. (Source: [14])

Further, in order to analyze curvature, most of them require intensive floating point operations [2, 56, 62, 142, 151]. For other details, the reader may look at [13, 54, 111, 125, 142, 145, 149, 155, 156]. The ADSS-based proposed here uses only integer operations, and yields a suboptimal polygonal approximation with linear time complexity (see [14]).

9.2.3 Extraction of ADSS

In the algorithm EXTRACT-ADSS, designed for extraction of ADSS from a digital curve segment S, we have used R1 along with certain modifications in R2. However, we have dropped R3 and R4, since they impose very tight restrictions on S to be recognized as a DSS. Such a policy has been done in order to successfully extract the ADSS, and some of the advantages are as follows.

- avoiding tight DSS constraints, especially while representing the gross pattern of a real-world image with digital aberrations/imperfections;
- enabling extraction of ADSS from a curve segment, thereby straightening a part of it when the concerned part is not exactly "digitally straight";
- reducing the number of extracted segments, thereby decreasing storage requirement and run-time in subsequent applications;

- reducing the CPU time of ADSS extraction;
- usage of integer operations only (e.g., to compute $\lfloor (p+3)/4 \rfloor$, 3 is added with p, followed by two successive right shifts).

Since the chain code of a curve segment is taken in a one-dimensional list, S, the ADSS may be characterized by the following sets of parameters:

- *orientations parameters*: n (non-singular element), s (singular element), l (length of leftmost run of n), and r (length of rightmost run of n). They play decisive roles on the orientation (and the digital composition, thereof) of the concerned ADSS. For example, in Fig. 9.5, the curve S_1 has $n = 0$, $s = 1$, and chain code $0^4 1 0^5 1 0^5 1 0^4 1 0^4 1 0^5$ having $l = 4$ and $r = 5$.
- *run length interval parameters*: p and q, where $[p, q]$ is the range of possible lengths (excepting l and r) of n in S that determines the level of approximation of the ADSS, subject to the following two conditions:

$$\text{(c1) } q - p \leqslant d = \lfloor (p+1)/2 \rfloor. \tag{9.2}$$

$$\text{(c2) } (l - p), (r - p) \leqslant e = \lfloor (p+1)/2 \rfloor. \tag{9.3}$$

While implementing EXTRACT-ADSS, we strictly adhere to R1, as it is directly related to the overall straightness of S. However, we have modified the stricture in R2 by considering that the run lengths of n can vary by more than unity, depending on the minimum run length of n. The rationale of modifying R2 to condition (c1) is that, while approximating the extracted line segments from S, an allowance of approximation (d) specified by (c1) is permitted. Given a value of p, the amount d by which q is in excess of p indicates the deviation of the ADSS from the actual/real line, since ideally (for a DSS) q can exceed from p by at most unity (the significance of d in characterizing an ADSS is detailed out in [14]).

Apart from d, the other parameter, namely e, is incorporated in (c2), which, along with (c1), ensures that the extracted ADSS is not badly approximated owing to some unexpected values of l and r. The DSS properties R1–R4, however, do not give any idea about the possible values of l and r (depending on n). Further, in the algorithm for DSS recognition [43], l and r are not taken into account for adjudging the DSS characteristics of a curve segment. However, we impose some bounds on the possible values of l and r, in order to ensure a reasonable amount of straightness at either end of an extracted ADSS. The values of d and e are heuristically chosen so that they become computable with integer operations only. Some other values, like $d = \lfloor (p+3)/4 \rfloor$ and $e = \lfloor (p+1)/2 \rfloor$, or so, may also be chosen provided the computation is realizable in integer domain and does not produce any undesirable ADSS. For example, in Fig. 9.5, the curve S_5 has $p = 1, q = 2, l = 11, r = 1$. In our case (Eqs. (9.2) and (9.3)), therefore, we get $d = 1$ and $e = 1$ resulting a violation of (c2) by l; thus S_5 will not be accepted as an ADSS.

To justify the rationale of (c1) and (c2), we consider a few digital curves, S_1–S_5, as shown in Fig. 9.5. It's interesting to observe that, although each of S_1 and S_2 has the appearance of a digital line segment, they fail to hold all the four properties of DSS simultaneously, as shown in their respective figures. The curve S_1 violates R3, and the curve S_2 violates R4. However, they satisfy R1, (c1), and (c2) and therefore,

Algorithm 1: EXTRACT-ADSS to find out the ordered list $\mathscr{A}$ of end points of ADSS in the input curve S that contains the chain code for each connected component. (Source: [14])

Algorithm EXTRACT-ADSS (S)
1. $\mathscr{A} \leftarrow \{1\}, u \leftarrow 1$
2. FIND-PARAMS ($\mathscr{C}, u$)
3. $c \leftarrow l$
4. **if** $s - n \pmod 8 \neq 1$ **then goto** Step 20
5. $p \leftarrow q \leftarrow$ length of next run of n
6. $d \leftarrow e \leftarrow \lfloor (p+1)/2 \rfloor$
7. $c \leftarrow c + 1 + p$
8. **if** $l - p > e$ **then goto** Step 20
9. **while** $q - p \leqslant d$
10. $k \leftarrow$ length of next run of n
11. **if** $l - k > e$ **then**
12. $c \leftarrow c + 1 + k$
13. **break**
14. **if** $k - p \leqslant e$ **then** $c \leftarrow c + 1 + k$
15. **else** $c \leftarrow c + 1 + p + e$
16. **if** $k < p$ **then**
17. $p \leftarrow k$
18. $d \leftarrow e \leftarrow \lfloor (p+1)/2 \rfloor$
19. **if** $k > q$ **then** $q \leftarrow k$
20. $u \leftarrow u + c$
21. $\mathscr{A} \leftarrow \mathscr{A} \cup \{u\}$
22. $u \leftarrow u + 1$ ▷ next start point
23. repeat from Step 2 until $\mathscr{C}$ is finished

Procedure FIND-PARAMS (S, u)
1. $i \leftarrow u$
2. **if** $\mathscr{C}[i] = \mathscr{C}[i+1]$ **then**
3. $n \leftarrow \mathscr{C}[i]$
4. $s \leftarrow$ element $\neq n$ following $\mathscr{C}[i+1]$
5. $l \leftarrow$ leftmost run length of n
6. **return**
7. **else**
8. $n \leftarrow \mathscr{C}[i], s \leftarrow \mathscr{C}[i+1], l \leftarrow 1$
9. $i \leftarrow i + 1$
10. **while** $\mathscr{C}[i+1] \in \{n, s\}$ ▷ end[$\mathscr{C}$] $= -1$
11. **if** $\mathscr{C}[i] = \mathscr{C}[i+1]$ **then**
12. **if** $\mathscr{C}[i] = s$ **then**
13. swap n and s
14. $l \leftarrow 0$
15. **return**
16. **else**
17. $i \leftarrow i + 1$
18. **return**

each of them is declared as an ADSS. Similarly, the curve S_3 satisfies R1–R4 and (c1, c2); it is both an ADSS and a DSS. However, none of the curves S_4 and S_5 can be announced as a DSS or an ADSS because of the violation of R2, (c1), and (c2).

9.2.4 Algorithm EXTRACT-ADSS

Algorithm 1 shows the algorithm EXTRACT-ADSS for extracting ADSS from the chain code of each digital curve segment, say S_k, stored in the list S. This requires n_k repetitions from Step 2 through Step 23, where n_k is the number of ADSS in S_k. Let the ith repetition on S_k produces the ADSS $\mathbf{L}_i^{(k)}$. Recognition of $\mathbf{L}_i^{(k)}$ is prompted by finding its corresponding parameters (n, s, l) using the FIND-PARAMS procedure in Step 2 of EXTRACT-ADSS. This is followed by checking/validation of

- property R1: Step 4 and Step 10;
- condition (c1): **while** loop check at Step 9;
- condition (c2): on the leftmost run length l in Step 8 and Step 11, and on the rightmost run length r in Step 14.

Proof of correctness For each ADSS, $\mathbf{L}_i^{(k)}$, we show that property R1 and conditions (c1) and (c2) are simultaneously satisfied. We also show that $\mathbf{L}_i^{(k)}$ is maximal in length in S_k in the sense that inclusion of the character (n or s or any other in $\{0, 1, \ldots, 7\}$) (or a substring of characters) that immediately precedes or follows the part of digital curve segment corresponding to $\mathbf{L}_i^{(k)}$ in S_k does not satisfy the ADSS property/conditions.

While checking R1 in Step 4 or Step 10, if an expected n or s is not found at the desired place in S_k, then the current ADSS, $\mathbf{L}_i^{(k)}$ ends with the previously checked valid characters. This is explicit in Step 4 and implicit in Step 10. Thus $\mathbf{L}_i^{(k)}$ satisfies R1, and is maximal from its starting point and the finishing end, since either it is the first ADSS in S_k or the previous ADSS $\mathbf{L}_{i-1}^{(k)}$, was maximal.

Now, for each new run (of n), (c1) is verified in Step 9—excepting the leftmost run, l, which is not required since p (and q) does not exist for a single run—after appropriately updating p and q in Step 17 and Step 19 respectively, whenever necessary. In Step 9, if it is found that q is unacceptably large (i.e., $q \not\leqslant p + d$), then the **while** loop (Steps 10–19) is not executed, and the current ADSS, $\mathbf{L}_i^{(k)}$, ends with the truncated part of that run (truncated maximally, i.e., up to length $p + e$, in Step 15 of the previous iteration) as its rightmost run, r.

For checking (c2), however, we have to be more careful. For the second run (i.e., the run immediately following l) of the current ADSS, (c2) is checked (with respect to l) in Step 8. It may be noted that, if $l - p > e$, then (c2) is not satisfied, and so the first two runs (l and its successor) trivially constitute an ADSS by Step 7; because for two runs, we get only l and r (and no p or q), and no relation is imposed between l and r to define an ADSS.

For the third and the subsequent run(s), if any, the corresponding run length is stored in k (Step 10). If some (small enough) k violates (c2), then that k is treated as r (Steps 11–13), and the current ADSS ends with that run as the rightmost run (of run length k), whereby the maximality criterion of the ADSS is fulfilled. Otherwise, if k does not exceed the maximum possible length of the rightmost run (checked in Step 14), then we consider k as a valid run of the current ADSS (Step 14), else we truncate it to the maximum permissible length ($p+e$) as the rightmost run (Step 15). Note that, if $k > p + e$, then $k > q$ (for $p + e \geqslant p + d \geqslant q$), and Step 19 updates q to k, whence (c1) will be false in Step 9 in the next iteration, and so, the ADSS will end here with the (maximally) truncated part ($p + e$) as its rightmost run.

Time complexity Determination of the parameters (n, s, l) in FIND-PARAMS consists of two cases—the first one (Steps 2–6) being easier than the second (Steps 7–18). In either of these two cases, the procedure searches linearly in S for two distinct (but not necessarily consecutive) chain code values and determines the parameters accordingly. As evident from the loop in either case, the three parameters are obtained using only a few integer comparisons. The number of comparisons is $l + 1$ for the first case, and that for the second case is the number of characters in S until two consecutive non-singular characters are found.

The parameters n, s, l obtained in FIND-PARAMS are successively passed through a number of check points, as mentioned earlier, which take constant time as

evident in Steps 3–8 of EXTRACT-ADSS. In Step 5 of EXTRACT-ADSS, the first run length of n is measured immediately after the leftmost run length of n, if any, and it starts from the first non-singular character out of the two consecutive characters detected in FIND-PARAMS. In Step 10 of EXTRACT-ADSS, we have another simple (and silent) loop that determines in linear time each valid run of n in S, the validity criteria being verified and updated in Steps 9–19, each of these steps taking constant time. Hence, for the ADSS, $\mathbf{L}_i^{(k)}$, the algorithm EXTRACT-ADSS, together with the procedure FIND-PARAMS, takes linear time; wherefore the time complexity for extraction of all ADSS in S is strictly linear on the number of points in S.

9.3 Digital Circularity

The sequence of digital points constituting a digital circle has an interesting relation with the distribution of perfect squares (square numbers) in integer intervals. Such number-theoretic concepts are found to be useful both for construction of digital circles [15] and for determining the digital circularity of a given digital curve segment [107]. We discuss here about the latter problem: Given a sequence of chain-codes (Sect. 9.2) or run-lengths (number of contiguous points with same x- or y-coordinate) [123] representing a digital curve segment S, the problem is to decide whether the segment S is an arc of a digital circle. If S is not entirely circular, then the maximum number of runs (starting from the first run) in S satisfying the properties of digital circularity is reported along with the corresponding radius or range of radii.

It may be mentioned here that, if S is an arbitrary 8-connected digital curve segment, then each individual run of S is always a run of one or more digital circles (Sects. 9.3.2 and 9.3.3). In other words, given any positive integer λ, there always exists a range of digital circles with their radii lying in $[r', r'']$, such that each of these circles (in each of the eight octants, and hence in Octant 1) contains some run(s) having length λ. Thus, trivially, each run of S is always a digital circular arc. The problem becomes non-trivial when two successive runs of S are considered, since we have to decide whether there exists a common digital circle having these two runs in succession. In fact, such circles may exist in multiple, which are *conflicting circles*, as explained in Sect. 9.3.2. And the problem complexity eventually increases when more run-lengths are considered from S. A representative example of only two runs is illustrated in Fig. 9.7.

A digital circle $\mathscr{C}^{\mathbb{Z}}(o, r)$ centered at $o(0, 0)$ and having radius $r \in \mathbb{Z}^+$ consists of eight symmetric octants, and the properties in one octant are also valid in the other seven octants with appropriate modifications (Fig. 9.6). Note that, a digital point (i, j) lies in Octant 1 if and only if $0 \leqslant i \leqslant j$. Hence, for simplicity of subsequent discussions we consider that, if S is digitally circular, then all the runs of S, w.l.o.g., lie in Octant 1. Thus, the chain-codes corresponding to S are considered to be of the form $0^a 7 0^b 7 0^c 7 \ldots$ $(a, b, c, \ldots \geqslant 0)$. If the chain-code representation of a digital curve segment is such that the curve segment lies in more than one octant,

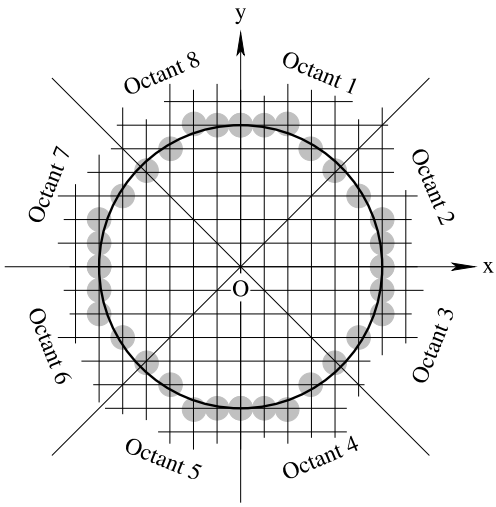


Fig. 9.6 Eight octants of a digital circle

then we split it for corresponding octants accordingly. Also, for notational simplicity, henceforth we consider that S denotes the (8-connected) digital curve segment, or equivalently, the sequence of either n runs or lengths $(a, b, c, \ldots)$ of these n runs, defining the underlying digital curve segment. We can consider the problem of determining the digital circularity of S in two versions, which are as follows:

Version 1 (Sect. 9.3.2) The sequence S is considered to have the starting correspondence with the topmost run (i.e., with maximum y-coordinate) in Octant 1 of the concerned digital circle. Thus, S is digitally circular, or, said to possess digital circularity, if and only if there exists a digital circle $\mathscr{C}^{\mathbb{Z}}(o, r)$ having center $o(0, 0)$ and radius r whose top n runs (corresponding to $k = 0, 1, \ldots, n-1$) are identical with the respective n runs of S.

Version 2 (Sect. 9.3.3) The sequence S may start from any run excepting the topmost run in Octant 1 of the concerned digital circle if it is digitally circular. So, S is digitally circular if and only if there exist a digital circle $\mathscr{C}^{\mathbb{Z}}(o, r)$ and an integer $k'(\geqslant 1)$ such that the successive n runs of $\mathscr{C}^{\mathbb{Z}}(o, r)$ starting from its k'th run (i.e., $k = k', k'+1, \ldots, k'+n-1$), are identical with the respective n runs of S.

Although Version 1 of the problem is a sub-problem of Version 2 (when $k' = 0$), we have treated them separately due to the fact that the latter is computationally more complex than the former. For, ensuring the properties of digital circularity in Version 2 implies not only deciding the existence of a radius r or a range of radii $[r', r'']$, but also finding an appropriate integer k' (whereas, $k' = 0$ in Version 1).

For an arbitrarily large circle, the integer k' may become arbitrarily large—especially when the runs of S correspond to the trailing runs of the circle in Octant 1—which requires a feasible trade-off for practical applications. Also, if S as a whole does not satisfy digital circularity and only the first $n'(< n)$ runs of S constitute the maximum subsequence satisfying digital circularity in Version 1, then we

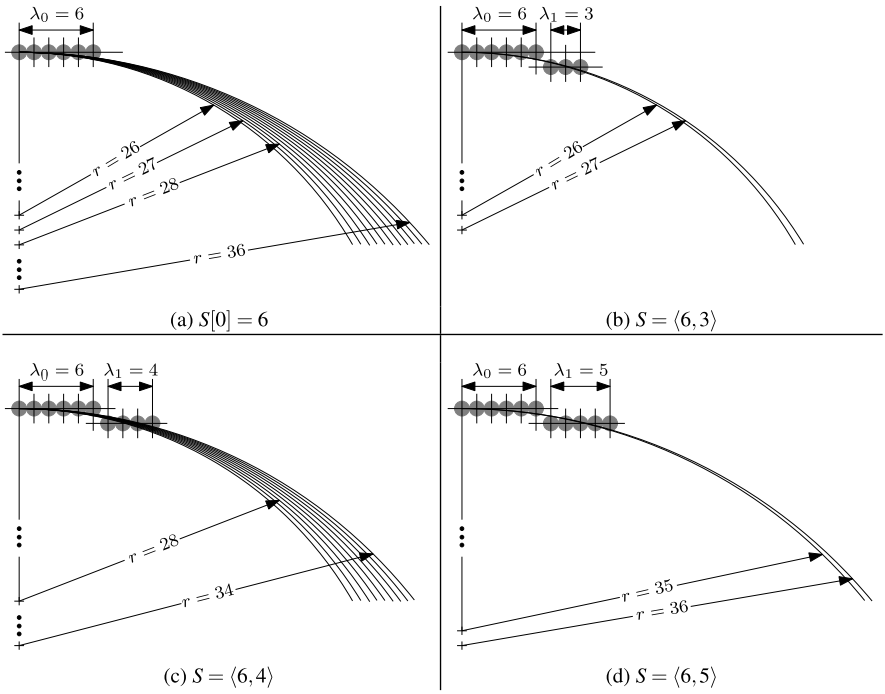


Fig. 9.7 (**a**) Given that $S[0] = 6$ is the first run-length of a digital curve segment S, which corresponds to the topmost run-length λ_0 of a digital circle, we find that there are many (*conflicting*) circles, and their radii are in [26, 36]. (**b**) If the next run-length is $S[1] = 3$, then out of these circles, only two circles ($r \in [26, 27]$) satisfy it ($\lambda_1 = S[1] = 3$), and thus the conflicting range decreases; (**c**) $S[1] = 4$ is satisfied by seven circles with $r \in [28, 34]$. (**d**) $S[1] = 5$ is satisfied by two circles with $r \in [35, 36]$. However, if $S[1] \leqslant 2$ or $S[1] \geqslant 6$, then there is no digital circle whose topmost run-length is $S[0] = 6$

can resort to Version 2 to find whether there exists some $k' > 0$ for which a longer subsequence of length $n''(> n')$ satisfies digital circularity, depending on the need and precision of the concerned application. In particular, we may also find the value of k' for which n'' is maximum, so that $n'' = n$ in the best case (see Fig. 9.8). Evidently, Version 2 is much harder to solve compared to Version 1. Solutions to the above two versions of the problem, in turn, are used by us to derive a circular segmentation of digital curve segments. If a digital curve segment S as a whole does not correspond to an arc of a digital circle, then we fragment S into a sequence of digital circular arcs, each of which is locally maximum in length.

9.3.1 Existing Works

Theorization and experimentation with the properties of digital discs/circles related to the reconstruction/segmentation problem can be traced back to 1970's

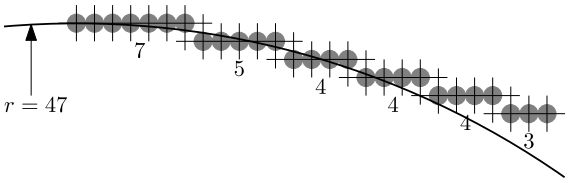
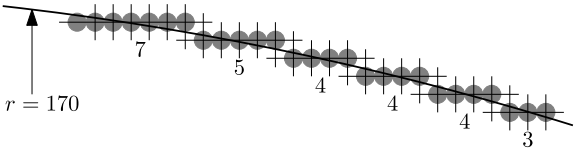


Fig. 9.8 An illustration to differentiate Version 1 and Version 2 of the problems for a digital curve segment $S = \langle 7, 5, 4, 4, 4, 3 \rangle$

Version 1. First three runs are digitally circular corresponding to the top three (i.e., $n' = 3$) runs of $\mathscr{C}^{\mathbb{Z}}(47)$.



Version 2. All seven runs (i.e., $n'' = n = 7$) are digitally circular with the correspondence of the first run ($S[0] = 7$) to the third ($k = 2$) run of $\mathscr{C}^{\mathbb{Z}}(170)$.

[47, 48, 67, 91, 104]. Most of those works, however, did not consider the inherent digital-geometric properties of digital discs/circles, which indeed is more complex than it appears [32]. In later years, and especially over the last few years, the challenge of the problem in the digital plane gradually has become apparent with the advent of new theoretical advances, such as digital calculus [104, 105], digital topology [87], digital geometry [24, 26, 82, 84], computational imaging [8, 11, 121], and theory of words and numbers [23, 86]. The classical results about the convex hull of the integer points in a disc [10] and a more recent work on the integer points of a hyperball [27] further emphasize that the study of digital-geometric properties of circles is an interesting topic of research for theoretical enrichment as well as practical applications.

A brief overview of the existing algorithms including the one based on number-theoretic approach in the chronological order is given below.

Circularity measure As an obvious and foremost way to decide whether a *set of digital points* $Q \subset \mathbb{Z}^2$ has a circular shape, a *circularity measure* was proposed in [67]. The objective was to estimate the closeness of the digital point set Q to a digital disk by approximating the corresponding Euclidean measure, $4\pi \mathrm{a}/\mathrm{p}^2$, a and p being the respective area and perimeter of Q. Although such a measure can be resorted to when a coarse approximation is in question, it fails to provide the exact information even when the concerned object is exactly circular in the digital plane [32, 79].

A geometric characterization of the digital disk was, therefore, introduced in [79] in consistency with the definitions of *digital convexity* and *digital polygonality*, in order to aid the procedure of determining whether or not a given digital object Q

is a digital disk.[1] The decision algorithm is based on verifying the convexity of the object Q, and subsequently checking whether each vertex pair from the convex hull of Q satisfies certain criteria of Euclidean geometry. The algorithm, however, has an excessive run-time of $O(n^3)$ complexity, n being the number of points in Q. It was improved to $O(n^2)$ in [80] using a relatively complicated geometric algorithm. However, the major drawback of all these algorithms is their lack of readiness to determine the circularity of Q when Q is an arbitrary fragment of a digital circle or a digital disk [129].

Arc separability Given two planar sets of points, P and Q, the problem of *arc separability* is a classical problem in computational geometry, which was first linked to recognize digital disks in [57]. The algorithm uses both the Voronoi and the furthest-point Voronoi diagrams on the real/Euclidean plane to find the set R (a convex polygon) of points, which are centers (at all feasible radii) of circles that contain all the members of P and none of Q. Clearly, the approach was mainly computational-geometric. Based on a similar approach, an algorithm to recognize digital circles from digital curve segments was presented later in [89]. However, complexity bounds of this algorithm were not provided, and it has been shown to have $O(n^2 \log n)$ computational cost in [40], where n is the number of points constituting the digital curve segment.

Linear programming In [134], a different computational-geometric technique was proposed to achieve linear time complexity for the circle recognition algorithm. It makes use of the *minimum covering circle* algorithm from [100], which, however, also has the inherent drawback of not recognizing an arc S of a digital circle like the earlier methods, since the relation of the minimum circle circumscribing S with the corresponding circle is not known. Later, based on the fact that the circular arc recognition problem is equivalent to solve a set of n inequalities in dimension 3, where n is the number of digital points of the digital curve segment, it was shown in [44] that linear programming can be, in fact, used to design a linear-time algorithm for detecting arcs of digital circles. However, no experimental results were presented in [44].

Curvature estimation Characterization of digital circular arcs has been related with curvature estimation in [41, 147]. While estimating the *local curvature* by moving a window along the digital contour, an optimality is achieved by considering the domain of all circular arcs that give rise to a specific digital pattern in the window. A domain, which can be one of the six types, namely straight, strictly convex, infinite convex, strictly concave, infinite concave, and non-circular, is calculated for all possible chain codes of length m from 3 to 9. Based on the domain type, the maximal recognizable radius, $MRR(m)$, is estimated to pose a limit on the radius of continuous arcs. As the radius of the arc domain varies, an unavoidable error

[1] In [79], Q is considered to be a digital disk if there exists a Euclidean/real circle such that Q comprises of all the digital points lying on or inside that real circle.

occurs in the radius or curvature estimation. This error is bounded by the *Geometric Minimum Variance Bound* (GMVB), an equivalent of the Cramér-Rao bound, which is computed using the characterization of the arc domain based on a moment generating function.

Hough transform Hough transform (HT) forms a standard practice to segment circular arcs [63, 96]. However, its computational burden is as high as $O(n^3)$, which makes the method unsuitable for real-time applications. Hence, improvements have been proposed over the years to detect circular arcs in an efficient way. In [59], several HT-based techniques have been discussed and compared, and *circular direct Hough transform* (CDHT) has been proposed to reduce the computational burden. Its major steps include (a) Hough space voting after a modified circle parametrization by first-order equations (instead of the conventional second-order equation); (b) clustering for recovery of spatial information; (c) detection of global maxima (i.e., circles) and local maxima (i.e., arcs); and (d) spurious peak identification.

The algorithm in [76] uses a 2D Hough transform for the detection of the circle centers in the Euclidean plane in the first step, followed by a validation of their existence by radius histogram in the next step. That the perpendicular bisector of every chord of a real/Euclidean circle passes through its center, is the property used there. This property for the *localization of centers* was also used earlier in [48], but it lacked robustness due to usage of only vertical and horizontal chords. Another algorithm in [81] is also similar to [76] in principle but avoids any gradient information, and uses a threshold value on the votes of chord pairs. In a recent work [37], sampling of points constituting the input digital contour has been used along with the above-mentioned chord property to reduce the voting computation.

Domain reconstruction A geometric characterization of the domain of a quarter of a digital circle was first described in [32] with an objective of *domain reconstruction* for the full circle from the domains of individual quarter circles. The domain of a digital circle is a volume in 3D parametric space, defined as the intersection of the set of upright cones at digital points inside the circle and that of the complements of upright cones at digital points outside the circle. A recent work using a similar notion may be seen in [40], which also correlates it with the arc separating problem. It shows how the *arc center domain*, $acd(P, Q)$, of two given point sets, P and Q, can be computed from the generalized Voronoi cell using the Euclidean metric. The algorithm uses the principle of *duality*—the computational-geometric approach to Hough transform—to obtain a circular segmentation of a digital curve segment using its associated polyline, L. Edges of the polyline L are added one by one and the emptiness of acd is tested. If the acd is empty, then a new domain is initialized. With the assumption that the input curve segment is *digitally convex* and consists of n points, the number of edges of L is bounded by $O(n^{\frac{2}{3}})$, wherefore the time complexity of the algorithm is shown to be $O(n^{\frac{4}{3}} \log n)$. Three constrained versions of online arc recognition problem and the domain reconstruction procedures are proposed recently in [128].

Linearity of tangent space Based on the fact that a sequence of chords (with successive endpoints and having equal length) of a circle corresponds to a sequence of collinear points in the *tangent space* [95], a linear-time algorithm has recently been proposed in [106]. The sequence of chords is first obtained by a polygonalization of the input digital curve. This chord sequence is represented as a sequence of points in the tangent space. By means of this representation, the problem of digital arc recognition in the object space is reduced to the problem of digital straight line recognition in the tangent space, which is solved in linear time. The algorithm can handle noisy and blurred images with a few heuristic parameters.

Number-theoretic solution Since the *squares of abscissas* of the grid points constituting the $k(\geqslant 0)$th run (in Octant 1) of the digital circle $\mathscr{C}^{\mathbb{Z}}(o, r)$, centered at $o(0, 0)$ and with radius r, lie in the interval $I_k := [\max\{0, (2k-1)r - k(k-1)\}, (2k+1)r - k(k+1) - 1]$ (Lemma 1), there is no change in the corresponding run-length with a change in r as long as I_k "slides on the integer axis" without any change in the squares contained by it. As a consequence, there exists a set of finitely many circles with identical length of their $k(\geqslant 0)$th runs, wherein lies the crux of the problem. Other added technicalities of the analysis include the fact that, with increasing k, the number of squares in I_k usually decreases and occasionally increases by at most unity, provided the radius r remains fixed. And also, for some radius $r' > r$ and for some $k' > k \geqslant 0$, the interval $I_{k'}$ corresponding to r' may contain a larger number of squares compared to I_k corresponding to r, which opens up new analytical possibilities, as revealed in this chapter. Eventually, it may happen that a digital curve segment S does not belong to any digital circle, or belongs to a particular digital circle, or belongs to a range of digital circles of consecutive integer radii, each of which whatsoever, is reported by our algorithm.

Interestingly, given a sequence S of run-lengths that correspond to an arc of a digital circle(s), the radius r or the range of radius $[r', r'']$, which satisfies the sequence S may depend on whether or not we analyze S starting from the top runs of the candidate circle(s). Especially, when S consists of a small number of runs, then it may correspond to an arc of a digital circle of radius r_1 and also to an arc of another digital circle of a significantly dissimilar radius r_2 such that the respective arcs are quite differently located apropos their run-positions. For example, if $S = \langle 7, 5, 4\rangle$ (run-lengths of the underlying curve segment), then it is an arc of $\mathscr{C}^{\mathbb{Z}}(r = 47, 48)$ corresponding to $k = 0, 1, 2$ and it is also an arc of $\mathscr{C}^{\mathbb{Z}}(r = 170, 171)$ corresponding to $k = 2, 3, 4$. This shows that there is a factor of inexactness while estimating the radius of S if it is found to be digitally circular (Fig. 9.8, Sect. 9.1).[2] A way out is to investigate its circularity starting from the lowest possible run-position, and ending at the highest possible one, which is a major feature of the proposed approach.

[2]The notion of inexactness in our work corresponds to a range of radius (and positions), which has some conceptual resemblance with the existence of generalized circumcenter proposed recently in [121].

9.3.2 Down the Top Run

A *digital curve segment* is defined as a finite sequence of integer points (i.e., points from $\mathbb{Z}^2$) connected (minimally) in 8-neighborhood [84]. A digital circle is a type of digital curve segment which has its pre-image as a real circle in $\mathbb{R}^2$. Depending on whether the radius and the center of a digital circle are real or integer values, several definitions of digital circles may be seen in the literature [3, 4, 15, 21, 58, 106, 113, 147]. We have considered the type of digital circle proposed in [21]. If we consider the radius $r \in \mathbb{Z}^+$ and the center $c = o(0, 0)$, then the first octant of the corresponding digital circle is given by

$$\mathscr{C}_1^{\mathbb{Z}}(o, r) = \left\{ (i, j) \in \mathbb{Z}^2 \;\middle|\; 0 \leqslant i \leqslant j \leqslant r \wedge \left| j - \sqrt{r^2 - i^2} \right| < \frac{1}{2} \right\}.$$

The entire digital circle consists of eight symmetric octants [15], and so is given by

$$\mathscr{C}^{\mathbb{Z}}(o, r) = \left\{ (i, j) \;\middle|\; \{|i|, |j|\} \in \mathscr{C}_1^{\mathbb{Z}}(o, r) \right\}. \tag{9.4}$$

Combining, we get the symmetry-free form as

$$\mathscr{C}^{\mathbb{Z}}(o, r) = \left\{ (i, j) \in \mathbb{Z}^2 \;\middle|\; \left| \max(|i|, |j|) - \sqrt{r^2 - \left(\min(|i|, |j|)\right)^2} \right| < \frac{1}{2} \right\}. \tag{9.5}$$

Clearly, an arbitrary digital circle $\mathscr{C}^{\mathbb{Z}}(p, r)$ having center at $p(i_p, j_p) \in \mathbb{Z}^2$ and radius $r \in \mathbb{Z}^+$ is given by

$$\mathscr{C}^{\mathbb{Z}}(p, r) = \left\{ (i + i_p, j + j_p) \;\middle|\; (i, j) \in \mathscr{C}^{\mathbb{Z}}(o, r) \right\}.$$

9.3.2.1 Nesting the Radii

We start with the following lemma [15] that relates each point, and each run thereof, of a digital circle, namely $\mathscr{C}^{\mathbb{Z}}(o, r)$ (Eq. (9.4) or Eq. (9.5)), to a unique square number in the corresponding integer interval.

Lemma 1 *The interval* $I_k := [u_k, v_k] = [\max\{0, (2k-1)r - k(k-1)\}, (2k+1)r - k(k+1) - 1]$ *contains the squares of abscissas of* (*all and only*) *the grid points constituting the* $k(\geqslant 0)$*th run in Octant* 1 *of* $\mathscr{C}^{\mathbb{Z}}(o, r)$.

If the top ($k = 0$) run-length be λ_0, then we can decide the corresponding range of radii of all those circles whose top run-length is λ_0, using the following lemma.

Lemma 2 λ_0 *is the length of top run of a digital circle* $\mathscr{C}^{\mathbb{Z}}(o, r)$ *if and only if* r *lies in the interval* $R_0 = [(\lambda_0 - 1)^2 + 1, \lambda_0^2]$.

Proof Let $r \in [(\lambda_0 - 1)^2 + 1, \lambda_0^2]$. From Lemma 1, for $k = 0$, we get $I_0 = [0, r - 1]$, or, $v_0 = r - 1$. Hence, the upper limit of I_0 is given by

$$v_0 \in \left[(\lambda_0 - 1)^2, \lambda_0^2 - 1 \right] \tag{9.6}$$

which contains exactly one square number, $(\lambda_0 - 1)^2$. As the number 0 (i.e., the point $(0, r)$) is included in the top run of $\mathscr{C}^{\mathbb{Z}}(o, r)$, the length of the top run becomes λ_0.

To prove the converse, let the length of the top run be λ_0. Then I_0 is such an interval that contains $(\lambda_0 - 1)^2$ as the largest square number. Hence, v_0 satisfies Eq. (9.6), or, $r - 1 \in [(\lambda_0 - 1)^2, \lambda_0^2 - 1]$, which implies $r \in [(\lambda_0 - 1)^2 + 1, \lambda_0^2]$. □

Lemma 2 shows that, given an arbitrary positive integer λ_0, there always exists a range of digital circles of consecutive radii lying in the interval R_0 whose length varies linearly with λ_0. However, for two given positive integers λ_0 and λ_1, there may or may not exist a digital circle whose top two runs have lengths λ_0 and λ_1 in succession. We have the following lemma.

Lemma 3 *λ_0 and λ_1 are the lengths of top two runs of a digital circle $\mathscr{C}^{\mathbb{Z}}(o, r)$ if and only if $r \in R_0 \cap R_1$, where, $R_1 = [\lceil \frac{(\Lambda_1 - 1)^2 + 3}{3} \rceil, \lfloor \frac{\Lambda_1^2 + 2}{3} \rfloor]$ and $\Lambda_1 = \lambda_0 + \lambda_1$.*

In particular, if $R_0 \cap R_1 = \emptyset$, then there exists no digital circle whose top two runs have length λ_0 and λ_1.

The proof of Lemma 3 follows from Lemma 1 and Lemma 2, and given in detail in [107]. The combined significance of Lemma 2 and Lemma 3 is illustrated in Fig. 9.7. Given that $\lambda_0 = 6$ and $\lambda_1 = 3$, we get the respective lower and upper limits for R_0 as $r_0' = (\lambda_0 - 1)^2 + 1 = 5^2 + 1 = 26$ and $r_0'' = \lambda_0^2 = 6^2 = 36$ (Lemma 2), and those for R_1 as $r_1' = \lceil ((\Lambda_1 - 1)^2 + 3)/3 \rceil = \lceil (8^2 + 3)/3 \rceil = 23$ and $r_1'' = \lfloor (\Lambda_1^2 + 2)/3 \rfloor = \lfloor (9^2 + 2)/3 \rfloor = 27$ (Lemma 3). Thus, $R_0 = [26, 36]$ and $R_1 = [23, 27]$, which implies that $[26, 36] \cap [23, 27] = [26, 27]$ is the range of radii of digital circles whose top two runs have lengths 6 and 3 in succession. For $r \in R_1' := R_1 \setminus R_0 = [23, 25]$, we have $\lambda_0 = 5$ and $\lambda_1 = 4$ (Lemma 1), which conforms to $\Lambda_1 = \lambda_0 + \lambda_1 = 9$, but not to the corresponding given values of λ_0 and λ_1.

Now, to decide whether there exits a valid range of radii corresponding to a given sequence of run-lengths, we can extend the above findings to obtain Theorem 1. Its proof is based on induction and uses the results of Lemma 1 and Lemma 2.

Theorem 1 (*Radii interval*) *$\langle \lambda_0, \lambda_1, \ldots, \lambda_n \rangle$ is the sequence of top $n + 1$ run-lengths of a digital circle $\mathscr{C}^{\mathbb{Z}}(o, r)$ if and only if*

$$r \in \bigcap_{k=0}^{n} R_k \tag{9.7}$$

where,

$$R_k = \left[\left\lceil \frac{1}{2k+1} \left((\Lambda_k - 1)^2 + k(k+1) + 1 \right) \right\rceil, \left\lfloor \frac{1}{2k+1} \left(\Lambda_k^2 + k(k+1) \right) \right\rfloor \right] \tag{9.8}$$

Algorithm 2: The decision algorithm DCT that verifies whether a sequence of top n runs corresponds to a digital circle

Input: `int` $S[0..n-1]$
Output: Whether S is digitally circular

```
1  Λ ← S[0];
2  [r′, r″] ← [(Λ − 1)² + 1, Λ²];
3  for k ← 1 to n − 1 do
4  |   Λ ← Λ + S[k];
5  |   s′ ← ⌈((Λ − 1)² + k(k + 1) + 1)/(2k + 1)⌉;
6  |   s″ ← ⌊(Λ² + k(k + 1))/(2k + 1)⌋;
7  |   if s″ < r′ or s′ > r″ then
8  |   |   print "S is circular up to (k − 1)th run for [r′, r″]";
9  |   |   return;
10 |   else
11 |   |   [r′, r″] ← [max(r′, s′), min(r″, s″)];
12 print "S is circular in entirety for [r′, r″]";
```

and

$$\Lambda_k = \sum_{j=0}^{k} \lambda_j .$$

In particular, if $\bigcap_{k=0}^{n} R_k = \emptyset$, *then there exists no digital circle whose top* $n+1$ *runs have length* $\langle \lambda_0, \lambda_1, \ldots, \lambda_n \rangle$.

9.3.2.2 The Algorithm DCT

Theorem 1 provides the way to determine whether a sequence S, consisting of n run-lengths, is digitally circular. Starting from $S[0]$, we can gradually come down the sequence while obtaining the corresponding radius/radii of the matching digital circle(s). The algorithm DCT (Digital Circularity from the Top run) that decides whether the sequence S corresponds to top n runs of a digital circle or a range of digital circles with radii in $[r', r'']$, is shown in Algorithm 2. For its proof of correctness, see [107]. To explain its time complexity, we observe that in each iteration of the **for** loop, $\Theta(1)$ time is consumed. Hence, if there exists at least one digital circle $\mathscr{C}^{\mathbb{Z}}(o, r)$ such that all n runs of S constitute the top n runs of $\mathscr{C}^{\mathbb{Z}}(o, r)$, then the algorithm DCT terminates successfully with $\Theta(n)$ time complexity. Otherwise, the algorithm terminates in $\Theta(n')$ time if $n' (< n)$ runs of S (i.e., $S[0..(n'-1)]$) constitute the top n' runs of some circle and $S[n']$ fails to be the length of n'th run of the circle.

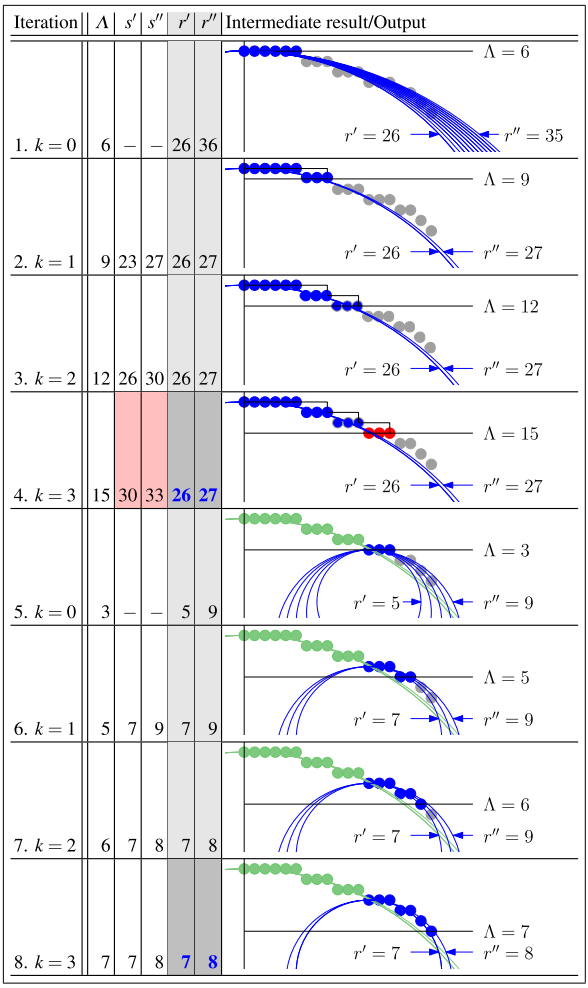


Fig. 9.9 Demonstration of circular segmentation on a representative digital curve segment corresponding to $S = \langle 6, 3, 3, 3, 2, 1, 1 \rangle$, using the algorithm DCT

Circular segmentation If the entire segment $S[0..n]$ is not digitally circular, then the algorithm DCT can be used iteratively to determine whether the full or a part of the remaining segment, i.e., $S[n'..n]$, is digitally circular. In case $S[n'..n]$ is circular in entirety, time complexity is $\Theta(n - n')$; otherwise, the time complexity is $\Theta(n'')$ if $S[n'..(n' + n'' - 1)]$ is maximally circular. In either case, on termination of the circular segmentation by iterative application of DCT, the total time complexity is given by $\Theta(n)$.

A demonstration of circular segmentation on a typical digital curve segment lying in Octant 1 is given in Fig. 9.9. Iteration 1 corresponds to Step 1 and Step 2 of the algorithm DCT (Algorithm 2) in which Λ is initialized to $S[0] = 6$, and the radius interval $R_0 := [r', r''] = [26, 36]$ is computed thereof. In Iteration 2, Λ is updated to $S[0] + S[1] = 6 + 3 = 9$, wherefore the next radius interval $[s', s'']$ is found

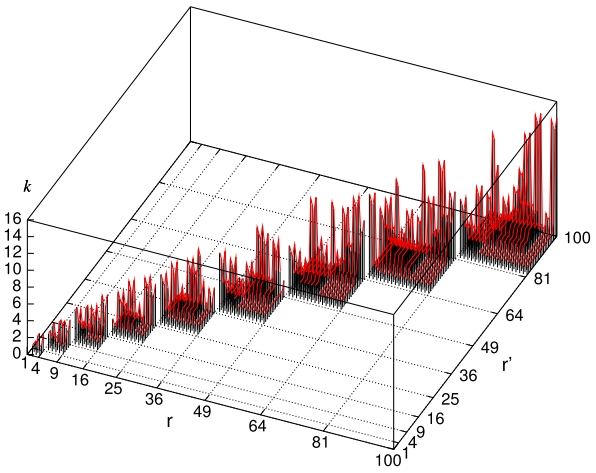


Fig. 9.10 Plot on conflicting radii starting from the top run-length. See text for explanation and also Fig. 9.11

to be [23, 27] (Steps 4–6 of DCT). The effective radius range is obtained to be $[26, 36] \cap [23, 37] = [26, 27]$ in Step 11 of DCT. In Iteration 3, Λ is increased from 9 to $9 + S[2] = 12$ in Step 4, and subsequently $[s', s'']$ are computed in Steps 5 and 6 of DCT. As $[s', s''] = [26, 30]$ has a nonempty intersection with the previous range, i.e., [26, 27], the effective radius range becomes [26, 27], as obtained in Step 11 of DCT. In Iteration 4, the new value of Λ becomes $12 + S[3] = 15$ for which the corresponding radius interval [30, 33] has no intersection with the previous interval [26, 27], as verified in Step 7 of DCT. Hence, the first three run-lengths of S satisfy digital circularity corresponding to the top three runs of the digital circles with $r \in [26, 27]$, which is reported in Iteration 4.

For the run-length $S[3] = 3$, Λ is re-initialized to $\Lambda = 3$ in Step 1 of DCT and the new radius range becomes $R_0 := [r', r''] = [5, 9]$, as computed in Step 2 of DCT. Proceeding in a similar fashion, the successive run-lengths of S are processed in Iterations 6–8 of Fig. 9.9. The algorithm of circular segmentation finally terminates on finding that $S[3..6]$ is digitally circular with $r \in [7, 8]$.

Rate of convergence It is quite interesting to note that the pattern of run-lengths in S has a significant impact on the rate of convergence of the radius or the range of radius of the underlying digital circles. It may so happen that although the first few runs have larger lengths in S, we might get a unique radius sooner than the case when the corresponding run-lengths are smaller. This is in dissent with the earlier notion that the length of the radius interval varies linearly with a particular run-length (Lemma 2). The 3D plot on conflicting radii r' versus a given radius r with increasing values of the position of the run $k(\geqslant 0)$ is shown in Fig. 9.10. From this plot, the overall increasing trend of conflicting radii and their resolving run-position with r is evident. This owes to the fact that for a small value of r and a "nearly equal" value of r', the number of corresponding equi-length top runs of $\mathscr{C}^{\mathbb{Z}}(o, r)$ and $\mathscr{C}^{\mathbb{Z}}(o, r')$ is usually less than that when r has a larger value with a nearly equal value of r'.

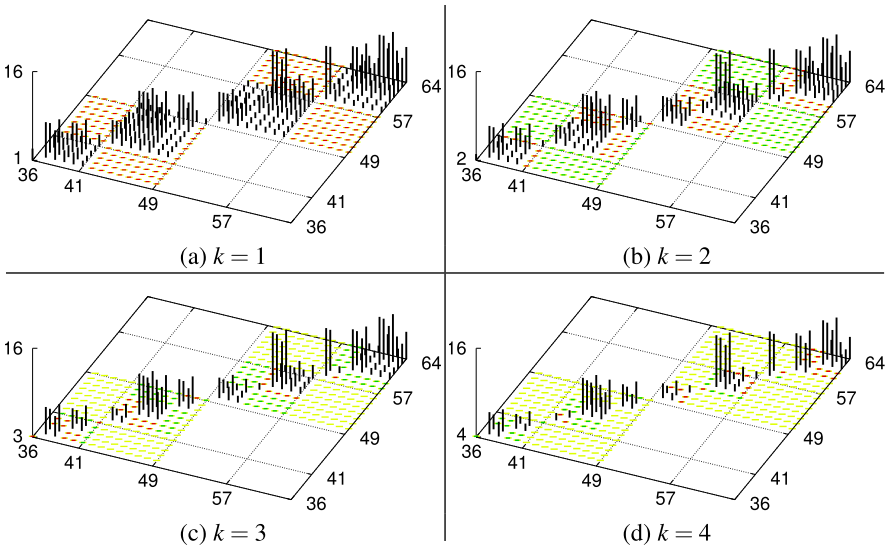


Fig. 9.11 Resolving the conflicting radii r' with increasing k for a part of the radius range from Fig. 9.10. Note that these plots are sliced off from Fig. 9.10 for $k = 1, 2, 3$ and 4. See text for further explanation

For an elaborate illustration, a part ($36 \leqslant r \leqslant 64$) of the plot of Fig. 9.10 has been given in Fig. 9.11, which shows how the conflicting radii gradually disappears as k increases from 1 to 4. As put in Lemma 2, r' is a conflicting radius for r at $k = 0$ if and only if r and r' are in an interval of the form $[(\lambda_0 - 1)^2 + 1, \lambda_0^2]$. Hence, for $k = 0$, the tuples (r', r) signify the integer points in $[1, 2^2]^2$, $[2^2 + 1, 3^2]^2$, $[3^2 + 1, 4^2]^2$, ..., as exhibited in the plot of Figs. 9.10 and 9.11. For $k \geqslant 1$, these *regions of conflict* gradually get disbanded by the effect of subsequent intervals R_k, $k \geqslant 1$ (Theorem 1).

For example, the radius in the range $[37, 49]$ that has the highest value of k to resolve the conflicting radii is $r = 45$. A similar radius in $[50, 64]$ is 55. For $r = 45$, we have all the radii in $[37, 49]$ as conflicting radii at $k = 0$. At $k = 1$, we have $[42, 49]$ to be conflicting with $r = 45$. Hence, if $r = 45$ and $r' = 41$ (or less), then no conflict arises from $\mathscr{C}^{\mathbb{Z}}(o, r')$ for $k \geqslant 2$. For $k = 2$, 3, and 4, the respective radii of conflict, therefore, gradually reduce to the ranges $[42, 46]$, $[44, 46]$, and $[44, 46]$. For $r = 55$ that lies in the next region of conflict, the respective regions of conflicting radii are $[50, 57]$, $[53, 57]$, $[54, 57]$, and $[54, 56]$ (Fig. 9.11(a–d)). It may be noticed in Fig. 9.11(d) that for $r = 45$, the conflicting radii are 44 and 46 for $k \geqslant 4$ out of which $r' = 44$ goes on conflicting (with $r = 45$) till $k = 7$ and $r' = 46$ till $k = 6$. Hence, the run-lengths of $\mathscr{C}^{\mathbb{Z}}(o, r = 45)$ do not get resolved until $k = 8$. Similarly, for $r = 55$, as $r' = 54$ and $r' = 56$ go on conflicting till $k = 9$ and $k = 4$ respectively, the run-lengths of $\mathscr{C}^{\mathbb{Z}}(o, 55)$ remain unresolved until $k = 10$. This explains the fact that a larger number of run-lengths are "usually" required to resolve a larger radius and a smaller number of run-lengths usually for a smaller radius. The opposite-cum-

unusual scenario, of course, also exists. Consider $r = 53$, which, although larger than $r = 45$, is resolved much earlier at $k = 3$ (Fig. 9.11(b, c)). Another such unusual instance is $r = 41$ that gets resolved at $k = 2$. Radii getting resolved at $k = 4$ are $r = 42, 43, 52, 57$, and so forth.

9.3.3 General Case

Some interesting properties of digital circularity for an arbitrary run or a sequence of runs that does not start from the topmost run ($k = 0$) are given below (proofs given in [107]).

Lemma 4 *If a digital circle of radius r contains a given run of length λ, then there exist two positive integers a and k such that $r \geqslant \lceil \max(f_{1,\lambda}(a,k), f_{2,\lambda}(a,k)) \rceil$, where*

$$f_{1,\lambda}(a,k) = \frac{(a-1)^2 + k(k-1) + 1}{2k-1} \tag{9.9}$$

and

$$f_{2,\lambda}(a,k) = \frac{(a+\lambda-1)^2 + k(k+1) + 1}{2k+1}. \tag{9.10}$$

Lemma 5 *If a digital circle of radius r contains a given run of length λ, then there exist two positive integers a and k such that $r \leqslant \lfloor \min(f_{3,\lambda}(a,k), f_{4,\lambda}(a,k)) \rfloor$, where*

$$f_{3,\lambda}(a,k) = \frac{a^2 + k(k-1)}{2k-1} \tag{9.11}$$

and

$$f_{4,\lambda}(a,k) = \frac{(a+\lambda)^2 + k(k+1)}{2k+1}. \tag{9.12}$$

Theorem 2 *An arbitrary run of given length λ belongs to a digital circle if and only if its radius lies in the range*

$$\mathscr{R}_{ak} = \left\{ r \mid r \geqslant \left\lceil \max_{a,k \in \mathbb{Z}^+} \left(f_{1,\lambda}(a,k), f_{2,\lambda}(a,k) \right) \right\rceil \right\} \cap \left\{ r \mid r \leqslant \left\lfloor \min_{a,k \in \mathbb{Z}^+} \left(f_{3,\lambda}(a,k), f_{4,\lambda}(a,k) \right) \right\rfloor \right\}.$$

It may be noted that, for a given value of k, $y = f_{1,\lambda}(x,k)$ represents a parabola with $x = 1$ as the axis. Hence, $r \geqslant f_{1,\lambda}(a,k)$ signifies the interior region of (and lying on) $f_{1,\lambda}(a,k)$. Similarly, for the same given value of k, $r \geqslant f_{2,\lambda}(a,k)$ is the interior region of $f_{2,\lambda}(a,k)$, and $r \leqslant f_{3,\lambda}(a,k)$ and $r \leqslant f_{4,\lambda}(a,k)$ are the respective exterior regions of the parabolas $f_{3,\lambda}(a,k)$ and $f_{4,\lambda}(a,k)$. As a result, the region $\mathscr{R}_{ak}$ signifies the (closed) region enclosed by the four parabolic curves, namely,

Table 9.1 Points of intersection (in $\mathbb{R}^2$) among the parabolas $\{f_{i,\lambda} \mid i = 1, 2, 3, 4\}$ defining $\mathscr{R}_{ak}$ ($\overline{k} = 2k - 1$, $\underline{k} = 2k + 1$, $\hat{\overline{k}} = k(k-1)$, $\hat{\underline{k}} = k(k+1)$, $\overline{\lambda} = \lambda - 1$)

Parabolas		Point	Abscissa of the point
$f_{1,\lambda}$	$f_{2,\lambda}$	α_{12}	$\frac{1}{2}(\overline{k}\lambda + \sqrt{(\overline{k}\lambda+2)^2 + 2(\overline{k}\,\overline{\lambda}^2 + 2\hat{\overline{k}} - 3)} + 2)$
$f_{2,\lambda}$	$f_{3,\lambda}$	α_{23}	$\frac{1}{2}(\overline{k}\,\overline{\lambda} + \sqrt{(\overline{k}\,\overline{\lambda})^2 + 2(\overline{k}\,\overline{\lambda}^2 + 2\hat{\underline{k}} - 1)})$
$f_{3,\lambda}$	$f_{4,\lambda}$	α_{34}	$\frac{1}{2}(\overline{k}\lambda + \sqrt{(\overline{k}\lambda)^2 + 2(\overline{k}\lambda^2 + 2k^2)})$
$f_{4,\lambda}$	$f_{1,\lambda}$	α_{41}	$\frac{1}{2}(\overline{k}\lambda + \underline{k} + \sqrt{(\overline{k}\lambda + \underline{k})^2 + 2(\overline{k}\lambda^2 + 2\hat{\overline{k}} - \underline{k} - 1)})$

Table 9.2 Specifications of the parabolas $\{f_{i,\lambda} \mid i = 1, 2, 3, 4\}$ (notations as in Table 9.1)

Parabola	Axis	Directrix	Length of Latus Rectum	Vertex	Focus
$f_{1,\lambda}$	$x = 1$	$\overline{k}y = 3/4$	$\overline{k}$	$(1, (\hat{\overline{k}} + 1)/\overline{k})$	$(1, (8\hat{\underline{k}} + 5)/(4\overline{k}))$
$f_{2,\lambda}$	$x = -\overline{\lambda}$	$\underline{k}y = 3/4$	$\underline{k}$	$(-\overline{\lambda}, (\hat{\underline{k}} + 1)/\underline{k})$	$(-\overline{\lambda}, (8\hat{\overline{k}} + 5)/(4\underline{k}))$
$f_{3,\lambda}$	$x = 0$	$\overline{k}y = -1/4$	$\overline{k}$	$(0, (\hat{\overline{k}})/\overline{k})$	$(0, (8\hat{\underline{k}} + 1)/(4\overline{k}))$
$f_{4,\lambda}$	$x = -\lambda$	$\underline{k}\,y = -1/4$	$\underline{k}$	$(-\lambda, \hat{\underline{k}}/\underline{k})$	$(-\lambda, (8\hat{\overline{k}} + 1)/(4\underline{k}))$

$\{f_{i,\lambda} \mid i = 1, 2, 3, 4\}$. The four points of intersection $\{\alpha_{ij} \mid j = (i \bmod 4) + 1,\ i = 1, 2, 3, 4\}$—one for every two consecutive curves, $f_{i,\lambda}$ and $f_{(i \bmod 4)+1,\lambda}$—defining the region $\mathscr{R}_{ak}$, are shown in Tables 9.1 and 9.2, an outline of the derivation being given in [107].

9.3.3.1 The Algorithm DCG

From Theorem 2, it is evident that for a particular positive value of k, if there exists a positive integer a such that the region $\mathscr{R}_{ak}$ is non-empty and contains at least one integer point, namely (a, r), then $\mathscr{C}^{\mathbb{Z}}(o, r)$ is the digital circle whose kth run-length is λ. Hence, in the algorithm DCG (Digital Circularity in General, Algorithm 3), we start with $k = k_{\min} = 1$ (Step 2) and compute all a's contained in $\mathscr{R}_{ak}$ using the procedure FIND-PARAMS (Procedure 1). For example, if $S[0] = 7$, then for $k = 1$, we get $\lceil \alpha_{23} \rceil = 9$ and $\lfloor \alpha_{41} \rfloor = 11$ using Table 9.1, whose physical significance is shown in Fig. 9.12. Let a possess m values, which are computed and stored in $A[0][0]$, $A[1][0], \ldots, A[m-1][0]$ (Step 4 of FIND-PARAMS). For each such value $A[i][0]$ of a, the corresponding range of radii lying in $\mathscr{R}_{ak}$ is stored in $A[i]$; the lower limit is stored in $A[i][1]$ and the upper limit in $A[i][2]$ (Steps 5–12 of FIND-PARAMS). Notice in Fig. 9.12 that for a given value of $\lambda(= 7)$, the valid radii or ranges of radius may not be contiguous. The circles with $r = 82, 83, \ldots, 86$, and with $r = 98, 99, 100$ do not have any run of length 7.

Algorithm 3: The decision algorithm DCG that verifies whether a sequence of n runs corresponds to a digital circle

Input: `int` $S[0..n-1]$
Output: Number of runs of S that are digitally circular; starting run from which circularity is satisfied; radius interval.

```
1  n_max ← 0;
2  for k' ← k_min to k_max do
3      Λ ← S[0], i ← 0;
4      FIND-PARAMS(A, Λ, k');
5      while i < m and n_max < n do // for all a's of first run
6          [s', s''] ← [r', r''] ← [A[i][1], A[i][2]];
7          Λ ← A[i][0] + S[0], j ← 1;
8          while j < n and s'' ≥ r' and s' ≤ r'' do // verify other n − 1
           runs
9              Λ ← Λ + S[j], k ← k' + j;
10             s' ← ⌈((Λ−1)^2 + k(k+1) + 1) / (2k+1)⌉;
11             s'' ← ⌊(Λ^2 + k(k+1)) / (2k+1)⌋;
12             if s'' ≥ r' and s' ≤ r'' then
13                 [r', r''] ← [max(r', s'), min(r'', s'')];
14                 j ← j + 1
15         i ← i + 1;
16         if n_max < j then
17             n_max ← j, k_off ← k', [r_min, r_max] ← [r', r'']
18 print n_max; k_off; [r_min, r_max]
```

For each $a = A[i][0]$ (Step 5 of DCG) corresponding to the first run $S[0]$ of the digital curve segment S, the corresponding radius range $[r', r'']$ and the related shift of run-length given by $a = A[i][0]$ are considered (Step 6 and Step 7 of DCG, respectively). For each such possible integer-tuple (a, k), the radius interval $[s', s'']$ for each of the subsequent run-lengths of S is computed (Steps 9–11). If $[s', s''] \cap [r', r''] \neq \emptyset$, then the effective radius range is updated to $[r', r'']$ (Step 13 of DCG). Otherwise, the current run does not belong to the same digital circle as of the preceding runs, and hence the maximum number or runs, denoted by j in DCG, satisfying digital circularity is stored to $n_{\max}$ (Steps 16 and 17), provided j is greater than the previous maximum, if any; the other output parameters, namely k_{off} and the radius interval $[r_{\min}, r_{\max}]$, are updated accordingly (Step 17).

In order to maximize the length of each segmented arc, we have to maximize the lengths of its leading and trailing runs, and so the algorithm DCG should be run on $S[1..n-2]$, as $S[0]$ is the leading and $S[n-1]$ is the trailing run. After obtaining $n_{\max}$, k_{off}, and $[r_{\min}, r_{\max}]$ in Step 17, Lemma 1 can be used to compute $k_{\text{off}} - 1$

Procedure 1: `Find-Params(int A, int Λ, int k)`

```
1  Compute {α_uv | 1 ⩽ u ⩽ 4 ∧ v = (u + 1) mod 4} // from Table 9.1;
2  i ← 0;
3  for a ← ⌈α23⌉ to ⌊α41⌋ do
4      A[i][0] ← a;
5      if a < α12 then
6          A[i][1] ← ⌈f2,λ(a, k)⌉
7      else
8          A[i][1] ← ⌈f1,λ(a, k)⌉
9      if a < α34 then
10         A[i][2] ← ⌊f3,λ(a, k)⌋
11     else
12         A[i][2] ← ⌊f4,λ(a, k)⌋
13     i ← i + 1;
14 m ← i
```

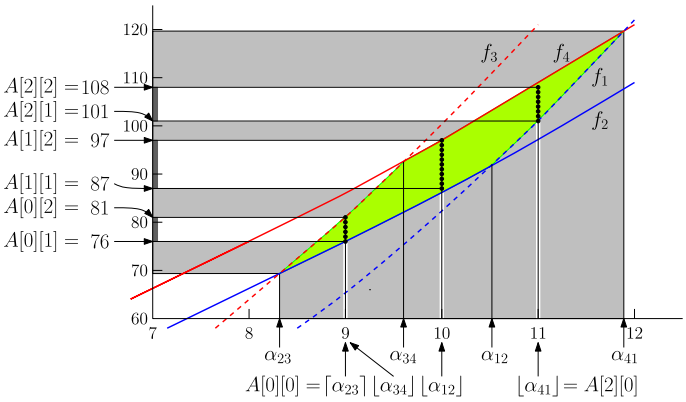


Fig. 9.12 Demonstration of the procedure FIND-PARAMS on a run-length 7 to obtain the solution space $\mathscr{R}_{ak}$ of the radius intervals $\{[r'_j, r''_j] \mid j = 0, 1, 2\}$ corresponding to $m = 3$ square numbers lying in the interval $[\lceil\alpha_{23}\rceil^2, \lfloor\alpha_{41}\rfloor^2] = [9^2, 11^2]$

and $k_{\text{off}} + n_{\max}$ corresponding to $r_{\max}$. Note that, $r_{\min}$ is not required to compute these run lengths, as we want to maximize the lengths of terminal runs, i.e., $S[0]$ and $S[n-1]$. The parts of $S[0]$ and $S[n-1]$ which are not in excess of $k_{\text{off}} - 1$ and $k_{\text{off}} + n_{\max}$ respectively, are accepted.

An interesting point regarding the changing nature of the range of a is that, as k increases, the range of a gets wider, thereby making the region $\mathscr{R}_{ak}$ possess more options for r. That is, the number of integer points (a, r) in $\mathscr{R}_{ak}$ goes on increasing with k, for a given value of λ. This, in turn, calls for more and more digital circles

to be tested when k is made higher, hence delaying the final solution. A feasible solution to arrest such delay has been designed by us using the concept of *infimum circle* and *supremum circle*, as explained in [107].

9.4 Polygonal Approximation

Extraction of the ADSS for each curve S_k in the given set (binary image) $\mathscr{I} := \{S_k\}_{k=1}^{K}$ of digital curve segments generates an ordered set of ADSS, namely $\mathscr{A}_k := \langle \mathbf{L}_i^{(k)} \rangle_{i=1}^{n_k}$, corresponding to S_k. In each such set $\mathscr{A}_k$, several consecutive ADSS may occur, which are approximately collinear and, therefore, may be combined together to form a single segment.

Let $\langle \mathbf{L}^{(k)} \rangle_{j_1}^{j_2}$ be the maximal (ordered) subset of the ADSS starting from $\mathbf{L}_{j_1}^{(k)}$ that conforms to some approximation criterion. Then these $j_2 - j_1 + 1$ segments in A_k are combined together to form a single straight line segment starting from the start point of $\mathbf{L}_{j_1}^{(k)}$ and ending at the end point of $\mathbf{L}_{j_2}^{(k)}$. This procedure is repeated for all such maximal subsets of $\mathscr{A}_k$ in succession to obtain the polygonal approximation (in case S_k is a closed curve) or polychain approximation (in case S_k is open), namely $\mathscr{P}_k$, corresponding to S_k.

In the proposed algorithm, depending on the approximation criterion, we have used a greedy method of approximating the concerned curve S_k starting from the very first ADSS in $\mathscr{A}_k$. Determination of a minimal set of DSS (and a minimal set $\mathscr{A}_k$ of ADSS, thereof) corresponding to a given curve S_k is known to be computationally intensive [84, 124]; so for real-time applications, a near-optimal but speedy solution is often preferred than the optimal one.

9.4.1 Approximation Criterion

There are several variants of approximation criteria available in the literature [125]. We have tested our algorithms with two variants of the approximation measures based on area deviation [145]. Both the algorithms are realizable in purely integer domain subject to few primitive operations only. The approximation criterion is defined w.r.t. the *approximation parameter* or *error tolerance*, denoted by τ, as follows.

9.4.1.1 Cumulative Error (Criterion $\mathbf{C}_{\sum}$)

Let $\langle \mathbf{L}^{(k)} \rangle_{j_1}^{j_2}$, be an ordered subset of A_k as discussed above. Then the ADSS ($j_2 - j_1 + 1$ in number) in A_k are replaced by a single straight line segment starting from

the start point of $\mathbf{L}_{j_1}^{(k)}$ and finishing at the end point of $\mathbf{L}_{j_2}^{(k)}$, if:

$$\sum_{j=j_1}^{j_2-1} \left|\triangle\left(s\left(\mathbf{L}_{j_1}^{(k)}\right), e\left(\mathbf{L}_{j}^{(k)}\right), e\left(\mathbf{L}_{j_2}^{(k)}\right)\right)\right| \leqslant \tau d_{\top}\left(s\left(\mathbf{L}_{j_1}^{(k)}\right), e\left(\mathbf{L}_{j_2}^{(k)}\right)\right) \tag{9.13}$$

where, $s(\mathbf{L}_j^{(k)})$ and $e(\mathbf{L}_j^{(k)})$ represent the respective start point and the end point of the ADSS $\mathbf{L}_j^{(k)}$, etc. The start point of $\mathbf{L}_j^{(k)}$ coincides with the end point of the preceding ADSS, if any, in $\mathscr{A}^k$, and the end point of $\mathbf{L}_j^{(k)}$ coincides with the succeeding one, if any. In Eq. (9.13), $|\triangle(p, q, r)|$ denotes twice the magnitude of area of the triangle with vertices $p := (x_p, y_p)$, $q := (x_q, y_q)$, and $r := (x_r, y_r)$, and $d_{\top}(p, q)$ the maximum isothetic distance between two points p and q. Since all these points are in two-dimensional digital space, the above measures are computable in the integer domain as shown in the following equations.

$$d_{\top}(p, q) = \max\left\{|x_p - x_q|, |y_p - y_q|\right\} \tag{9.14}$$

$$\triangle(p, q, r) = \begin{vmatrix} 1 & 1 & 1 \\ x_p & x_q & x_r \\ y_p & y_q & y_r \end{vmatrix} \tag{9.15}$$

From Eq. (9.15), it is evident that $\triangle(p, q, r)$ is a determinant that gives twice the signed area of the triangle with vertices p, q, and r. Hence the ADSS in the given subset are merged to form a single straight line segment, say $\widetilde{\mathbf{L}}$, provided the cumulative area of the triangles ($j_2 - j_1$ in number), having $\widetilde{\mathbf{L}}$ as base and the third vertices being the end points of the ADSS (excepting the last one) in the subset $\langle \mathbf{L}^{(k)} \rangle_{j_1}^{j_2}$, does not exceed the area of the triangle with base $\widetilde{\mathbf{L}}$ (isothetic length) and height τ.

9.4.1.2 Maximum Error (Criterion $\mathbf{C_{max}}$)

With similar notations as mentioned above, using the maximum error criterion, the ADSS in $\langle \mathbf{L}^{(k)} \rangle_{j_1}^{j_2}$ would be replaced by a single piece, provided the following condition is satisfied.

$$\max_{j_1 \leqslant j \leqslant j_2-1} \left|\triangle\left(s\left(\mathbf{L}_{j_1}^{(k)}\right), e\left(\mathbf{L}_{j}^{(k)}\right), e\left(\mathbf{L}_{j_2}^{(k)}\right)\right)\right| \leqslant \tau d_{\top}\left(s\left(\mathbf{L}_{j_1}^{(k)}\right), e\left(\mathbf{L}_{j_2}^{(k)}\right)\right) \tag{9.16}$$

The rationale of considering two such criteria is as follows. Since we would be replacing a number of ADSS, which are almost straight, and more importantly, are not ordinary digital curves of arbitrary patterns and arbitrarily curvatures, the end point of each ADSS makes a triangle with the replacing segment, namely $\widetilde{\mathbf{L}}$. So the sum of the areas of triangles formed by the end points of these ADSS in combination with the replacing line $\widetilde{\mathbf{L}}$ gives a measure of error due to approximation of *all ADSS in* $\langle \mathbf{L}^{(k)} \rangle_{j_1}^{j_2}$ by $\widetilde{\mathbf{L}}$. Alternatively, if we are guided by the worst case approximation, that is, if the mostly digressing ADSS is considered to estimate the error, then the

maximum of the areas of these triangles should be considered as the error measure for approximation of *worst ADSS in* $\langle \mathbf{L}^{(k)} \rangle_{j_1}^{j_2}$ by $\widetilde{\mathbf{L}}$.

Empirical observations as reported in [14], reveal that the above two criteria are essentially similar in the sense that they produce almost identical polygons for different digital curve segments for different values of the error tolerance (i.e., τ). This is quite expected as far as the output is concerned.

As mentioned earlier in Sect. 9.4, to construct polygonal approximation we consider the start point of the first ADSS (i.e., $\mathbf{L}_{j_1}^{(k)}$), and the end point of the last ADSS (i.e., $\mathbf{L}_{j_2}^{(k)}$). This can be justified as follows.

- The sum (for criterion $\mathrm{C}_{\sum}$) or the maximum (for criterion $\mathrm{C}_{\max}$) of the isothetic distances of the end points of each ADSS from the replacing line $\widetilde{\mathbf{L}}$ never exceeds the specified error tolerance τ. This follows easily on expansion of the left hand side of the corresponding Eqs. (9.13) and (9.16), and from the fact that the term $d_{\top}(s(\mathbf{L}_{j_1}^{(k)}), e(\mathbf{L}_{j_2}^{(k)}))$ represents the isothetic length of $\widetilde{\mathbf{L}}$.
- Since each ADSS $\mathbf{L}_j^{(k)}$ is approximately a DSS, we consider that $\nexists p \in \mathbf{L}_j^{(k)}$ such that the isothetic distance of p from DSL passing through the end points of $\mathbf{L}_j^{(k)}$ exceeds unity (as testified in our experiments). Although for sufficiently long ADSS, this may not hold for the underlying conditions (c1) and (c2); however, in our experiments with real world images, this was found to hold. In the case of any violation, some heuristics may be employed to find the error points and to find smaller ADSS to resolve the problem.

9.4.2 Algorithm for Polygonal Approximation

The algorithm for polygonal approximation of a sequence of ADSS in the set $\mathscr{A}$, using the approximation criterion of Eq. (9.13), is described in Algorithm 4. To take care of the criterion $\mathrm{C}_{\max}$ of Eq. (9.16), a similar procedure may be written.

Final time complexity As explained in Sect. 9.2.4, the time complexity for extracting the ADSS in a set of digital curve segments, $\mathscr{I} := \{S_k\}_{k=1}^{K}$, is given by $\Theta(N_1) + \Theta(N_2) + \cdots + \Theta(N_K) = \Theta(N)$, where $N (= N_1 + N_2 + \cdots + N_K)$ is the total number of points representing $\mathscr{I}$. Now, in the algorithm MERGE-ADSS, we have considered only the ordered set of vertices of the ADSS corresponding to the curves, so that the worst-case time complexity in this stage is linear in N. Hence, the overall time complexity is given by $\Theta(N) + \mathscr{O}(N) = \Theta(N)$, whatsoever may be the error of approximation τ.

9.4.3 Quality of Approximation

The goodness of an algorithm for polygonal approximation is quantified, in general, by the amount of discrepancy between the approximate polygon(s) (or polychain(s))

Algorithm 4: MERGE-ADSS for polygonal approximation of a sequence of ADSS in $\mathscr{A}$ using criterion C_{max}. (Source: [14])

Input: $\mathscr{A}, n, \tau$
Output: Vertices of polygonal approximation in $\mathscr{A}$
1. **for** $m \leftarrow 1$ to n
2. **for** $S \leftarrow 0, i \leftarrow 1$ to $(n - m - 1)$
3. $S \leftarrow S + \triangle(\mathscr{A}[m], \mathscr{A}[m+i], \mathscr{A}[m+i+1])$
4. $dx \leftarrow |\mathscr{A}[m].x - \mathscr{A}[m+i+1].x|$
5. $dy \leftarrow |\mathscr{A}[m].y - \mathscr{A}[m+i+1].y|$
6. $d \leftarrow \max\{dx, dy\}$
7. **if** $S \leqslant d\tau$
8. delete $\mathscr{A}[m+i]$ from $\mathscr{A}$
9. **else**
10. **break**
11. $m \leftarrow m + i - 1$

and the original set of digital curve segments. There are several measures to assess the approximation of a curve S_k, such as

- compression ratio $CR = N_k/M_k$, where N_k is the number of points in S_k and M_k is the number of vertices in the approximate polygon $\mathscr{P}_k$;
- the integral square error (ISE) between S_k and $\mathscr{P}_k$.

Since there is always a trade-off between CR and ISE, other measures may also be used [69, 126, 133]. These measures, however, may not always be suitable for some intricate approximation criterion. For example, the figure of merit [133], given by FOM = CR/ISE, may not be suitable for comparing approximations for some common cases, as shown by [125]. In a work by [143], the percentage relative difference, given by $((E_{approx} - E_{opt})/E_{opt}) \times 100$, has been used, where E_{approx} is the error incurred by a suboptimal algorithm under consideration, and E_{opt} the error incurred by the optimal algorithm, under the assumption that same number of vertices are produced by both the algorithms. Similarly, one may use two components, namely fidelity and efficiency, given by $(E_{opt}/E_{approx}) \times 100$ and $(M_{opt}/M_{approx}) \times 100$ respectively, where M_{approx} is the number of vertices in the approximating polygon produced by the suboptimal algorithm and M_{opt} is the same produced by the optimal algorithm subject to same E_{approx} as the suboptimal one [125].

The algorithm proposed here is not constrained by the number of vertices M_k of the output polygon $\mathscr{P}_k$, and therefore, the measures of approximation where M_k acts as an invariant, are not applicable. Instead, we have considered the error of approximation, namely τ, as the sole parameter in our algorithm, depending on which the number of vertices M_k corresponding to $\mathscr{P}_k$ will change. A high value of τ indicates a loose or slacked approximation, whence the number of vertices M_k decreases automatically, whereas a low value of τ implies a tight approximation,

thereby increasing the number of vertices in the approximate polygon. Hence, in accordance to the usage of τ in both of our proposed methods, one based on criterion $C_{\sum}$ and the other on $C_{\max}$, the total number of vertices $M := M_1 + M_2 + \cdots + M_K$ in set of approximate polygons $\{\mathscr{P}_k\}_{k=1}^{K}$ corresponding to the input set of digital curve segments, namely $\mathscr{I} := \{S_k\}_{k=1}^{K}$, versus τ, provides the necessary quality of approximation. Since the total number of points lying on all the points in $\mathscr{I}$ characterizes (to some extent) the complexity of $\mathscr{I}$, we consider the compression ratio (CR) as a possible measure of approximation.

Another measure of approximation is given by how much a particular point $(x, y) \in S_k \in \mathscr{I}$ has deviated in the corresponding polygon $\mathscr{P}_k$. If $\tilde{p} := (\tilde{x}, \tilde{y})$ be the point in $\mathscr{P}_k$ corresponding to $p := (x, y)$ in $\mathscr{I}$, then for all points in $\mathscr{I}$, this measure is captured by the variation of the number of points with isothetic deviation $d_\perp$ w.r.t. $d_\perp$, where the (*isothetic*) *deviation from* p *to* $\tilde{p}$ is given by

$$dev_\perp(p \to \tilde{p}) = \min\{|x - \tilde{x}|, |y - \tilde{y}|\}. \tag{9.17}$$

Further, since $dev_\perp(p \to \tilde{p})$ depends on the chosen value of τ in our algorithm, the fraction of the number of points in $\mathscr{I}$ with deviation $d_\perp$ varies plausibly with τ. So, the *isothetic error frequency* (IEF) (or, simply *error frequency*), given by

$$f(\tau, d_\perp) = \frac{1}{N}\big|\{p \in \mathscr{I}\colon dev_\perp(p \to \tilde{p}) = d_\perp\}\big|, \tag{9.18}$$

versus τ and $d_\perp$, acts as the second measure that provides the error distribution for the polygonal approximation of $\mathscr{I}$.

It may be observed that, the error frequency distribution in Eq. (9.18) is equivalent to the probability density function, and satisfies the criterion

$$\sum_{d_\perp} f(\tau, d_\perp) = 1, \quad \text{for } \tau = 0, 1, 2, \ldots.$$

In fact, the error frequency distribution function in our measure is a bivariate distribution of (finite-size) samples of size N, depending on the two variables, namely τ and $d_\perp$. A study on the nature of the error frequency distribution for a sufficiently large population of arbitrary digital curves may be, therefore, a promising area of theoretical analysis of polygonal approximation of digital curves.

9.4.4 Experimental Results

In Fig. 9.13, we have presented the comparative results on polygonal approximation of a benchmark curve, "chromosome" [142]. Our implementation is in C in Intel Core 2 Duo CPU E4500 2.20 GHz, Mandriva Linux 2008. The results show that MERGE-ADSS compares favorably with others when we consider $\tau = 1$ in MERGE-ADSS. For $\tau = 2, 3, \ldots$, the approximation obtained by MERGE-ADSS requires fewer number of vertices (M), but at the cost of some error incurred, and may not be profitable for small curves like "chromosome" (and other test/benchmark

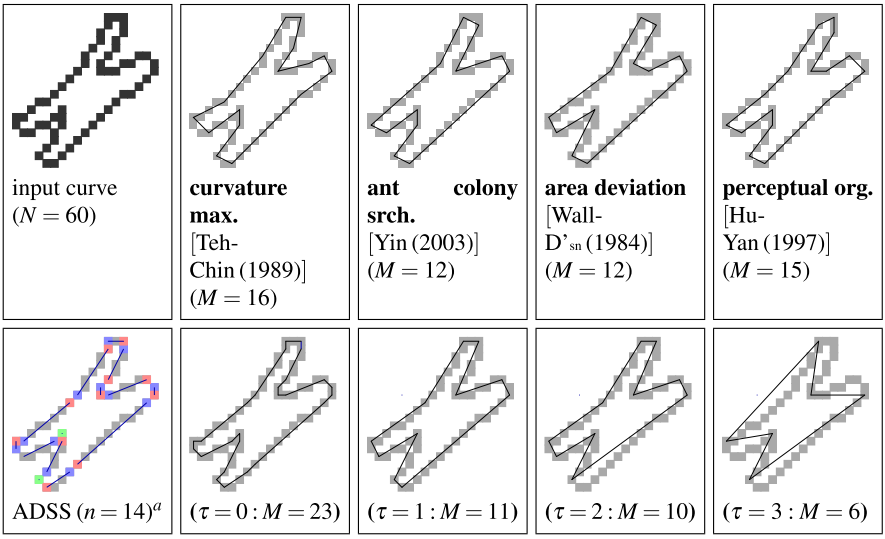


Fig. 9.13 Results of polygonal approximations on "chromosome" by some existing methods and MERGE-ADSS (using criterion $C_{\sum}$), after applying EXTRACT-ADSS. [a]A *blue point* indicates the start point and a *red point* indicates the end point of an ADSS. A *green point* indicates an ADSS with its end point coinciding with its start point, which is a degenerate case arising out of our consideration of the chain code of a start point w.r.t. the end point of the previous ADSS in the sequence. (Source: [14])

Table 9.3 Comparison of DSS and ADSS extraction algorithms on different images. (Source: [14])

Image name & size (pixels)	No. of points	P.A.[a] (s)	No. of segs.		Avg. length		CPU time (s)		
			DSS	ADSS	DSS	ADSS	DSS	ADSS	$\{\mathscr{P}\}$[b]
bird-nestlings 480×320	3041	6.38	902	327	3.37	9.30	5.42	0.17	0.05
climber 320 × 330	2750	5.92	1170	419	2.35	6.56	6.74	0.20	0.05
India 325 × 285	2552	7.15	1735	597	1.47	4.27	9.06	0.29	0.06
leaf 240 × 256	1312	3.88	341	106	3.85	12.38	2.17	0.08	0.02
spider 292 × 286	1767	4.20	583	157	3.03	11.25	3.93	0.11	0.03
test-001 140 × 1050	2809	6.26	858	276	3.27	10.18	4.48	0.14	0.04
vase 408 × 333	6066	10.81	1972	681	3.07	8.91	14.52	0.43	0.10

[a]CPU time for polygonal approximation using area deviation [145] without ADSS

[b]Average CPU time for polygonal approximation with ADSS using criteria $C_{\sum}$ and C_{max}, and $\tau = 1$–20

curves considered in the existing works [125, 142, 145, 149, 155]). However, for sufficiently large curve segments, slackening of τ reduces the number of vertices to the desired limit, as evident from Fig. 9.14. Table 9.3 shows results on some test images for tolerance varying from 1 to 20.

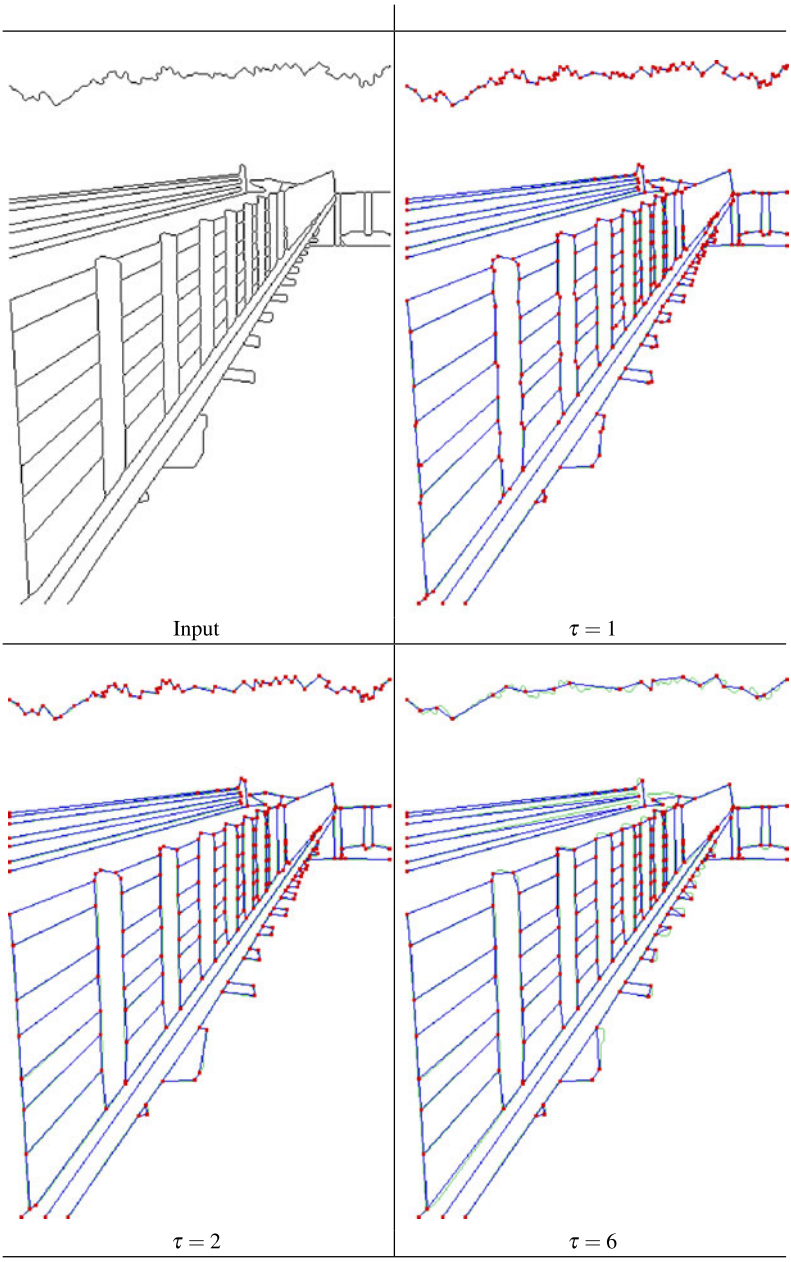


Fig. 9.14 Results of polygonal approximation on a real-world image (after edge detection and thinning) by EXTRACT-ADSS, followed by MERGE-ADSS (using $C_{\sum}$)

9.5 Circular Arc Segmentation

Fast and accurate recognition of circles or circular arcs in a digital image is a challenging problem with practical relevance to computer vision applications as well as in medical imaging. In Sect. 9.3, we have given a brief overview of different techniques. Out of all these techniques, Hough transform (HT) is mostly followed in some or other variant. Though HT is robust against noises, clutters, object defects, and shape distortions, it often requires intensive computation and a large amount of memory.

In this section, we discuss a technique of recognizing digital circles as well as circular arcs, based on the *chord property* and the *sagitta property* [146]. A preliminary version of this may be seen in [12]. All the arc segments are first extracted, and then using the *chord property*, the circularity of each arc segment is verified and the circular segments are identified. The *sagitta property* is applied to determine the radii of the circular arc segments, and in turn, the corresponding centers. Finally, two arc segments with closest radii and centers are merged iteratively to obtain a complete circle or a larger circular arc segment. To improve the accuracy of computing the centers and radii, a technique based on restricted Hough transform (RHT) is used.

In real-world images, a digital circle or a circular arc may intersects other circular arcs or digital straight line segments. To handle such cases, first we need to detect the digital circular segments separately and then merge them efficiently to form a complete digital circle or a larger circular arc segment. Further, as the contour may be given as thick curve segments, we use thinning [63] as preprocessing before applying the algorithm. The subsequent steps may be briefed as follows.

9.5.1 Finding the Intersection Points

In order to detect each segment separately, first of all we detect all the points of intersection (among the digital curve segments) and end points (for open digital curves), and store them in a list $\mathscr{P}$. As we detect circular segments first and then merge them to form a complete circle or a larger circular segment, we have to do some special treatment for a free/isolated closed curve. Consider S to be a free and closed digital curve segment. We put two virtual points of intersection, say, $p_1 \in S$ and $p_2 \in S$, such that, if S_1 and S_2 be the two resultant segments ($S_1 \cup S_2 = S$) whose (virtual) end points are p_1 and p_2, then the lengths of S_1 and S_2 differ by at most unity.

9.5.2 Storing the Curve Segments

A *thin digital curve segment* is a set of pixels having two end pixels and a minimal list of pixels that establish connectivity between the end pixels using 8-neighbor

rule [63]. For each point $p_i \in \mathscr{P}$, the corresponding segment(s) incident at p_i is/are extracted. If p_i is an end point of a digital curve segment S, then there is only one segment, i.e., S, incident at p_i. If p_i is a virtual point of intersection created for a free and closed digital curve, namely $S = S_1 \cup S_2$, then there are two segments, i.e., S_1 and S_2, incident at p_i. And number of digital curve segments incident at p_i is three or more if and only if p_i is an actual point of intersection between two or more digital curve segments.

To discard the spurious segments, we consider the length of (i.e., number of digital points constituting) each segment incident at each $p_i \in \mathscr{P}$. If a segment is negligibly small (10 pixels or less), then it is discarded; otherwise, it is stored in a list of segments, $\mathscr{L}$. Each node of the list contains the (coordinates of) two endpoints, the center and the radius of the circular arc if obtained, and a pointer to the linked list of the concerned curve points. To identify the circular arc segments, we need to first remove the digital straight line segments from the list $\mathscr{L}$. It may be mentioned here that there are several techniques to determine digital straightness [14, 84, 85, 125]. We have used the concept of area deviation [145], which is realizable in purely integer domain using a few primitive operations only. The method is as follows.

Let $S := \langle a = c_1, c_2, \ldots, c_k = b \rangle$ be a digital curve segment with end points a and b. Let c_i $(2 \leqslant i \leqslant k-1)$ be any point on the segment S other than a and b. Let h_i be the distance of the point c_i from the real straight line segment $\overline{ab}$. Then S is considered to be a single digital straight line segment starting from a and ending at b, provided

$$\max_{2 \leqslant i \leqslant k-1} \left| \triangle(a, c_i, b) \right| \leqslant \tau_h d_\top(a, b). \tag{9.19}$$

Here, $|\triangle(a, c_i, b)|$ denotes twice the magnitude of area of the triangle with vertices $a := (x_1, y_1)$, $c_i := (x_i, y_i)$, and $b := (x_k, y_k)$, and $d_\top(a, b) := \max(|x_1 - x_k|, |y_1 - y_k|)$ is the maximum isothetic distance between a and b.

9.5.3 Deviations of Chord Property

Let θ_m be the angle subtended by (chord) $\overline{ab}$ at the midpoint m of a segment $S \in \mathscr{L}$ with end points a and b. Let θ_c be the angle subtended by $\overline{ab}$ at an arbitrary point $c \in S \setminus \{a, b\}$. Then, according to the *chord property*, $\theta_c = \theta_m$ if the segment S is a part of the Euclidean (real) circle. But this is not exactly true for a digital circle. Hence, we use a variant of this useful chord property (Fig. 9.15). If $\theta_c \in [\theta_m - \varepsilon, \theta_m + \varepsilon]$ for all $c \in S$ excepting a few points near its two ends, ε being a small positive quantity (Fig. 9.15), then the digital curve S is circular.

Let $\mathscr{C}^{\mathbb{R}}(o, r)$ be the real circle centered at $o(0, 0)$ and having radius $r \in \mathbb{Z}^+$. Let $\mathscr{A}^{\mathbb{R}}(\alpha, \beta)$ be an arc of $\mathscr{C}^{\mathbb{R}}(o, r)$ having end points $\alpha(x_\alpha, y_\alpha) \in \mathscr{C}^{\mathbb{R}}(o, r)$ and $\beta(x_\beta, y_\beta) \in \mathscr{C}^{\mathbb{R}}(o, r)$, such that $x_\alpha, x_\beta \in \mathbb{Z}$. Let $\gamma(x_\gamma, y_\gamma) \in \mathbb{R}^2$ be an arbitrary point in $\mathscr{A}^{\mathbb{R}}(\alpha, \beta) \setminus \{\alpha, \beta\}$. Let the chord $\overline{\alpha\beta}$ subtends an angle ϕ_γ at γ. Let $a, b, c \in \mathbb{Z}^2$ be the respective points in the digital circle $\mathscr{C}^{\mathbb{Z}}(o, r)$ corresponding to α, β, γ, and the angle subtended by the line segment $\overline{ab}$ at c be ϕ_c.

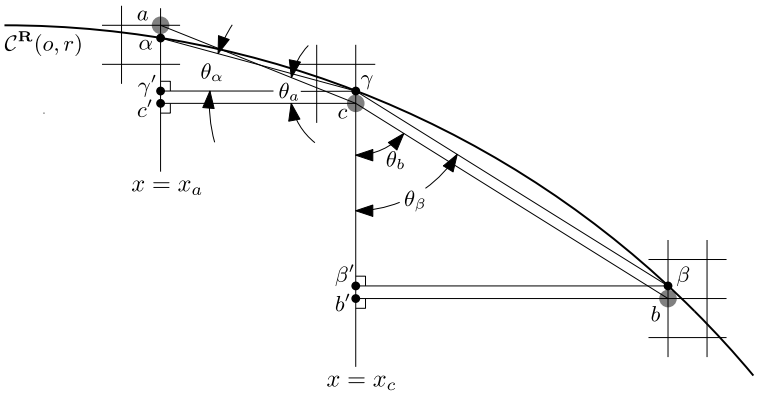


Fig. 9.15 Deviation of the chord property. Points in $\mathbb{R}^2$ $(\alpha, \beta, \ldots)$ or in the real circle $\mathscr{C}^{\mathbb{R}}(o,r)$ are shown in *black*, and points shown as larger *gray blobs* (a, b, c) belong to the digital circle, $\mathscr{C}^{\mathbb{Z}}(o,r)$. $\mathscr{A}^{\mathbb{R}}(\alpha,\beta)$ is an arc of $\mathscr{C}^{\mathbb{R}}(o,r)$ in Octant 1, which corresponds to the given digital (*circular*) arc $\mathscr{A}^{\mathbb{Z}}(a,b)$. As c changes its place along $\mathscr{A}^{\mathbb{Z}}(a,b)$ such that $y_\gamma - \frac{1}{2} < y_c < y_\gamma + \frac{1}{2}$, the angle $\phi_c (= \theta_a + \theta_b + \pi/2)$ gets deviated by $(+/-)\varepsilon$

Now, consider that $\mathscr{A}^{\mathbb{R}}(\alpha,\beta)$ is an arc of $\mathscr{C}^{\mathbb{R}}(o,r)$ in Octant 1, which corresponds to the digital (circular) arc $\mathscr{A}^{\mathbb{Z}}(a,b)$. Since $a(x_a, y_a)$, $b(x_b, y_b)$, and $c(x_c, y_c)$ are the respective points of $\mathscr{C}^{\mathbb{Z}}(o,r)$ corresponding to $\alpha(x_\alpha, y_\alpha)$, $\beta(x_\beta, y_\beta)$, and $\gamma(x_\gamma, y_\gamma)$ of $\mathscr{C}^{\mathbb{R}}(o,r)$, we have $x_\alpha = x_a \in \mathbb{Z}$, $x_\beta = x_b \in \mathbb{Z}$, and $x_\gamma = x_c \in \mathbb{Z}$ (Fig. 9.15). Further, owing to the digitization scheme, we have

$$\begin{aligned} y_\alpha - \frac{1}{2} < y_a < y_\alpha + \frac{1}{2}, \qquad y_\beta - \frac{1}{2} < y_b < y_\beta + \frac{1}{2}, \\ y_\gamma - \frac{1}{2} < y_c < y_\gamma + \frac{1}{2}. \end{aligned} \tag{9.20}$$

Let c' and γ' be the respective feet of perpendiculars dropped from c and γ to the vertical line $x = x_a$, and b' and β' be those from b and β to the vertical line $x = x_c$. Let $\theta_a, \theta_\alpha, \theta_b$, and θ_β be the acute angles subtended at c and γ by the corresponding perpendiculars. Then, $\phi_c = \theta_a + \theta_b + \pi/2$ and $\phi_\gamma = \theta_\alpha + \theta_\beta + \pi/2$. Clearly, for $c \in \mathscr{A}^{\mathbb{Z}}(a,b)$, we have

$$\max(\phi_c) = \max(\theta_a + \theta_b) + \pi/2 \leqslant \max(\theta_a) + \max(\theta_b) + \pi/2, \tag{9.21}$$

$$\min(\phi_c) = \min(\theta_a + \theta_b) + \pi/2 \geqslant \min(\theta_a) + \min(\theta_b) + \pi/2, \tag{9.22}$$

or,

$$\phi_c \in \big[\min(\theta_a) + \min(\theta_b) + \pi/2, \max(\theta_a) + \max(\theta_b) + \pi/2\big]. \tag{9.23}$$

Let $\delta_{y_{ac}} = y_a - y_c$, $\delta_{x_{ca}} = x_c - x_a$, and $\delta_{y_{\alpha\gamma}} = y_\alpha - y_\gamma$. Then the *deviation of* θ_a *from* θ_α is given by

$$\delta_{\theta_{a\alpha}} = \theta_a - \theta_\alpha = \tan^{-1}\frac{y_a - y_c}{x_c - x_a} - \tan^{-1}\frac{y_\alpha - y_\gamma}{x_\gamma - x_\alpha} \tag{9.24}$$

$$= \tan^{-1}\frac{\delta_{y_{ac}}}{\delta_{x_{ca}}} - \tan^{-1}\frac{\delta_{y_{\alpha\gamma}}}{\delta_{x_{ca}}} \quad (\text{since } x_a = x_\alpha \text{ and } x_c = x_\gamma)$$

$$= \tan^{-1}\left(\frac{(\delta_{y_{ac}} - \delta_{y_{\alpha\gamma}})\delta_{x_{ca}}}{\delta_{x_{ca}}^2 + \delta_{y_{ac}}\delta_{y_{\alpha\gamma}}}\right) < \tan^{-1}\left(\frac{\delta_{x_{ca}}}{\delta_{x_{ca}}^2 + \delta_{y_{ac}}\delta_{y_{\alpha\gamma}}}\right) \quad (\text{Eq. (9.20)}) \tag{9.25}$$

$$\approx \tan^{-1}\left(\frac{\delta_{x_{ca}}}{\delta_{x_{ca}}^2 + \delta_{y_{ac}}^2}\right) = \tan^{-1}\frac{1}{|\overline{ca}|} \quad (\text{with } \delta_{y_{ac}} \approx \delta_{y_{\alpha\gamma}}). \tag{9.26}$$

Thus, the deviation of θ_a from θ_α is at most $\tan^{-1}(1/\overline{ca})$, and higher the distance of a from c, lesser is the deviation. Similar deviation, namely $\delta_{\theta_{b\beta}}$, also comes into play while considering θ_b, and as the distance of c from b increases, the deviation becomes insignificant. Hence, if m be the middle pixel (one of two, if there are two such) of $\mathscr{A}^{\mathbb{Z}}(a, b)$, then the maximum possible deviation of ϕ_m from ϕ_γ is given by

$$\tau_\phi = 2\tan^{-1}\frac{1}{|\overline{am}|} = 2\tan^{-1}\frac{2}{|\overline{ab}|}. \tag{9.27}$$

For practical cases, the distance of c or m from a or b is quite low, and hence such deviations have to be considered for proper results. Our algorithm possesses this feature, resulting to its satisfactory performance in terms of both precision and robustness.

9.5.4 Verifying the Circularity

The list $\mathscr{L}$ contains segments which are not digitally straight, as explained in Sect. 9.5.2. That is, each segment S in $\mathscr{L}$ is made of at least one circular segment with or without one or many intervening straight pieces. So, for each segment S, we check its circularity using the *chord property* as explained in Sect. 9.5.3. If the segment S consists of both circular and straight components, then we extract its circular part(s) only from S, store these circular segment(s) in the list $\mathscr{L}$ with necessary updates, and remove the original segment S from $\mathscr{L}$.

We start checking the circularity of $S := \langle a = c_1, c_2, \ldots, c_k = b\rangle$ starting from the end point, a. We consider an *appropriately small prefix of* S, namely $S_j := \langle a = c_1, c_2, \ldots, c_j\rangle$, where $j = \min(\tau_s, k)$, and verify the circularity for one-third of the points lying in the central region of S_j, namely

$$S_j^{(m)} := \langle c_{\lfloor j/3\rfloor}, c_{\lfloor j/3\rfloor+1}, \ldots, c_{m-1}, c_m, c_{m+1}, \ldots, c_{2\lfloor j/3\rfloor-1}, c_{2\lfloor j/3\rfloor}\rangle.$$

In our experiments, we have considered $\tau_s = 15$. Two-third points (one-third from either end) of S are discounted from circularity test as they are prone to excessive deviations of chord property, as explained in Sect. 9.5.3. Hence, if c_m ($m = \lfloor j/2\rfloor$) be the midpoint of S_j and the angle subtended by the chord $\overline{ac_j}$ at m is estimated to

be ϕ_m, then S_j is considered to be satisfying the chord property, provided the angles ϕ_c subtended by $\overline{ac_j}$ at all points $c \in S_j^{(m)} \setminus \{a, c_j\}$ satisfy the following equation.

$$\max_{c \in S_j^{(m)}} \left\{ |\phi_c - \phi_m| \right\} \leqslant \tau_\phi. \tag{9.28}$$

If S_j is found to be circular, then we augment it to $S_{j'} := \langle a = c_1, c_2, \ldots, c_{j'} \rangle$, where $j' = \min(j + \lfloor \frac{1}{3}\tau_s \rfloor, k)$, in order to include the next $\lfloor \frac{1}{3}\tau_s \rfloor$ pixels from S, and again verify the chord property for $S_{j'}^{(m')}$ with $m' = \lfloor j'/2 \rfloor$. The process is continued until all points in S are verified or the chord property fails for some prefix S_j (or $S_{j'}$) of S.

9.5.5 Combining the Arcs

If S is a circular arc with end points a and b, then its *sagitta* is the straight line segment drawn perpendicular to the chord $\overline{ab}$, which connects the midpoint μ of ab with S. The *sagitta property* is as follows: If the perpendicular to ab at μ intersects S at s, then the radius of the circle whose arc is S, is given by

$$r = \frac{d^2(a, b)}{8d(\mu, s)} + \frac{d(\mu, s)}{2} \tag{9.29}$$

where, $d(a, b)$ denotes the Euclidean distance between the points a and b.

The radius, and hence the center, of each circular arc $S \in \mathscr{L}$ are computed using the *sagitta property*, and stored in the node corresponding to S. While combining the circular arcs, necessary care has to be taken for the inevitable error that creeps in owing to the usage of sagitta property, which is a property of real circles only. Since we deal with digital curve segments, the *cumulative error* of the effective radius computed for a combined/growing circular arc using the aforesaid sagitta property is very likely to increase with an increase in the number of segments constituting that arc. Hence, to enhance accuracy, we merge two digital circular segments S and S' into $S'' := S \cup S'$, if (i) S and S' have a common end point in $\mathscr{P}$ and (ii) S'' satisfies the *chord property*. Since the node corresponding to each segment in the list $\mathscr{L}$ contains end points, center, radius, and a pointer to the list of curve points, the attributes of the segment S are updated by those of S'', and the data structure $\mathscr{P}$ is updated accordingly.

9.5.6 Finalizing the Centers and Radii

In spite of the treatments to reduce discretizations error while employing chord property to detect circular arcs and while employing sagitta property to combine two or more circular arcs and compute the effective radius and center, some error may

still be present in the estimated values of the radii and the center. To remove such errors, we apply a restricted Hough transform (RHT) on each circular arc $S \in \mathscr{L}$ with a small parameter space [34]. Let $q(x_q, y_q)$ and r be the respective center and radius of S estimated using the sagitta property. Then the restricted parameter space is taken as $[x_q - \delta, x_q + \delta] \times [y_q - \delta, y_q + \delta] \times [r - \delta, r + \delta]$, such that $\delta = \tau_H r$, where $0 < \tau_H < 1$ ($\tau_H = \frac{1}{8}$ in our experiments). A 3D integer array, H, is taken corresponding to this parameter space, each of whose entry is initialized to zero. For every three distinct and non-collinear points c, c', and c'' from S, we estimate the center $q'(x', y')$ and the radius r' of the (real) circle passing through c, c', and c''. If each of $rd(x')$, $rd(y')$, and $rd(r')$ lies within the corresponding bounds of H, then the entry in H corresponding to $rd(x', y', r')$ is incremented ($rd(x') = \lfloor x' + \frac{1}{2} \rfloor$, etc.). Finally, the entry in H corresponding to the maximum frequency provides the final center and radius of S.

9.5.7 Some Results

A demonstration of the proposed method on a sample image (`6.pbm`) is shown in Fig. 9.16. All the digital curve segments in the image are extracted and stored in the list $\mathscr{L}$. At first, the straight line segments are removed from $\mathscr{L}$ with a small computation time. For example, $L(f(304, 45), o(304, 94))$, $L((304, 94), (303, 155))$, etc. are some of the straight line segments that are removed.[3] Then using the *chord property*, the circular segments are detected with necessary updates in the list $\mathscr{L}$. For example, the digital curve segment from $d(174, 174)$ to $b(199, 293)$ consists of two circular segments. Those two circular segments are extracted; one segment from $d(174, 174)$ to $c(199, 249)$ is stored in the node of the original segment and the other one from $c(199, 250)$ to $b(199, 293)$ is stored in a newly created node and inserted in the list $\mathscr{L}$. Similarly, the segment from $m(223, 174)$ to $(307, 349)$ consists of a circular arc and a straight line segment. From this segment, we extract the circular part and remove the straight part. Necessary updates in $\mathscr{L}$ and $\mathscr{P}$ are made. The circular segments in the list L are shown in Fig. 9.16(d) by different colors and are enumerated in Table 9.4.

Two or more smaller adjacent arcs are combined if they jointly satisfy the *chord property* in order to get larger arcs for reducing the computational error in the next step of applying the *sagitta property*. After such combining/merging, the number of circular segments gets reduced to almost 50%, as reflected in Table 9.4 and Fig. 9.16(e). Next, the radius and the center of each arc in $\mathscr{L}$ are computed using the *sagitta property* and stored in the node of the corresponding arc. Figure 9.16(f) shows the center of each circular arc as $+$. The detailed information of the circular segments stored in the list $\mathscr{L}$ is given in Table 9.4. The radius and center of the

[3] Here we use $L((x, y), (x', y'))$ to denote the digital straight line segment joining the points (x, y) and (x', y').

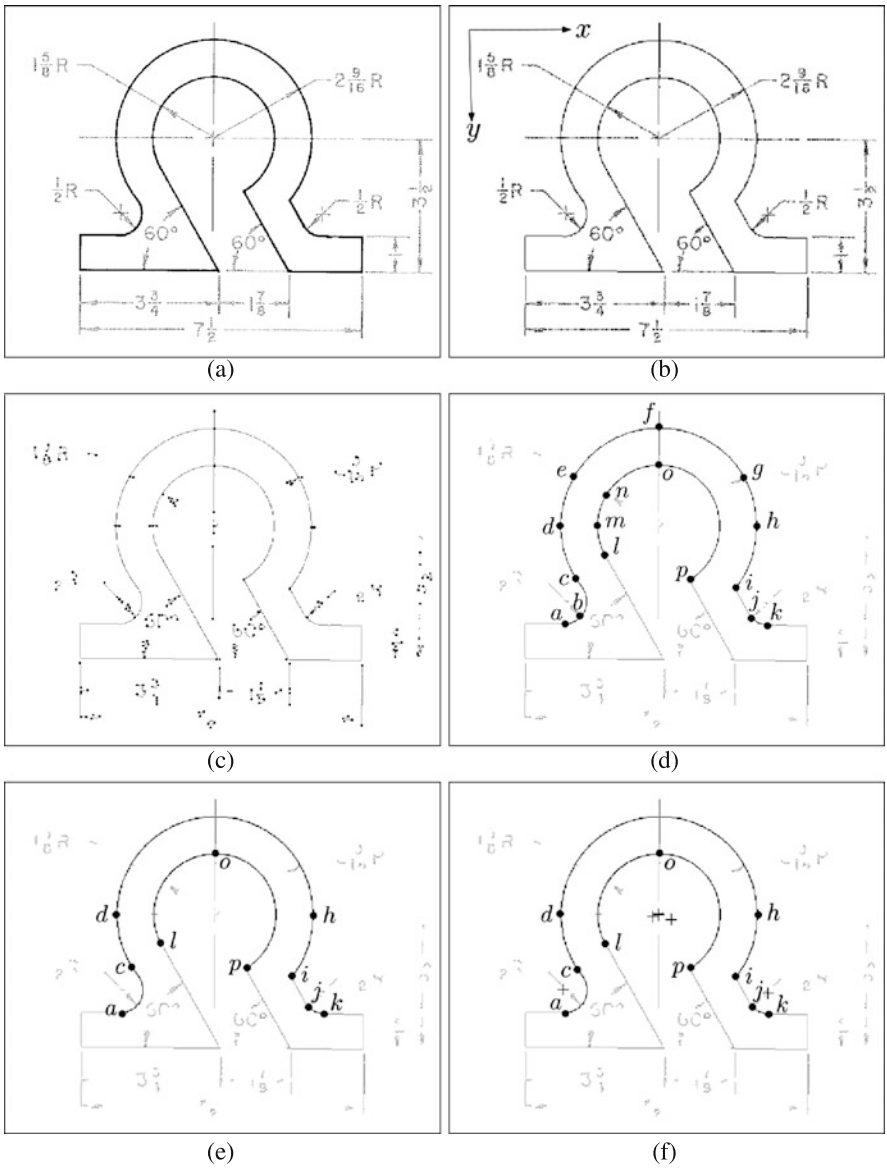


Fig. 9.16 Step-wise snapshots of the algorithm on `6.pbm`: (**a**) input; (**b**) after thinning; (**c**) intersection points and end points in L, removing small arcs; (**d**) detected circular arcs by *chord property*; (**e**) after combining adjacent arcs; (**f**) centers detected by *sagitta property*; (**g**) after merging circular arcs; (**h**) after applying RHT; (**i**) final result; (**j**) ground-truth

combined arc are estimated as the weighted arithmetic means of the radii and centers of the constituent arcs, respectively; the weight is taken as the number of points of the constituent arc. For example, some of the segments in Table 9.4 are combined

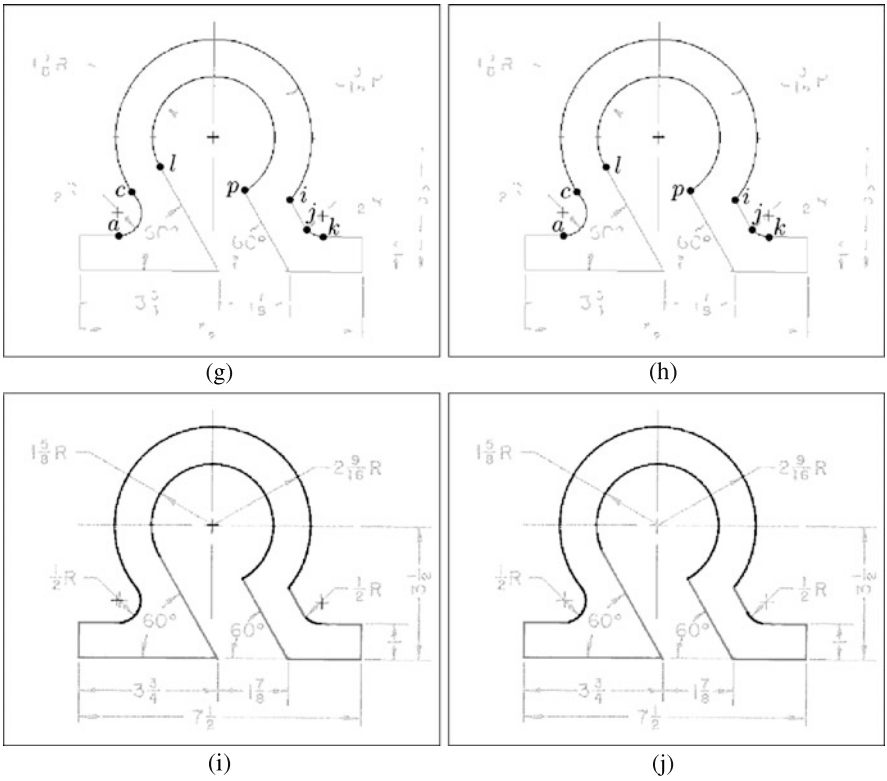


Fig. 9.16 (Continued)

into segments 1 and 2 (Fig. 9.16(g)), with the updated information being listed in Table 9.4. Next we apply RHT on each arc. Resultant image is shown in Fig. 9.16(h) and the arcs are detailed out in Table 9.4. Finally, we consider the detected circular arcs in the original (i.e., input) image and for each pixel on a detected arc S, the object pixels in its 8-neighborhood are iteratively marked as pixels of the corresponding thick circular arc. Figure 9.16(i) shows the detected thick circular arcs of the input image. Figure 9.17 shows the set of results on another image, `2007-1.tif`.

9.6 Future Work and Open Problems

Digital straightness and digital circularity have interesting properties, some of which have been discovered so far, and some are yet to. Digital straight lines have also some relation with Farey sequences [64, 84, 136], and their proper employment in the digital plane can also result to efficient algorithms, as shown recently in [116, 117]. Digital circularity, whether in theoretical or in empirical sense, has also many

Table 9.4 Changes in $\mathscr{L}$ after successive stages on the image of Fig. 9.16

Seg. No.	End point 1	End point 2	# Curve points	Center	Radius	Curve points
(a) After detecting the circular arcs						
1	f(303, 45)	e(193, 107)	118	–	–	(303, 45), (302, 45), (301, 45), . . . , (193, 107)
2	f(305, 45)	g(414, 109)	118	–	–	(305, 45), (306, 45), (307, 45), . . . , (414, 109)
3	o(304, 93)	p(344, 243)	188	–	–	(304, 93), (305, 94), (306, 94), . . . , (344, 243)
4	e(192, 108)	d(174, 172)	65	–	–	(192, 108), (191, 109), (191, 110), . . . , (174, 172)
5	g(415, 110)	h(432, 174)	65	–	–	(415, 110), (416, 111), (417, 112), . . . , (432, 174)
6	n(236, 131)	o(301, 94)	82	–	–	(236, 131), (236, 130), (236, 129) . . . , (304, 94)
7	n(235, 132)	m(223, 172)	42	–	–	(235, 132), (234, 132), (233, 133), . . . , (223, 172)
8	d(174, 174)	c(199, 249)	76	–	–	(174, 174), (174, 175), (174, 176), . . . , (199, 249)
9	m(223, 174)	l(233, 213)	40	–	–	(223, 174), (223, 175), (223, 176), . . . , (233, 213)
10	h(432, 175)	i(404, 255)	81	–	–	(432, 175), (432, 176), (432, 177), . . . , (404, 225)
11	c(199, 250)	b(200, 293)	45	–	–	(199, 250), (200, 251), (201, 252), . . . , (200, 293)
12	b(199, 294)	a(181, 303)	20	–	–	(199, 294), (199, 295), (198, 296), . . . , (181, 303)
13	j(426, 295)	k(445, 304)	20	–	–	(426, 295), (427, 296), (428, 296), . . . , (445, 304)
(b) After combining adjacent arcs and detection of centers by *sagitta property*						
1	o(304, 93)	p(344, 243)	188	(302, 174)	81	(304, 93), (305, 94), (306, 94), . . . , (344, 243)
2	h(432, 174)	d(174, 172)	366	(303, 174)	129	(432, 174), (432, 173), (432, 172), . . . , (174, 172)
3	d(174, 174)	c(199, 249)	76	(294, 176)	120	(174, 174), (174, 175), (174, 176), . . . , (199, 249)
4	l(233, 213)	o(301, 94)	164	(302, 173)	79	(233, 213), (233, 212), (232, 211), . . . , (301, 94)
5	h(432, 175)	i(404, 255)	81	(319, 180)	113	(432, 175), (432, 176), (432, 177), . . . , (404, 255)
6	c(199, 250)	a(181, 303)	65	(177, 272)	31	(199, 250), (200, 251), (201, 252), . . . , (181, 303)
7	j(426, 295)	k(445, 304)	20	(446, 277)	27	(426, 295), (427, 296), (428, 296), . . . , (445, 304)
(c) After merging						
1	l(233, 213)	p(344, 243)	352	(302, 173)	80	(233, 213), (233, 212), (232, 211), . . . , (344, 243)
2	c(199, 249)	i(404, 255)	523	(303, 174)	124	(199, 249), (198, 248), (197, 247), . . . , (404, 255)
3	c(199, 250)	a(181, 303)	65	(177, 272)	31	(199, 250), (200, 251), (201, 252), . . . , (181, 303)
4	j(426, 295)	k(445, 304)	20	(446, 277)	27	(426, 295), (427, 296), (428, 296), . . . , (445, 304)
(d) After RHT						
1	l(233, 213)	p(344, 243)	352	(303, 173)	80	(233, 213), (233, 212), (232, 211), . . . , (344, 243)
2	c(199, 249)	i(404, 255)	523	(303, 174)	129	(199, 249), (198, 248), (197, 247), . . . , (404, 255)
3	c(199, 250)	a(181, 303)	65	(177, 272)	31	(199, 250), (200, 251), (201, 252), . . . , (181, 303)
4	j(426, 295)	k(445, 304)	20	(447, 276)	28	(426, 295), (427, 296), (428, 296), . . . , (445, 304)

relevant applications, as shown in [78, 93, 108, 109, 118, 119, 127–129]. Some open problems that are relevant to the context of this chapter are given below.

1. **DSS Cover** Given a (finite) connected set of digital points, P, find the minimal set of digitally straight segments (DSS) so that each point of P belongs to some DSS from the minimal set. This problem is quite related to vectorization of thick digital curves in general, which is of course in $\mathbb{Z}^2$ [36, 52, 55, 70, 71, 117, 139, 157]. So far, there has been some work in $\mathbb{R}^2$, which is possibly not strongly related with the digital version. As stated in [65, 68], the minimum line cover problem is, given a set P of n points in $\mathbb{R}^2$, the smallest number l of straight lines have to be found to cover all points in S.

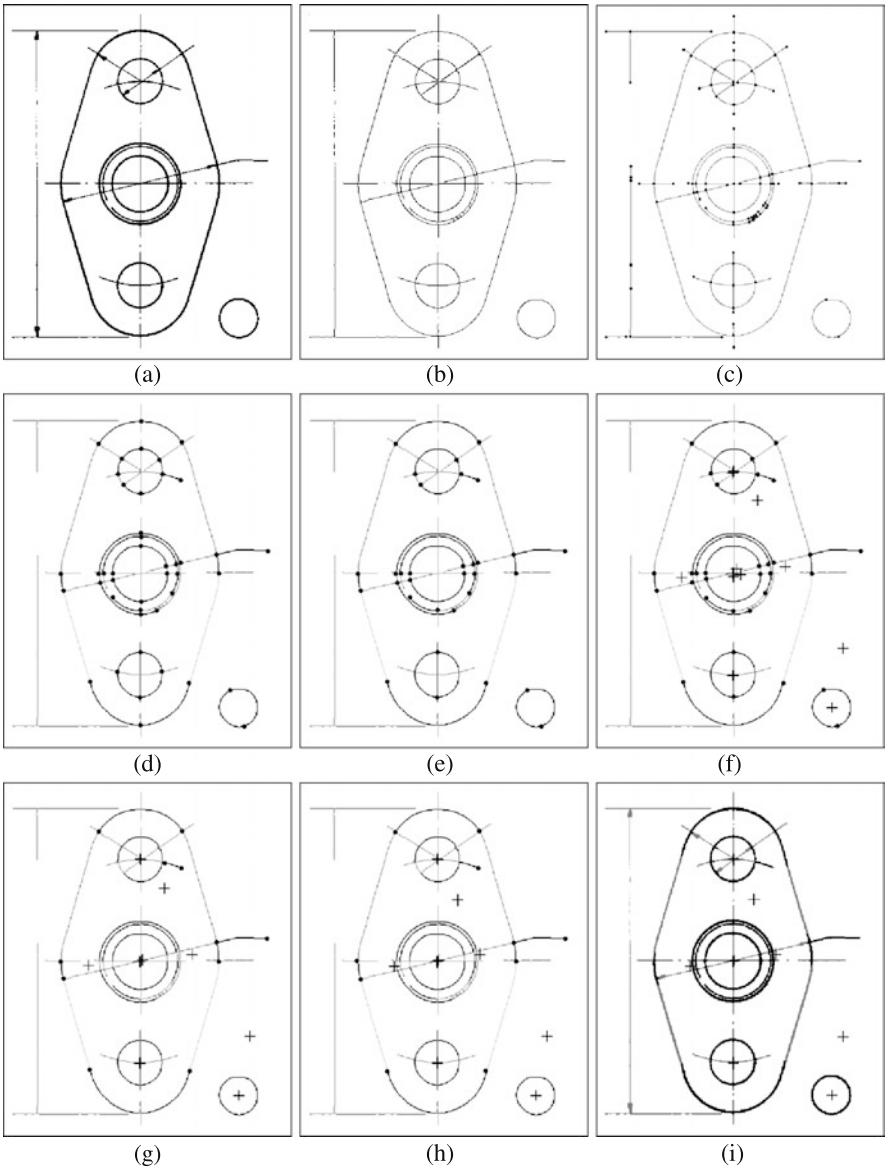


Fig. 9.17 Step-wise snapshots of our experiment on `2007-1.tif`: (**a**) input; (**b**) after thinning; (**c**) intersection points and end points in $\mathscr{L}$; (**d**) detected circular arcs by *chord property*; (**e**) after combining adjacent arcs; (**f**) centers detected by *sagitta property*; (**g**) after merging circular arcs; (**h**) after applying RHT; (**i**) final result

This problem has been known to be NP-hard [101] for over 20 years. The decision version has been shown to be fixed-parameter tractable in [94], and an $O(n \log l)$-time algorithm has been given in [65] for $l = O(\log^{1-\varepsilon} n)$.

2. **DCA Cover** For $P \subset \mathbb{Z}^2$, a similar and more difficult problem is to find the minimal set of digitally circular arcs (DCA) whose union produces S. When P has a circular pattern (e.g., digitized documents containing thick circular curves, often found in engineering drawings), such a DCA cover would be more efficient than a DSS cover, thereby yielding better vectorization.
3. **Combined Cover** In general, (digital) lines and circular arcs may get intermixed, wherefore an optimal solution may be required. This can be resolved by an appropriate employment of exact or approximate solutions in terms of DSS cover and DCA cover.

   A trade-off is also an obvious issue, which might be addressed in this context.
4. **δ-DSS Cover** A restricted (and possibly more application-relevant) version of line cover problem in $\mathbb{Z}^2$ is, given a positive integer δ and a (finite) point set $P \subset \mathbb{Z}^2$, find a minimal set of DSS, D, such that each point of P has an isothetic distance of at most δ from some DSS $\mathbf{d} \in D$. The isothetic distance between two points $p(x, y)$ and $p'(x', y')$ is given by $d_\perp(p, p') = \min\{|x - x'|, |y - y'|\}$. Hence, for each point $p \in P$, there should exist some DSS $\mathbf{d} \in D$ so that $d_\perp(p, \mathbf{d}) \leqslant \delta$, where $d_\perp(p, \mathbf{d}) = \min\{d_\perp(p, p') :\ p' \in \mathbf{d}\}$.

   This is a variant of the traditional skeletonization or thinning problem [63], and is in fact, more well-defined.
5. **δ-DCA Cover** The problem of δ-DSS cover can also be framed for a restricted version of DCA cover, as follows. Find a minimal set of digitally circular arcs, A, such that each point of P has an isothetic distance of at most δ from some arc $\mathbf{a} \in A$.

   In case there are multiple solutions for the above two problems, we can impose some other metric to be optimized in addition. For example, if there exists multiple instances of D, then find the one for which $\sum_{p \in P} d_\perp(p, p' :\ p' \in \mathbf{a}_p \in A)$ is minimized; here $\mathbf{a}_p$ denotes a/the digitally circular arc in A from which the isothetic distance of p is minimum.

9.7 Conclusion

Several theoretical perspectives and related applications of digital straightness and digital circularity have been discussed here. Comparative studies of existing approaches to solving some contemporary problems have been investigated, and digital-geometric techniques have been explained for modeling and analyzing them in the digital plane. Appropriate modelings and reformulations have been done to conceive the problems in the digital plane. Some applications of these ideas to polygonal approximation and circular arc segmentation have been aptly demonstrated with theoretical and experimental results.

It is evident from the discussions and the algorithms presented here, that a set of ADSS extracted from a digital curve segment is significantly smaller in size than that of DSS extracted from the same, although each ADSS can be treated as sufficiently straight for various practical applications. Furthermore, the CPU time needed for

ADSS extraction is remarkably less than that for DSS extraction. The extracted set of ADSS can be combined to determine a compact polygonal approximation of a digital curve based on certain approximation criteria and a specified error tolerance. The algorithm and related procedures require only primitive integer operations and hence run very fast compared to the existing methods.

To determine digital circularity, we have shown how integer arithmetic can be used to compute successive radius intervals during *radii nesting*. The algorithm DCT is very efficient when the circularity testing has to be done on a digital curve segment S with the correspondence of its first run with one of the extremum runs of the underlying digital circle/circular arc. As the algorithm DCT is based on run-length analysis and not on the constituent digital points/pixels of S, the time complexity of DCT is effectively linear in the number of runs, which is one of its major features. The concept of *conflicting radii* has also been discussed to show how the rate of convergence of the effective radius interval depends on the run-length pattern of a digital curve segment. Demonstrations and elaborations detailed out in this chapter clearly reveal the challenge of getting a digital-geometric solution to digital circularity.

For real-world applications (e.g., vectorization) in which the circular arcs usually deviate from the actual/well-defined digital circular arcs, an approximation algorithm might be more useful. One such is explained from [12], which identifies digital circles and circular arcs from a binary image using *chord property* and *sagitta property*, and is not strictly based on the notion of approximate circularity or related number-theoretic analysis. The decision problem on *approximate digital circularity* is that, given an approximation parameter τ and a digital curve segment S, whether there exists a digital circle $\mathscr{C}^{\mathbb{Z}}(r)$ for which each point p of S is at most τ units apart from the corresponding nearest point of $\mathscr{C}^{\mathbb{Z}}(r)$. A judicious relaxation of the integer interval of radii is required to address this problem. The digital-geometric solution based solely on integer intervals is technically engrossing, which, if achieved, would open up novel possibilities to solve the approximate solution of circular arc segmentation in related applications.

References

1. Aken, J.R.V., Novak, M.: Curve-drawing algorithms for raster display. ACM Trans. Graph. **4**(2), 147–169 (1985)
2. Anderson, I.M., Bezdek, J.C.: Curvature and tangential deflection of discrete arcs: a theory based on the commutator of scatter matrix pairs and its application to vertex detection in planar shape data. IEEE Trans. Pattern Anal. Mach. Intell. **6**, 27–40 (1984)
3. Andres, E.: Discrete circles, rings and spheres. Comput. Graph. **18**(5), 695–706 (1994)
4. Andres, E., Jacob, M.: The discrete analytical hyperspheres. IEEE Trans. Vis. Comput. Graph. **3**(1), 75–86 (1997)
5. Asano, T., Brass, P., Sasahara, S.: Disc covering problem with application to digital halftoning. Theory Comput. Syst. **46**(2), 157–173 (2010)
6. Asano, T., Brimkov, V.E., Barneva, R.P.: Some theoretical challenges in digital geometry: a perspective. Discrete Appl. Math. **157**(16), 3362–3371 (2009)

7. Asano, T., Katoh, N.: Number theory helps line detection in digital images. In: ISAAC, pp. 313–322 (1993)
8. Asano, T., Klette, R., Ronse, C. (eds.): Geometry, Morphology, and Computational Imaging. LNCS, vol. 2616. Springer, Berlin (2003)
9. Attneave, F.: Some informational aspects of visual perception. Psychol. Rev. **61**(3), 183–193 (1954)
10. Balog, A., Bárány, I.: On the convex hull of the integer points in a disc. In: Proc. 7th Annual Symposium on Computational Geometry, SCG 1991, pp. 162–165 (1991)
11. Barneva, R.P., Brimkov, V.E., Hauptman, H.A., Jorge, R.M.N., Tavares, J.M.R.S. (eds.): Computational Modeling of Objects Represented in Images, 2nd Int. Symposium Comp-IMAGE 2010. LNCS, vol. 6026. Springer, Berlin (2010)
12. Bera, S., Bhowmick, P., Bhattacharya, B.B.: Detection of circular arcs in a digital image using chord and sagitta properties. In: Extended Version of Proc. Eighth Int. Workshop on Graphics Recognition, GREC 2009. LNCS, vol. 6020, pp. 69–80 (2010)
13. Bezdek, J.C., Anderson, I.M.: An application of the *c*-varieties clustering algorithms to polygonal curve fitting. IEEE Trans. Syst. Man Cybern. **15**, 637–641 (1985)
14. Bhowmick, P., Bhattacharya, B.B.: Fast polygonal approximation of digital curves using relaxed straightness properties. IEEE Trans. Pattern Anal. Mach. Intell. **29**(9), 1590–1602 (2007)
15. Bhowmick, P., Bhattacharya, B.B.: Number-theoretic interpretation and construction of a digital circle. Discrete Appl. Math. **156**(12), 2381–2399 (2008). doi:10.1016/j.dam.2007.10.022
16. Bhowmick, P., Biswas, A., Bhattacharya, B.B.: Isothetic polygons of a 2D object on generalized grid. In: Proc. 1st Int. Conf. Pattern Recognition and Machine Intelligence (PReMI). LNCS, vol. 3776, pp. 407–412. Springer, Berlin (2005)
17. Biswas, A., Bhowmick, P., Bhattacharya, B.B.: TIPS: On finding a Tight Isothetic Polygonal Shape covering a 2D object. In: Proc. 14th Scandinavian Conf. Image Analysis (SCIA). LNCS, vol. 3540, pp. 930–939. Springer, Berlin (2005)
18. Blinn, J.F.: How many ways can you draw a circle? IEEE Comput. Graph. Appl. **7**(8), 39–44 (1987)
19. Boyer, C.B.: A History of Mathematics, 2nd edn. Wiley, New York (1991)
20. Bresenham, J.E.: An incremental algorithm for digital plotting. In: Proc. ACM Natl. Conf. (1963)
21. Bresenham, J.E.: A linear algorithm for incremental digital display of circular arcs. Commun. ACM **20**(2), 100–106 (1977)
22. Bresenham, J.E.: Run length slice algorithm for incremental lines. In: Earnshaw, R.A. (ed.) Fundamental Algorithms for Computer Graphics. NATO ASI Series, vol. F17, pp. 59–104. Springer, New York (1985)
23. Brimkov, V., Coeurjolly, D., Klette, R.: Digital planarity—a review. Discrete Appl. Math. **155**(4), 468–495 (2007). http://dx.doi.org/10.1016/j.dam.2006.08.004
24. Brimkov, V.E., Barneva, R.P.: Graceful planes and lines. Theor. Comput. Sci. **283**(1), 151–170 (2002). http://dx.doi.org/10.1016/S0304-3975(01)00061-5
25. Brimkov, V.E., Barneva, R.P.: Connectivity of discrete planes. Theor. Comput. Sci. **319**(1–3), 203–227 (2004)
26. Brimkov, V.E., Barneva, R.P.: Connectivity of discrete planes. Theor. Comput. Sci. **319**(1–3), 203–227 (2004). http://dx.doi.org/10.1016/j.tcs.2004.02.015
27. Brimkov, V.E., Barneva, R.P.: On the polyhedral complexity of the integer points in a hyperball. Theor. Comput. Sci. **406**(1–2), 24–30 (2008)
28. Brimkov, V.E., Barneva, R.P.: Advances in combinatorial image analysis. Pattern Recognit. **42**(8), 1623–1625 (2009)
29. Brimkov, V.E., Barneva, R.P.: Combinatorial approach to image analysis. Discrete Appl. Math. **157**(16), 3359–3361 (2009)
30. Brimkov, V.E., Moroni, D., Barneva, R.P.: Combinatorial relations for digital pictures. In: DGCI, pp. 189–198 (2006)

31. Brons, R.: Linguistic methods for description of a straight line on a grid. Comput. Graph. Image Process. **2**, 48–62 (1974)
32. Chattopadhyay, S., Das, P.P., Ghosh-Dastidar, D.: Reconstruction of a digital circle. Pattern Recognit. **27**(12), 1663–1676 (1994)
33. Chaudhuri, B.B., Rosenfeld, A.: On the computation of the digital convex hull and circular hull of a digital region. Pattern Recognit. **31**(12), 2007–2016 (1998)
34. Chen, T.C., Chung, K.L.: An efficient randomized algorithm for detecting circles. Comput. Vis. Image Underst. **83**(2), 172–191 (2001)
35. Chen, T.C., Chung, K.L.: A new randomized algorithm for detecting lines. Real-Time Imaging **7**, 473–481 (2001)
36. Chiang, Y., Knoblock, C.: An approach to automatic road vectorization of raster maps. In: Proc. GREC 2009 (2009)
37. Chiu, S.H., Liaw, J.J.: An effective voting method for circle detection. Pattern Recognit. Lett. **26**(2), 121–133 (2005)
38. Chung, W.L.: On circle generation algorithms. Comput. Graph. Image Process. **6**, 196–198 (1977)
39. Climer, S., Bhatia, S.K.: Local lines: a linear time line detector. Pattern Recognit. Lett. **24**, 2291–2300 (2003)
40. Coeurjolly, D., Gérard, Y., Reveillès, J.P., Tougne, L.: An elementary algorithm for digital arc segmentation. Discrete Appl. Math. **139**, 31–50 (2004)
41. Coeurjolly, D., Miguet, S., Tougne, L.: Discrete curvature based on osculating circle estimation. In: IWVF-4: Proc. 4th Int. Workshop Visual Form, pp. 303–312. Springer, London (2001)
42. Coeurjolly, D., Sivignon, I., Dupont, F., Feschet, F., Chassery, J.M.: On digital plane preimage structure. Discrete Appl. Math. **151**(1–3), 78–92 (2005)
43. Creutzburg, E., Hübler, A., Wedler, V.: On-line recognition of digital straight line segments. In: Proc. 2nd Int. Conf. AI and Inf. Control Systems of Robots, pp. 42–46 (1982)
44. Damaschke, P.: The linear time recognition of digital arcs. Pattern Recognit. Lett. **16**, 543–548 (1995)
45. Danielsson, P.E.: Comments on circle generator for display devices. Comput. Graph. Image Process. **7**(2), 300–301 (1978)
46. Davies, E.: Machine Vision: Theory, Algorithms, Practicalities. Academic Press, London (1990)
47. Davies, E.R.: A modified Hough scheme for general circle location. Pattern Recognit. **7**(1), 37–43 (1984)
48. Davies, E.R.: A high speed algorithm for circular object detection. Pattern Recognit. Lett. **6**, 323–333 (1987)
49. Davis, L.S., Rosenfeld, A., Agrawala, A.K.: On models for line detection. IEEE Trans. Syst. Man Cybern. **6**, 127–133 (1976)
50. Debled-Rennesson, I., Reveilles, J.P.: A linear algorithm for segmentation of digital curves. Int. J. Pattern Recognit. Artif. Intell. **9**, 635–662 (1995)
51. de Vieilleville, F., Lachaud, J.O., Feschet, F.: Convex digital polygons, maximal digital straight segments and convergence of discrete geometric estimators. J. Math. Imaging Vis. **27**(2), 139–156 (2007)
52. Dori, D., Liu, W.: Sparse pixel vectorization: an algorithm and its performance evaluation. IEEE Trans. Pattern Anal. Mach. Intell. **21**(3), 202–215 (1999)
53. Doros, M.: Algorithms for generation of discrete circles, rings, and disks. Comput. Graph. Image Process. **10**, 366–371 (1979)
54. Dunham, J.G.: Optimum uniform piecewise linear approximation of planar curves. IEEE Trans. Pattern Anal. Mach. Intell. **8**, 67–75 (1986)
55. Fan, K., Chen, D., Wen, M.: A new vectorization-based approach to the skeletonization of binary images. In: 3rd Int. Conf. Document Analysis and Recognition, ICDAR 1995 (1995)
56. Fischler, M.A., Wolf, H.C.: Locating perceptually salient points on planar curves. IEEE Trans. Pattern Anal. Mach. Intell. **16**(2), 113–129 (1994)

57. Fisk, S.: Separating point sets by circles, and the recognition of digital disks. IEEE Trans. Pattern Anal. Mach. Intell. **8**, 554–556 (1986)
58. Foley, J.D., Dam, A.V., Feiner, S.K., Hughes, J.F.: Computer Graphics—Principles and Practice. Addison-Wesley, Reading (1993)
59. Foresti, G.L., Regazzoni, C.S., Vernazza, G.: Circular arc extraction by direct clustering in a 3D Hough parameter space. Signal Process. **41**, 203–224 (1995)
60. Freeman, H.: On the encoding of arbitrary geometric configurations. IRE Trans. Electron. Comput. **EC-10**, 260–268 (1961)
61. Freeman, H.: Techniques for the digital computer analysis of chain-encoded arbitrary plane curves. In: Proc. National Electronics Conf., vol. 17, pp. 421–432 (1961)
62. Freeman, H., Davis, L.S.: A corner finding algorithm for chain-coded curves. IEEE Trans. Comput. **26**, 297–303 (1977)
63. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Addison-Wesley, Reading (1993)
64. Graham, R., Knuth, D., Patashnik, O.: Concrete Mathematics. Addison-Wesley, London (1994)
65. Grantson, M., Levcopoulos, C.: Covering a set of points with a minimum number of lines. In: Calamoneri, T., Finocchi, I., Italiano, G. (eds.) Algorithms and Complexity. LNCS, vol. 3998, pp. 6–17. Springer, Berlin (2006)
66. Guru, D.S., Shekar, B.H., Nagabhushan, P.: A simple and robust line detection algorithm based on small eigenvalue analysis. Pattern Recognit. Lett. **25**, 1–13 (2004)
67. Haralick, R.M.: A measure for circularity of digital figures. IEEE Trans. Syst. Man Cybern. **4**, 394–396 (1974)
68. Heednacram, A.: The NP-hardness of covering points with lines, paths and tours and their tractability with FPT-algorithms. Ph.D. thesis, Institute for Integrated and Intelligent Systems, Science, Environment, Engineering and Technology, Griffith University (2001)
69. Held, A., Abe, K., Arcelli, C.: Towards a hierarchical contour description via dominant point detection. IEEE Trans. Syst. Man Cybern. **24**, 942–949 (1994)
70. Hilaire, X., Tombre, K.: Robust and accurate vectorization of line drawings. IEEE Trans. Pattern Anal. Mach. Intell. **28**(6), 890–904 (2006)
71. Hilaire, X., Tombre, K.: Robust and accurate vectorization of line drawings. IEEE Trans. Pattern Anal. Mach. Intell. **28**(6), 890–904 (2006)
72. Horn, B.K.P.: Circle generators for display devices. Comput. Graph. Image Process. **5**(2), 280–288 (1976)
73. Hsu, S.Y., Chow, L.R., Liu, C.H.: A new approach for the generation of circles. Comput. Graph. Forum **12**(2), 105–109 (1993)
74. Imai, H., Iri, M.: Computational geometric methods for polygonal approximations of a curve. Comput. Vis. Graph. Image Process. **36**, 31–41 (1986)
75. Imiya, A., Ootani, H., Tatara, K.: Medial set, boundary, and topology of random point sets. In: Theoretical Foundations of Computer Vision, pp. 196–217 (2002)
76. Ioannoua, D., Hudab, W., Lainec, A.: Circle recognition through a 2D Hough transform and radius histogramming. Image Vis. Comput. **17**, 15–26 (1999)
77. Kenmochi, Y., Imiya, A.: Combinatorial topologies for discrete planes. In: DGCI, pp. 144–153 (2003)
78. Kerautret, B., Lachaud, J.O., Nguyen, T.P.: Circular arc reconstruction of digital contours with chosen Hausdorff error. In: DGCI, pp. 247–259 (2011)
79. Kim, C.: Digital disks. IEEE Trans. Pattern Anal. Mach. Intell. **6**, 372–374 (1984)
80. Kim, C.E., Anderson, T.A.: Digital disks and a digital compactness measure. In: Proc. 16th Annual ACM Symposium on Theory of Computing, pp. 117–124 (1984)
81. Kim, H.S., Kim, J.H.: A two-step circle detection algorithm from the intersecting chords. Pattern Recognit. Lett. **22**(6–7), 787–798 (2001)
82. Klette, R.: Digital geometry—the birth of a new discipline. In: Davis, L.S. (ed.) Foundations of Image Understanding, pp. 33–71. Kluwer Academic, Boston (2001)
83. Klette, R.: Topologies on the planar orthogonal grid. In: ICPR (2), pp. 354–357 (2002)

84. Klette, R., Rosenfeld, A.: Digital Geometry: Geometric Methods for Digital Picture Analysis. Morgan Kaufmann, San Francisco (2004)
85. Klette, R., Rosenfeld, A.: Digital straightness—a review. Discrete Appl. Math. **139**(1–3), 197–230 (2004)
86. Klette, R., Žunić, J.: Interactions between number theory and image analysis. In: Latecki, L.J., Mount, D.M., Wu, A.Y. (eds.) Vision Geometry IX. Proc. SPIE, vol. 4117, pp. 210–221 (2000)
87. Kong, T.Y.: Digital topology. In: Davis, L.S. (ed.) Foundations of Image Understanding, pp. 33–71. Kluwer Academic, Boston (2001)
88. Koplowitz, J., Lindenbaum, M., Bruckstein, A.: The number of digital straight lines on an $n \times n$ grid. IEEE Trans. Inf. Theory **36**, 192–197 (1990)
89. Kovalevsky, V.A.: New definition and fast recognition of digital straight segments and arcs. In: Proc. 10th Int. Conf. Pattern Recognition (ICPR), pp. 31–34. IEEE Comput. Soc., Los Alamitos (1990)
90. Kulpa, Z.: A note on "Circle generator for display devices". Comput. Graph. Image Process. **9**, 102–103 (1979)
91. Kulpa, Z., Kruse, B.: Algorithms for circular propagation in discrete images. Comput. Vis. Graph. Image Process. **24**(3), 305–328 (1983)
92. Lachaud, J.O., Montanvert, A.: Continuous analogs of digital boundaries: a topological approach to iso-surfaces. Graph. Models **62**(3), 129–164 (2000)
93. Lamiroy, B., Guebbas, Y.: Robust and precise circular arc detection. In: Extended Version of Proc. Eighth Int. Workshop on Graphics Recognition, GREC 2009. LNCS, vol. 6020, pp. 49–60 (2010)
94. Langerman, S., Morin, P.: Cover things with things. Discrete Comput. Geom. **33**(4), 717–729 (2005)
95. Latecki, L.J., Lakämper, R.: Shape similarity measure based on correspondence of visual parts. IEEE Trans. Pattern Anal. Mach. Intell. **22**(10), 1185–1190 (2000)
96. Leavers, V.: Survey: which Hough transform? CVGIP, Image Underst. **58**, 250–264 (1993)
97. Lothaire, M.: Algebraic Combinatorics on Words. Cambridge Mathematical Library (2002)
98. McIlroy, M.D.: A note on discrete representation of lines. AT&T Bell Lab. Tech. J. **64**(2), 481–490 (1985)
99. Mcllroy, M.D.: Best approximate circles on integer grids. ACM Trans. Graph. **2**(4), 237–263 (1983)
100. Megiddo, N.: Linear time algorithm for linear programming in $\mathbb{R}^3$ and related problems. SIAM J. Comput. **12**, 759–776 (1983)
101. Megiddo, N., Tamir, A.: On the complexity of locating linear facilities in the plane. Oper. Res. Lett. **1**(5), 194–197 (1982)
102. Mieghem, J.A.V., Avi-Itzhak, H.I., Melen, R.D.: Straight line extraction using iterative total least squares methods. J. Vis. Commun. Image Represent. **6**, 59–68 (1995)
103. Mignosi, F.: On the number of factors of Sturmian words. Theor. Comput. Sci. **82**(1), 71–84 (1991)
104. Nakamura, A., Aizawa, K.: Digital circles. Comput. Vis. Graph. Image Process. **26**(2), 242–255 (1984)
105. Nakamura, A., Rosenfeld, A.: Digital calculus. Inf. Sci. **98**, 83–98 (1997)
106. Nguyen, T.P., Debled-Rennesson, I.: A linear method for segmentation of digital arcs. Rapport de recherche no. 0001, Centre de recherche INRIA, Nancy (2010)
107. Pal, S., Bhowmick, P.: Determining digital circularity using integer intervals. J. Math. Imaging Vis. (2011). doi:10.1007/s10851-011-0270-6
108. Pal, S., Bhowmick, P.: Fast circular arc segmentation based on approximate circularity and cuboid graph (communicated 2011)
109. Pal, S., Bhowmick, P., Biswas, A.: FACET: a fast approximate circularity estimation technique. In: Proc. Second Int. Conf. Emerging Applications of Information Technology, EAIT-2011, pp. 106–109. IEEE Comput. Soc., Los Alamitos (2011)
110. Pavlidis, T.: Structural Pattern Recognition. Springer, New York (1977)

111. Pavlidis, T.: Algorithms for shape analysis and waveforms. IEEE Trans. Pattern Anal. Mach. Intell. **2**, 301–312 (1980)
112. Perez, J.C., Vidal, E.: Optimum polygonal approximation of digitized curves. Pattern Recognit. Lett. **15**, 743–750 (1994)
113. Pitteway, M.L.V.: Algorithm for drawing ellipses or hyperbolae with a digital plotter. Comput. J. **10**(3), 282–289 (1967)
114. Pitteway, M.L.V.: Integer circles, etc.—some further thoughts. Comput. Graph. Image Process. **3**, 262–265 (1974)
115. Povazan, I., Uher, L.: The structure of digital straight line segments and Euclid's algorithm. In: Proc. Spring Conf. Computer Graphics, pp. 205–209 (1998)
116. Pratihar, S., Bhowmick, P.: Shape decomposition using Farey sequence and saddle points. In: Proceedings of the Seventh Indian Conference on Computer Vision, Graphics and Image Processing, pp. 77–84. ACM, New York (2010)
117. Pratihar, S., Bhowmick, P.: Vectorization of thick digital lines using Farey sequence and geometric refinement. In: Proceedings of the Seventh Indian Conference on Computer Vision, Graphics and Image Processing, pp. 518–525. ACM, New York (2010)
118. Richard, A., Wallet, G., Fuchs, L., Andres, E., Largeteau-Skapin, G.: Arithmetization of a circular arc. In: Proc. 15th IAPR International Conference on Discrete Geometry for Computer Imagery, DGCI '09. LNCS, vol. 5810, pp. 350–361. Springer, Berlin (2009)
119. Ritter, N., Cooper, J.: New resolution independent measures of circularity. J. Math. Imaging Vis. **35**(2), 117–127 (2009)
120. Rocha, J., Bernardino, R.: Singularities and regularities on line pictures via symmetrical trapezoids. IEEE Trans. Pattern Anal. Mach. Intell. **20**(4), 391–395 (1998)
121. Rodríguez, M., Abdoulaye, S., Largeteau-Skapin, G., Andres, E.: Generalized perpendicular bisector and circumcenter. In: Computational Modeling of Objects Represented in Images, 2nd Int. Symposium CompIMAGE 2010. LNCS, vol. 6026, pp. 1–10. Springer, Berlin (2010)
122. Rosenfeld, A.: Digital straight line segments. IEEE Trans. Comput. **23**(12), 1264–1268 (1974)
123. Rosenfeld, A., Kak, A.C.: Digital Picture Processing, 2nd edn. Academic Press, New York (1982)
124. Rosenfeld, A., Klette, R.: Digital straightness. Electronic Notes in Theor. Comput. Sci. **46**, 1–32 (2001). http://www.elsevier.nl/locate/entcs/volume46.html
125. Rosin, P.L.: Techniques for assessing polygonal approximation of curves. IEEE Trans. Pattern Anal. Mach. Intell. **19**(6), 659–666 (1997)
126. Rosin, P.L., West, G.A.W.: Non-parametric segmentation of curves into various representations. IEEE Trans. Pattern Anal. Mach. Intell. **17**, 1140–1153 (1995)
127. Roussillon, T., Sivignon, I., Tougne, L.: On-line recognition of digital circular arcs. In: Proc. 15th IAPR International Conference on Discrete Geometry for Computer Imagery, DGCI '09. LNCS, vol. 5810, pp. 34–45. Springer, Berlin (2009)
128. Roussillon, T., Sivignon, I., Tougne, L.: Measure of circularity for parts of digital boundaries and its fast computation. Pattern Recognit. **43**(1), 37–46 (2010)
129. Roussillon, T., Tougne, L., Sivignon, I.: On three constrained versions of the digital circular arc recognition problem. In: Proc. 15th IAPR International Conference on Discrete Geometry for Computer Imagery, DGCI '09. LNCS, vol. 5810, pp. 34–45. Springer, Berlin (2009)
130. Saha, P.K., Chaudhuri, B.B.: 3d digital topology under binary transformation with applications. Comput. Vis. Image Underst. **63**(3), 418–429 (1996)
131. Saha, P.K., Chaudhuri, B.B., Chanda, B., Majumder, D.D.: Topology preservation in 3d digital space. Pattern Recognit. **27**(2), 295–300 (1994)
132. Said, M., Lachaud, J.O., Feschet, F.: Multiscale discrete geometry. In: DGCI, pp. 118–131 (2009)
133. Sarkar, D.: A simple algorithm for detection of significant vertices for polygonal approximation of chain-coded curves. Pattern Recognit. Lett. **14**, 959–964 (1993)

134. Sauer, P.: On the recognition of digital circles in linear time. Comput. Geom. **2**, 287–302 (1993)
135. Schröder, K., Laurent, P.: Efficient polygon approximations for shape signatures. In: Proc. Int. Conf. Image Processing (ICIP), pp. 811–814. IEEE Comput. Soc., Los Alamitos (1999)
136. Schroeder, M.: Fractions: continued, Egyptian and Farey (Chap. 5). In: Number Theory in Science and Communication. Springer Series in Information Sciences, vol. 7 (2006)
137. Schuster, G.M., Katsaggelos, A.K.: An optimal polygonal boundary encoding scheme in the rate distortion sense. IEEE Trans. Circuits Syst. Video Technol. **7**, 13–26 (1998)
138. Smeulders, A.W.M., Dorst, L.: Decomposition of discrete curves into piecewise segments in linear time. Contemp. Math. **119**, 169–195 (1991)
139. Song, J.: An object oriented progressive-simplification based vectorization system for engineering drawings: model, algorithm, and performance. IEEE Trans. Pattern Anal. Mach. Intell. **24**(8), 890–904 (2002)
140. Sonka, M., Hlavac, V., Boyle, R.: Image Processing, Analysis, and Machine Vision. Chapman & Hall, London (1993)
141. Suenaga, Y., Kamae, T., Kobayashi, T.: A high speed algorithm for the generation of straight lines and circular arcs. IEEE Trans. Comput. **28**, 728–736 (1979)
142. Teh, C.H., Chin, R.T.: On the detection of dominant points on digital curves. IEEE Trans. Pattern Anal. Mach. Intell. **2**(8), 859–872 (1989)
143. Ventura, J.A., Chen, J.M.: Segmentation of two-dimensional curve contours. Pattern Recognit. **25**, 1129–1140 (1992)
144. Voss, K.: Coding of digital straight lines by continued fractions. Comput. Artif. Intell. **10**, 75–80 (1991)
145. Wall, K., Danielsson, P.E.: A fast sequential method for polygonal approximation of digitized curves. Comput. Vis. Graph. Image Process. **28**, 220–227 (1984)
146. Weisstein, E.W.: Sagitta. From MathWorld—a Wolfram web resource. http://mathworld.wolfram.com/Sagitta.html (1993)
147. Worring, M., Smeulders, A.W.M.: Digitized circular arcs: characterization and parameter estimation. IEEE Trans. Pattern Anal. Mach. Intell. **17**(6), 587–598 (1995)
148. Wright, W.E.: Parallelization of Bresenham's line and circle algorithms. IEEE Comput. Graph. Appl. **10**(5), 60–67 (1990)
149. Wu, L.D.: A piecewise linear approximation based on a statistical model. IEEE Trans. Pattern Anal. Mach. Intell. **6**, 41–45 (1984)
150. Wu, X., Rokne, J.G.: Double-step incremental generation of lines and circles. Comput. Vis. Graph. Image Process. **37**(3), 331–344 (1987)
151. Wuescher, D.M., Boyer, K.L.: Robust contour decomposition using a constant curvature criterion. IEEE Trans. Pattern Anal. Mach. Intell. **13**(1), 41–51 (1991)
152. Xie, Y., Ji, Q.: Effective line detection with error propagation. In: Proc. Int. Conf. Image Processing (ICIP), pp. 181–184. IEEE Comput. Soc., Los Alamitos (2001)
153. Yao, C., Rokne, J.G.: Hybrid scan-conversion of circles. IEEE Trans. Vis. Comput. Graph. **1**(4), 311–318 (1995)
154. Yin, P.Y.: A new method for polygonal approximation using genetic algorithms. Pattern Recognit. Lett. **19**(11), 1017–1026 (1998)
155. Yin, P.Y.: Ant colony search algorithms for optimal polygonal approximation of plane curves. Pattern Recognit. **36**, 1783–1797 (2003)
156. Yin, P.Y.: A discrete particle swarm algorithm for optimal polygonal approximation of digital curves. J. Vis. Commun. Image Represent. **15**(2), 241–260 (2004)
157. Zou, J., Yan, H.: Line image vectorization based on shape partitioning and merging. In: Proc. ICPR 2000, vol. 3, pp. 994–997 (2000)

Chapter 10
Shape Analysis with Geometric Primitives

Fabien Feschet

Abstract In this chapter, a unifying framework is presented for analyzing shapes using geometric primitives. It requires both a model of shapes and a model of geometric primitives. We deliberately choose to explore the most general form of shapes through the notion of connected components in a binary image. According to this choice, we introduce geometric primitives for straightness and circularity which are adapted to thick elements. Starting from a model (the tangential cover) which emerged in the digital geometry community we show how to use Constrained Delaunay Triangulation to represent all shapes as a connected path well adapted to the recognition of geometric primitives. We moreover describe how to map our framework into the class of circular arc graphs. Using this mapping we present a multi-primitives analysis which is suitable for self-organizing a shape with respect to prescribed geometric primitives. Further work and open problems conclude the chapter.

10.1 Introduction

The raw data contained in an image is usually insufficient for a complete understanding of its content. It is a common problem to extract and represent shapes within an image. For instance it may be used to look for a similar object in a database or to recognize important objects which might help to understand with a computer the content of the image. There are usually two ways to represent shapes in an image either by its boundary or by its interior [17]. The most simple model for representing the boundary is the polygonal model which is somewhat efficient and easy to compute. However, in full generality, more complex geometric primitives than segments must be used to have an efficient and powerful representation. For instance, spline curves are often used for capturing shapes outlines (see references in [17]). Several outline capturing techniques are presented in [17] but most often they rely

F. Feschet (✉)
IGCNC - EA 2782, Clermont Université, Université d'Auvergne, 49 Bd. F. Mitterrand, 63000 Clermont-Ferrand, France
e-mail: research@feschet.fr

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*, Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_10,

on a fitting procedure to approximate the shapes. Moreover, in vectorization approaches [12], fitting with line segments and circles is commonly used. However, as mentioned in [12], the extraction of the thickness of the geometric primitives is usually hard and the presence of noise is a central source of errors.

We propose another approach based on a completely different paradigm. We start by noting that in binary images, the notion of connected components is easy to manage. This is the same for the notion of contour of an object. In the presence of noise, the notion of contour can be replaced, to deal with noise, by two contours [13] which enclose the expected true contour of the object. This leads to the notion of digital outlines. However, digital outlines is just a particular case of viewing a shape as a connected component. Thus, our presentation mainly focuses on connected components but with a model which is also well adapted to digital outlines.

Instead of trying to fit a continuous primitive to our digital data, we use the notion of geometric digital primitives. This approach is well known [15] and usually uses digital straight segments (DSS). The main purpose of the present chapter is to provide means to manipulate thick shapes. So we both recall an extension of the notion of DSS, the α-blurred segments where α is a parameter of thickness, and provide an extension of the notion of thick digital arcs with our notion of α-thick annulus. Having defined basic digital primitives, the first part of the analysis consists in extracting those primitives. For α-blurred segments, this has been done in [6] where an efficient algorithm was presented. By explicitly encoding the interior of shapes with the use of triangulations, we can force geometric primitives to be organized through an extension of the notion of *tangential cover* [9]. The idea of this powerful representation is that when two geometric primitives intersect, it is better to keep both primitives instead of cutting them to have disjoint primitives. This has been for instance used in [10] to solve the min-DSS problem which consists in constructing the polygonal representation of a digital closed curve with a minimal number of DSS in linear time. In the present paper, we provide an extension of the tangential cover, called the *predicate cover (PC)*, in order to manipulate other primitives than just DSS. We show the connection between the predicate cover and the class of circular arcs graphs. Using this, we are able to provide a generic framework well adapted to thin digital shapes, thick shapes, digital outlines and more generally arbitrary connected components.

10.2 The Tangential Cover

As noted previously in the introduction, we will consider different types of shapes and different geometric primitives. The goal of this section is to emphasize the original construction called *the tangential cover* which has been introduced in the context of digital curves [9]. In the next section, we will introduce various types of shapes we are going to analyze in the framework presented in this chapter. We will show that, in fact, all these representations can be manipulated in the same way under a very simple and natural hypothesis. This hypothesis will then serve as a basic building block in our framework and is deduced from the tangential cover construction.

10.2.1 Shapes as Digital Curves

As mentioned in [26], one of the most widely used representations of shapes is based on the boundary of an object. In this model, a shape is the boundary of a compact object. Hence, since there is no holes inside the object, one and only one boundary is necessary and sufficient to entirely represent the object. This boundary is usually a digital curve. We recall basic definitions in the following.

We consider points in the digital plane $\mathbb{Z}^2$ and some adjacency α on those points. The usual adjacencies are the classical 4- and 8-adjacencies [15]. Let us recall that if we denote by $\mathscr{N}()$ a norm in $\mathbb{R}^2$, they are defined by the fact that two points p and q are adjacent if and only if $\mathscr{N}(q-p) \leq 1$. The norm $\mathscr{N}()$ is the l_1 norm—corresponding to the sum of the absolute differences of the coordinates of the points—for the 4-adjacency and it is the infinite norm l_∞—corresponding to the max of the absolute differences of the coordinates of the points—for the 8-adjacency. In the sequel, the adjacency is fixed. Of course, our work readily extends to the case of other cellular decompositions of the real plane $\mathbb{R}^2$ such as the triangular or the hexagonal decomposition.

A digital α-path in the plane (a digital path, for short) is a sequence of points $(p_0, \ldots, p_n)$ of $\mathbb{Z}^2$ such that p_i and p_{i+1} are α-adjacent and $p_i \neq p_{i+1}$. It must be noted that a path is indeed an ordered set of points of $\mathbb{Z}^2$ such that two consecutive points are α-adjacent. Hence, for a given set of points, there may be several paths describing it depending on the chosen ordering. A path is closed if and only if p_0 and p_n are α-adjacent. When a path is closed, indices of the points in the path are intended modulo $n+1$, the length of the path. For a path P, a subpath is a subset of points of P which is an α-path. For an α-path P, either closed or not, we denote by $\mathscr{SP}(P)$ the set of all subpaths of P.

A closed digital curve is a digital path where each point p_i has exactly two α-adjacent points in the path. An open digital curve is a digital path with the same property except for the points p_0 and p_n which have only one α-adjacent point in the path.

Following Wagenknecht [25], many contour tracing algorithms have been proposed in the literature. Among them, the inter-pixel boundary is of interest due to the property that it avoids many of the usual topological issues encountered when dealing, for instance, with one pixel wide objects.

10.2.2 Digital Straight Segments

Definition 1 The set of points (x, y) of $\mathbb{Z}^2$ verifying

$$\mu \leq ax - by < \mu + \omega \tag{10.1}$$

with μ, a and b in $\mathbb{Z}$ and ω in $\mathbb{N} - \{0\}$ is called arithmetical line with slope a/b, with shift parameter μ and with thickness ω. It is denoted by $D(a, b, \mu, \omega)$.

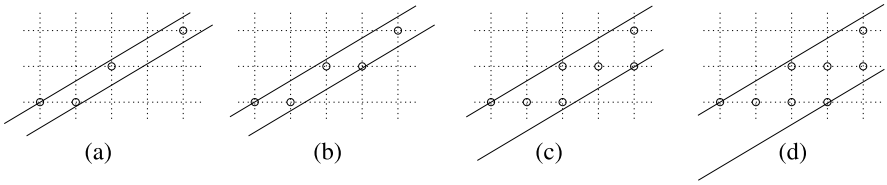


Fig. 10.1 Lines with $a = 3$, $b = 5$, $\mu = 0$: (**a**) $\omega = 4$ (**b**) $\omega = 5$ (**c**) $\omega = 8$ (**d**) $\omega = 10$

Informally, the arithmetical lines correspond to all the points with integer coordinates in a strip limited by the lines $ax - by = \mu$ and $ax - by = \mu + \omega - 1$. The thickness ω corresponds to the number of lines of slope a/b which goes through the integer points of the strip (see Fig. 10.1).

The width parameter ω controls the thickness but also the connectivity of the line. Indeed, arithmetical lines are:

(a) disconnected if $\omega < \max(|a|, |b|)$,
(b) strictly eight connected if $\omega = \max(|a|, |b|)$,
(c) strictly four connected if $\omega = |a| + |b|$ and
(d) thick and four connected if $\omega > |a| + |b|$.

Definition 2 A discrete segment is a finite subset of an arithmetical line.

If the line is connected, so is the discrete segment. The recognition problem is the problem of deciding whether or not a given set of points of $\mathbb{Z}^2$ is a discrete segment. This problem was solved by Debled and Reveillès [5] for the cases (b), (c). The complexity of the recognition process is $\mathcal{O}(n)$ where n is the number of points. Their algorithm is incremental meaning that points are added one by one. An extension of the algorithm has been given by Vialard [24] to add points on both sides of the segment. It can be easily proved that points can also be suppressed on both sides. All algorithms compute parameters of a line containing the given set of points.

10.2.3 The Tangential Cover

We wish to represent 2D digital curves using digital primitives. We expect such a representation to provides important geometric information about digital curves. To do so, we introduce the tangential cover of a digital curve and we refer to [15] and [21] for definitions and properties of digital segments.

Definition 3 ([9]) The tangential cover (TC) of a digital curve $\mathscr{C}$ is the set of all maximal digital straight segments of connected subsets of $\mathscr{C}$ with respect to the inclusion order.

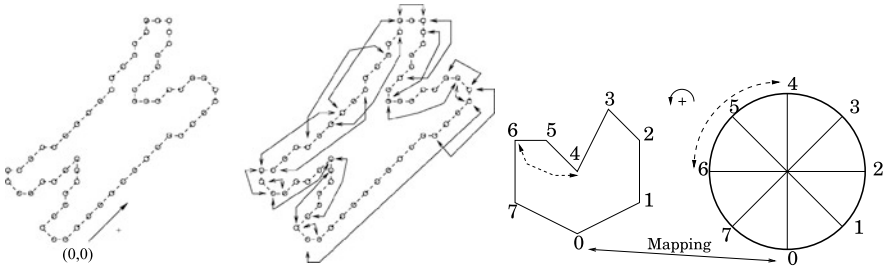


Fig. 10.2 (*Left*) The TC is the set of all maximal segments. (*Right*) The mapping onto the unit circle

The TC may be mapped onto a circular arcs graph. A 2D closed digital curve is homeomorphic to a circle. Since the points of the curve are indexed in order, it is possible to map each point of the curve to a point on the unit circle, as illustrated in Fig. 10.2. Each segment of the TC then corresponds to an arc on the circle. In Fig. 10.2 (right), let us consider that a segment exists between points 4 and 6. This segment is then mapped onto an arc of the circle.

This representation has proven itself to be a very useful and efficient tool for digital curves analysis. For instance, it has been used to solve the Min-DSS problem [10]. It has also been used to estimate global digital curvature [13] and tangents along digital contours [16]. In its primary form, the TC could only manage the extraction of maximal DSS (digital straight segments as in [15]). However, since the tangential cover can be embedded in a circular arcs graph, we could apply classical algorithms on the graph instead of algorithms on the digital curve. For instance, a polygonalization of a digital curve is a set of maximal segment sharing their ending points and covering the curve. Hence, each polygonalization is a path or a cycle—for closed curves—in the graph. Thus, the Min-DSS problem which consists in computing a minimal length polygonalization corresponds to a shortest path or a shortest cycle in the graph.

Among the properties used to build the tangential cover, two must be absolutely retained. First, it is important to notice that the curve is viewed as a totally ordered list of points. Second, it appears that the geometric primitives must be ordered by inclusion. As such, it appears that not all geometric primitives will lead to efficient algorithms. Indeed, the linear complexity for the computation of the tangential cover mainly comes from the fact that a digital straight segment is maximal whenever it becomes impossible to add a point in both sides without losing the DSS property. We will use these properties to generalize the tangential cover in Sect. 10.5.

10.3 Generic Shape Representation

The digital curve model is not enough powerful to represent all shapes encountered in practice. For instance, it avoids the possibility of self-intersections or of thick

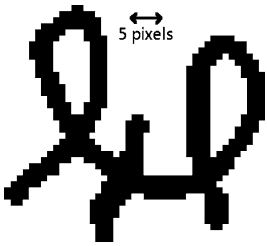


Fig. 10.3 Hand-drawn sample shape

curves. It is known that in case of noisy object, it is a good strategy to enclose the noisy boundary into an inner and an outer boundary [19]. In this case, the representation of shapes is called an outline that is two digital curves, one containing the other and such that the boundary is between the curves. As such, the outline is in general of variable thickness and the exact position of the boundary inside the outline is unknown. However, self-intersection is not allowed to define the inner and the outer curves. In the most general case, the shape is simply a connected component as complex as possible.

Since the analysis aims at organizing shapes into geometric primitives, it appears that the thickness of the curve must be taken into account both in the representation of shapes and of geometric primitives. In the following, we present the general model for connected components and in the next section, geometric primitives allowing thick objects will be presented.

10.3.1 Shapes as Connected Components

We are focusing on raw black and white digital images such as Fig. 10.3. We wish to obtain a decomposition of each shape within such images into sets of geometric primitives which should capture geometric features of the shapes. We are then confronted with the presence of noise in the raw data, which is inherent to most digital images. Such noise can either be due to the acquisition technique or to the nature of the image itself. Noise produces irregularities, which usually prevent the recognition of meaningful geometric features. Thus some pre-processing techniques must be performed before the recognition in order to clean the shape. We provide a method which is robust to noise and may be applied to raw shapes without the help of any of these usual pre-processing techniques. The most straightforward way to analyze such images is to extract the set of all their connected components and to take into account the interior of the shape. This feature may be achieved by computing a constrained Delaunay triangulation of the connected component, as in [7, 22]. Contours are extracted using a simple marching technique on the inter-pixel boundaries and the triangulation is performed with the set of boundaries as constrained edges. The shape is then decomposed into the set of all the computed triangles (see Fig. 10.4). Due to the constraints, all vertices of the triangles consist of contour points (no Steiner points are added) and the set of edges perfectly

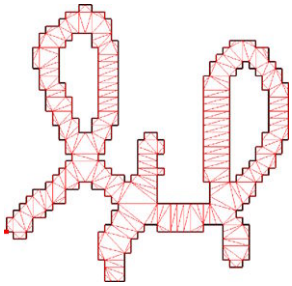


Fig. 10.4 Constrained Delaunay triangulation of the sample shape

represent the variability of the shape. Moreover its intrinsic thickness is taken into account.

10.3.2 α-Path on Connected Components

As noted previously, we can use a unified strategy for dealing with digital curves and the more general model of shapes as connected components. Indeed, in both cases, there are simple elements, a pixel on the one hand and a triangle on the other hand, and a connectivity relation α on those elements. In the former case, the usual 4- and 8-connectivity are adopted and in the latter case the edge adjacency relation between triangles is used. Thus, in both cases, a shape can be viewed as an α-path. In the sequel, we show how to solve the problem of computing a total ordering of the elements of this α-path.

10.3.2.1 Decomposition into Branches

We wish to compute a total ordering on an α-path. In [7], we presented an algorithm that solves this problem in the case where the input shape is a polygon with one hole. However this algorithm does not manage the general case. This algorithm relied on the edge-neighborhood of the triangles. The shape representation described in [22] also uses the same characterization. A triangle with *three* edge neighbors is called a *junction* triangle. A triangle with only *one* edge neighbor is a *terminating* triangle. Finally, a triangle with *two* edge neighbors is a *branch* triangle. Our idea is to decompose the shape into a set of branches, using Algorithm 1.

This simple algorithm produces a set of branches which always begin with a *junction* triangle and may either end with another *junction* triangle or a *terminating* triangle. If the algorithm is processed on the raw set of triangles, it generates a lot of small branches caused by structurally unimportant features. Thus we use a geometric pruning method to get rid of these, as in [20, 22]. This pruning method is based on a morphological significance ratio ρ_0 which must be fixed in advance. ρ_0 may be related to the desired α thickness value chosen for the subsequent primitive recognition. Once the set of triangles has been pruned accordingly, Algorithm 1 is performed. The result on the sample shape is exhibited in Fig. 10.5.

Algorithm 1: Decomposition into branches

```
StartNewBranch;
T ← Find any Junction triangle ;
Add(T,Branch) ;
Mark(T);
Push(Neighbors(T),Stack);
while !Empty(Stack) do
    T ←Pop(Stack) ;
    Add(T,Branch) ;
    Mark(T) ;
    if T is a Branch triangle then
        T ← Unmarked(Neighbors(T))
    else
        if T is a Junction triangle then
            Push(Unmarked(Neighbors(T)),Stack)
        StartNewBranch;
```

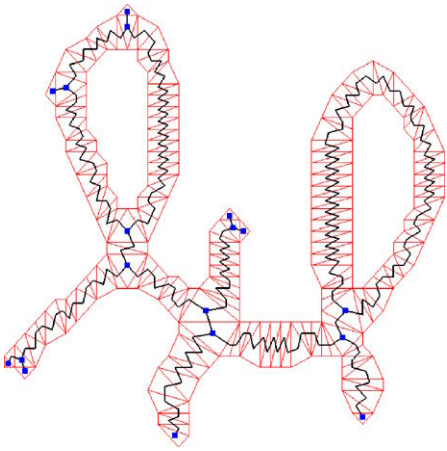


Fig. 10.5 Branches obtained from the pruned triangulation. The *dots* represent the endpoints of each branch

10.3.2.2 Ordering the Branches

Our shape is now represented by a set of branches. We must create a path which joins those branches in order. In this section we explain how to obtain an optimal traversal so that back and forth moves are avoided as much as possible along the branches. The problem is to find a path that relies all the branches with a minimal cost. In graph theory, this problem is known as the Chinese Postman Problem [11]. It is rather straightforward to convert our set of branches into an instance of the Chinese

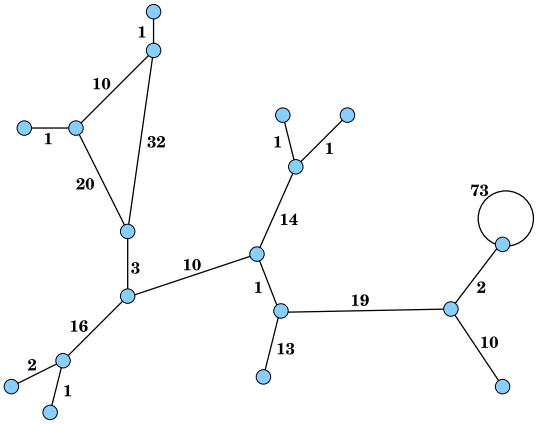


Fig. 10.6 Graph representation of the decomposition

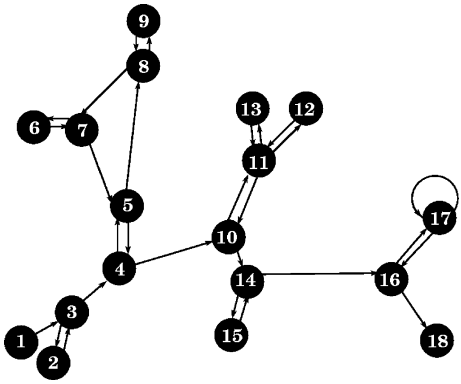


Fig. 10.7 The minimal cost traversal: 1-3-2-3-4-5-8-9-8-7-6-7-5-4-10-11-12-11-13-11-10-14-15-14-16-17-16-18, for a total weight of 267

Postman Problem. Each endpoint of the branches becomes a vertex of the graph, and each branch itself becomes an edge. Each edge is weighted by the number of triangles contained in the corresponding branch. For the sample shape, the graph is exhibited in Fig. 10.6. Then we apply an algorithm given in [11] to obtain a minimal cost traversal of the branches (see Fig. 10.7).

10.4 Geometric Primitives

The DSS model not only concerns straight part exclusively but also is not suitable for dealing with thick shapes. Hence, we now introduce a new notion of straightness which quite resembles the original one of DSS but also deals with circular primitives in the same vein. The main idea is to cover both cases of thin and thick shapes.

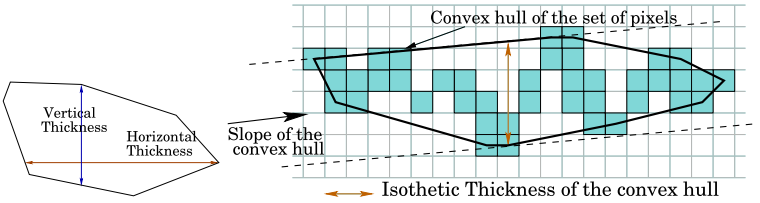


Fig. 10.8 (*Left*) The illustration of the isothetic thickness of a convex hull. (*Right*) An α-blurred straight segment, with $\alpha = 5$

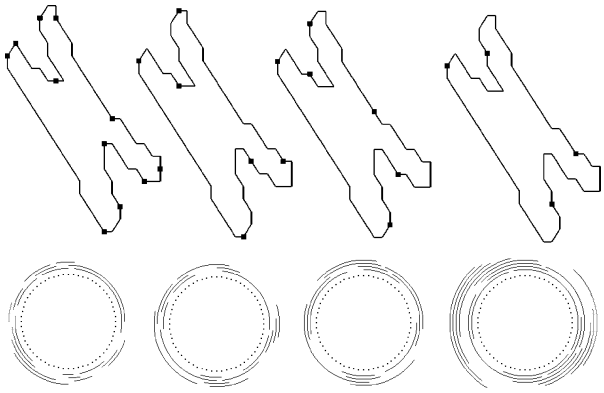


Fig. 10.9 The chromosome shape, with four different thickness values. $\alpha = 1, 2, 3, 4$ pixels. The *dots* represent a good polygonalization for each α, and on the *bottom* lie the associated tangential covers, represented by their *circular arcs* graph

10.4.1 Blurred Straight Segments

The arithmetical definition of DSS, initiated by Reveillès in [21], is a rather rigid one with regards to noise. If the processed shape is somewhat irregular, the recognized segments may be a lot shorter than expected. So we rely on an extended definition to overcome this drawback.

Definition 4 ([4]) A set of points S is an α-blurred straight segment if and only if the isothetic thickness of its convex hull is less than a given real number α. The isothetic thickness of a convex hull is the minimum between its horizontal width and its vertical height.

Figure 10.8 shows an example of an α-blurred straight segment. This definition allows us to break the rigidity of arithmetical DSS. The value of α is a parameter and must be fixed in advance. An efficient algorithm which tests if a set of points is an α-blurred straight segment is given in [2]. Its time complexity is $\mathcal{O}(\log n)$. In Fig. 10.9, the tangential cover using blurred straight segments is shown by various

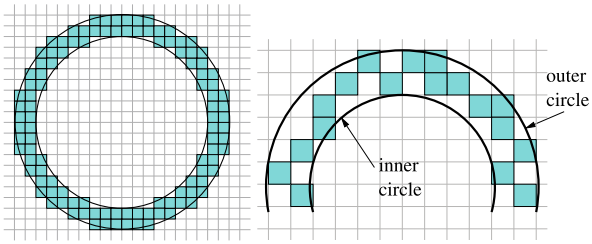


Fig. 10.10 (*Left*) A digital ring. (*Right*) An α'-thick digital arc, with $\alpha' = 2$

α values. We can remark that as expected, the complexity in number of maximal geometric primitives depends on the chosen parameter α.

10.4.2 Digital Circular Arcs and Annulus

We now wish to extend the model of parameterized thickness to a circular digital primitive. In [12], Hilaire and Tombre define a fuzzy digital arc as a set of pixels "close enough" to a circular arc of the real plane, with some constraints. These constraints limit the recognition of digital arcs to rather thin ones, which do not suit our usage. Hence we use another approach. According to the work of Andrès [1], a digital ring is the set of all digital points comprised between two concentric Euclidean circles (see Fig. 10.10). Our definition is related to the one of digital ring and is a classical annulus.

Definition 5 A set of pixels S is an α'-thick digital arc if and only if it is completely enclosed between two Euclidean circles $\mathscr{C}_0$ and $\mathscr{C}_1$ with common center o and whose difference of radii $dr = r_1 - r_0$ is less than a given real number α', having $r_1 > r_0$.

An example of an α'-thick digital arc is given in Fig. 10.10. Contrary to digital rings, some holes may appear in α'-thick digital arcs. In other words, it is not necessary for all digital points comprised between $\mathscr{C}_0$ and $\mathscr{C}_1$ to be in S for it to be an α'-thick digital arc. We wish to determine whether a given set of pixels S is a part of an α'-thick digital arc. This problem is equivalent to finding the minimum-width enclosing annulus of S, and comparing its width to α'. There exists such an algorithm within the Computational Geometry Algorithm Library CGAL for instance. Its time complexity is $\mathscr{O}(n)$.

10.5 The Predicate Cover

In previous works, we generalized the definition of the TC [7] in order to process different primitives. According to previous section, all shapes will be seen as a totally ordered α-paths and as such we can use the inclusion order between subpaths.

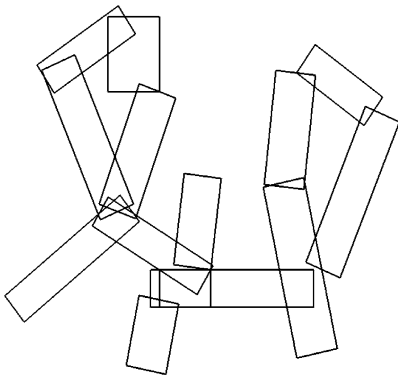


Fig. 10.11 The decomposition of the whole shape in the order of the branches, $\alpha = 5$ pixels

Definition 6 Let C be a totally ordered α-path, and let P be a binary predicate of validity. The predicate cover (PC) of C is the set of all maximal valid subpaths of C. Validity is intended with respect to P and maximality is intended with respect to the inclusion order.

The generality of the notion of subpaths and its associated predicate of validity allow us to process various digital primitives. In this chapter, our goal is to represent digital curves using circular and straight digital primitives simultaneously. It should be noticed that since subpath is an α-connected subpart of the path, clearly the number of valid subpaths is at most $\mathcal{O}(n^2)$. This bound is also valid when taking into account the predicate P.

Consider for example a computation with the predicate cover. We can extract a polygonalization of a connected component. We can traverse branches during the recognition to merge branches (see Fig. 10.11). Visual linear parts are well recognized and it is possible to cluster the segments to obtain meaningful subparts of the original shape.

From these experiments, we also conclude that in the study of shapes, the local thickness is well captured by our algorithm. If parts of the center path of the shape is linear, we can capture it only if the thickness is rather constant with respect to the bound α.

10.6 Multi-primitives Analysis

The case of computing the predicate with one geometric primitive is somewhat a direct extension of the classical tangential cover. However, the most important property of the predicate cover is that the model is valid for several primitives. Hence, it is straightforward to look at the decomposition of a shape with several primitives. The principle of the analysis rely on a competition among the different primitives while moving on the shape. As previously noted, the algorithm is more easily understood on the associated circular arcs graph than on the shape. We thus use this

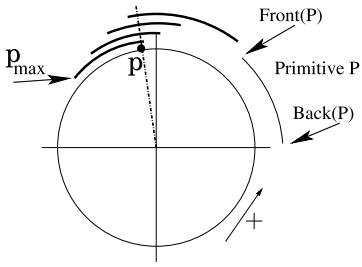


Fig. 10.12 The *front*(P) and *back*(P) elements are illustrated on a primitive P. The *bold circular arcs* represent the primitive pencil $\mathscr{P}\mathscr{P}(p)$ of the point with index p

model. It should be noted that more than two primitives might be considered simultaneously without changing the method.

10.6.1 α-Blurred Straight Segments Versus α′-Thick Digital Arcs

We suppose that the predicate cover has been computed both for α-blurred segments and for α'-thick arcs on a shape $\mathscr{C}$. Our goal is to obtain a decomposition of $\mathscr{C}$ into both segments and arcs, using both predicate covers. This decomposition should capture geometric features of the shape. Obviously, straighter parts should be represented by segments and more circular ones should be represented by arcs. It resembles a polygonalization, but with several primitives. For given α and α' values, there exists a large amount of possible decompositions. The optimal decomposition(s) should be the one(s) which contain the minimum number of primitives. Our method should be able to compute the optimal decomposition, or at least reasonably approach it.

Let us first describe how to obtain a polygonalization with only one primitive using the predicate cover. This algorithm mimics the classical algorithm of shortest path in a graph but will be simply modified for our purpose. The shortest path may be computed easily. To do this, let us define the notion of *primitive pencil* $\mathscr{P}\mathscr{P}(p)$ of any element, in the path describing the shape, with index p. $\mathscr{P}\mathscr{P}(p)$ is the set of all primitives that contain p (see Fig. 10.12). The *front*(P) and *back*(P) elements are the start and end points of a primitive P. Remind that the graph is oriented with respect to the orientation of the path. Given the orientation, we easily determine the primitive $P*$ of $\mathscr{P}\mathscr{P}(p)$ that reaches the point with index $p_{\max} = front(P*)$ that is the farthest away from p on the path. This point $p_{\max}$ defines a function $f()$, that is $f(p) = p_{\max}$. This $f()$ function is computed in linear time, since we only need to compute it for the beginning and ending points of each primitive.

The indices of the vertices of the polygonalization of $\mathscr{C}$ starting at point index p are obtained by iterates of the function $f()$. More formally, $vp_i = f^i(p)$ with vp_i being the index of the i-th vertex of the polygonalization, and with $f^i(p)$ being the i-th iterates of the function $f()$. We use this process for both predicate covers, building two functions $f_S()$ and $f_A()$ respectively for the segments PC and for the arcs

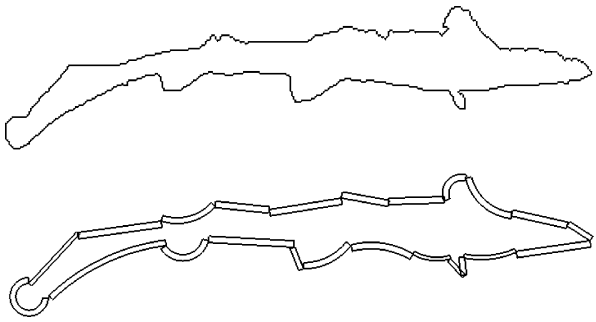


Fig. 10.13 The original shape (*top*) has been decomposed into α-blurred straight segments and α'-thick digital arcs (*bottom*), with $\alpha = \alpha' = 4$ pixels

PC. In this scope, a multi-primitive decomposition of a shape may be seen as a series of applications of the functions $f_X()$. To be more precise, such a decomposition is a word on the alphabet $\{f_A, f_S\}$.

A simple multi-primitive decomposition may be deduced straightaway. We map both predicate covers onto the same circular arcs graph and use the following greedy algorithm. We choose a starting point with index p_0. We have $f_S(p) = p_{Smax}$ and $f_A(p) = p_{Amax}$. If $p_{Amax} > p_{Smax}$ we choose the corresponding arc, else we choose the segment. The process stops when the point with index p_0 is reached again. This means that the decomposition is complete. The shape is then reconstructed using the chosen primitives and their associated parameters. An example is presented in Fig. 10.13. In the case of open curves, the only difference is that the starting point is necessarily one of the endpoints, depending on the orientation.

However, this method does not allow us to control the quality of the solution (its proximity to the optimum). Moreover, we do not have any information on the behavior of the decomposition with regards to the used primitives. Due to their construction, the two types of predicate covers contain all possible decompositions of the curve. To obtain a guarantee on the quality of the solution, a better method is to build the complete tree of all possible decompositions. We describe this process in the next subsection.

10.6.2 Building the Complete Tree

Our idea is the following: from a given point with index p on the path, we may either choose the longest segment or the longest arc covering p (longest meant as the result of the $f_X()$ function described previously). Thus there exists two different possible decompositions starting from p. Then the same process is applied to both results, and so on. This process defines a binary tree. The root of the tree is the chosen starting point with index p_0 (see the next subsection). A node of the tree is a partial decomposition of the path from p_0 to the current index p defined by any of the $f_X()$ functions. The process stops when $f_X(p) \geq p_0$, and the current node is then a leaf of the tree. Such a leaf is a complete decomposition of the path, and a backtracking from the leaf to the root gives us the corresponding word on

Table 10.1 This table shows some statistics on the full decomposition tree

Curve	*	Chromosome (61 pts)	Leaf (119 pts)	Semicircle (103 pts)
$\alpha = 1$	$\#D$	4323	NA	3811
	$\#MinD$	20	NA	1
	$\|MinD\|$	9	NA	5
$\alpha = \sqrt{2}$	$\#D$	964	255744	810
	$\#MinD$	10	504	1
	$\|MinD\|$	8	15	5
$\alpha = 2$	$\#D$	150	11768	192
	$\#MinD$	18	88	1
	$\|MinD\|$	6	11	4
$\alpha = 2\sqrt{2}$	$\#D$	72	4480	127
	$\#MinD$	56	672	1
	$\|MinD\|$	6	11	4
$\alpha = 4$	$\#D$	25	175	32
	$\#MinD$	1	11	2
	$\|MinD\|$	3	6	3

*To read the table: $\#D$ is the number of possible decompositions; $\#MinD$ is the number of minimal decompositions in terms of number of primitives; $|MinD|$ is the number of primitives of minimal decompositions. $\alpha = \alpha'$ for the whole table

the alphabet $\{f_A, f_S\}$. We build the whole tree and analyze the results. We focus on the leaves with the smallest tree depth. Those leaves constitute a pool of minimal decompositions in terms of number of primitives. We show some statistics in Table 10.1. Three classical shapes are dealt with at different thickness values. We use a $(\sqrt{2})^i$, $i \in \{0, \ldots, 4\}$ growth model for the thickness, having $\alpha = \alpha' = (\sqrt{2})^i$. For instance, for the chromosome shape and for $\alpha = \alpha' = 1$, there exists 4323 possible decompositions. Upon these decompositions, a pool of 20 of them contain only 9 primitives, which is the minimal value.

The building of the full tree allows us to better understand the behavior of the decompositions. However its main drawback is of course the combinatorial explosion that results. As exhibited in Table 10.1 for the leaf shape, which is a small curve (only 119 points) but rather irregular, the number of possible decompositions explodes for small α values. Thus its usage is limited to rather small curves for behavior studies only, and should be avoided otherwise. In the next section, we describe our way to avoid the full deployment of the tree while still obtaining a good, if not optimal, decomposition.

10.6.2.1 The Choice of the Starting Point

Choosing an arbitrary starting point index p_0 does not allow us to warrant the optimality of our decompositions. Anyway, we may improve the results by finding a "good" starting point. The best choice would be the endpoint of some primitive, in order to begin with a maximal primitive. But we are working with two types of primitives, so a good starting point for one primitive is not necessarily good for the other one. We propose a quite simple method. $f_X(p) - p$ represents the length of the useful part of the primitive. For each point index p of the curve, we compute $Q_p = f_A(p) - p + f_S(p) - p$ and choose p_0 such that Q_{p_0} is maximal. This way, both primitives are taken into account for the choice of p_0.

10.6.3 A Partial Tree

We describe our method to avoid the full deployment of the tree while maintaining good results. Our approach is related to *branch and bound* techniques. We use a criterion to evaluate each node. All nodes which are not leaves and whose children have not been explored yet are considered *"open nodes"* suitable for evaluation. We only explore the node with the best criterion value upon the pool of open nodes. The process stops when the first leaf is built. We describe our criterion in the following.

10.6.3.1 Our Criterion

The criterion we use to evaluate the open nodes is based on the notion of "covering rate" $Cr(n)$ of a node n. Remind that each node is a partial decomposition of the path (see Sect. 10.6.2). Thus it covers a part of the path. We compute the number of points $Q(n)$ covered until the current node, and divide this number by the tree depth $D(n)$ of the node. So the formula for the criterion is: $Cr(n) = \frac{Q(n)}{D(n)}$. It is the average value of the number of points contained in a primitive. The node with maximal $Cr()$ value upon the pool of open nodes is the one to explore. Thus at each step, we deploy the branch that has the best chance to cover the entire curve with the best covering rate. For instance let us take a look at Fig. 10.14 (left). The decomposition starts with point index p_0. Each branch is labeled with the nature of its primitive (A for circular arc, S for segment), and each node is labeled with its covering rate $Cr()$. The pool of open nodes is represented, and the black node is the one that has the best covering rate. Hence it is the one that will be explored next. On the rightmost figure, the node has been explored and the pool of open nodes has been updated accordingly. The best node is now located in another branch. This process allows us to go back to previous unexplored nodes with lower depth if needed. $Cr(n)$ is computed in constant time at each node creation.

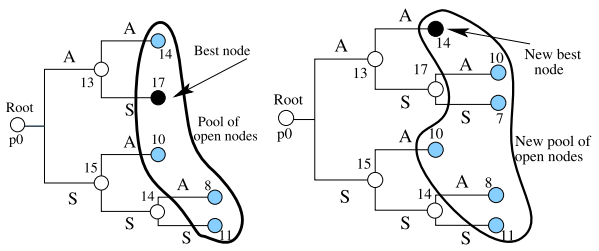


Fig. 10.14 Here is an example of the partial exploration of the tree based on our covering rate criterion

10.6.4 Experimental Results

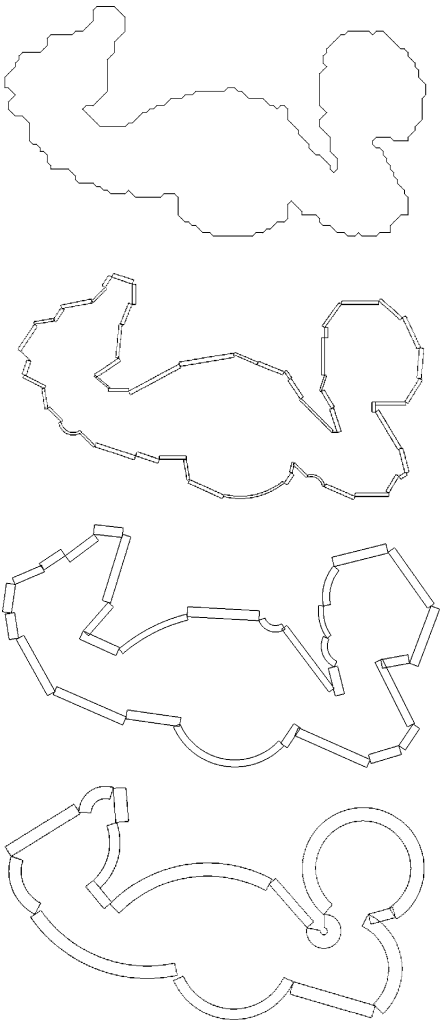
We applied our method to a series of digital curves known as the *SQUID* database [18]. In Fig. 10.15 are exhibited several decompositions of a hippocampus shape. We used various thickness values, from $\alpha = \alpha' = 1$ to $\alpha = \alpha' = 4$ pixels. It is interesting to see how the local geometric features are captured at each different thickness value. For instance, the tail of the hippocampus is recognized as a circular part at thickness $\alpha = \alpha' = 2$, then it is again a series of straight parts for greater thickness values, and finally for $\alpha = \alpha' = 4$ the circular structure reappears. Let us remark that the rate of circular arcs increases along with the thickness. Indeed, for small thickness values, the definition of digital circular arcs is too arithmetically constrained to result in long primitives when the input shapes are somewhat irregular. A greater α value allows for a greater tolerance to irregularities, and more circular parts appear. This behavior is illustrated in the accompanying table of Fig. 10.15.

To illustrate the quality of our greedy algorithm, we decomposed several subsampled digital shapes and compared the results using the full tree and the partial tree. These results are exhibited in Fig. 10.16. Those curves have been sub-sampled to approximately 200 pixels each in order to deploy the full tree. The accompanying statistics table of Fig. 10.16 shows that the greedy algorithm approaches or equals the minimal decomposition size. We conjecture that a greedy algorithm may guarantee an upper bound of one extra primitive with regards to the minimal size, but this has yet to be proven.

10.7 Future Work and Open Problems

Future work should be focused on multi-scale analysis of shapes, that is using varying scale for the parameters introduced in the geometric primitive characterization. Some work has already been done in this direction [8, 14], but more is needed to understand precisely how to manage those varying scales. As a consequence on this research direction, the complexity of the multi-scale analysis should be evaluated carefully in order to avoid to simply rerun current algorithms as many times as there are scales to analyze. This problem has not been tackled yet, but deserves attention in the future.

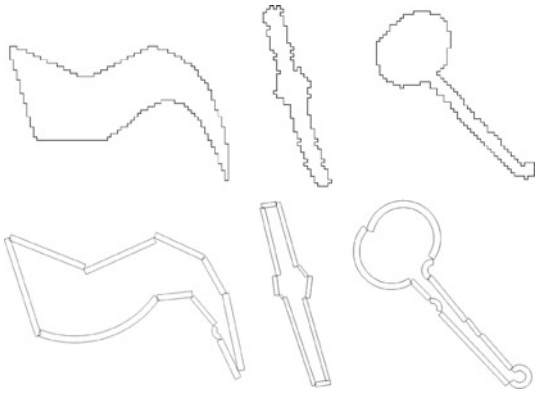
The main open problems concern the dimension of the shapes. Indeed, it is not difficult to see that the presented framework can easily be extended to the case



Thickness	$\alpha = 1$	$\alpha = \sqrt{2}$	$\alpha = 2$	$\alpha = 2\sqrt{2}$	$\alpha = 3$	$\alpha = 4$
Size of the decomposition	86	57	39	30	25	15
Ratio arcs/segments	2.3 %	5.3 %	12.8 %	20 %	28 %	60 %

Fig. 10.15 The shape is shown on *top*. Then we show the results of our multi-primitive decompositions for various thickness values increasing from *left* to *right* and from *top* to *bottom* (note that $\alpha = \alpha'$ for all decompositions). These values are exhibited on the *bottom table*, along with some statistics

of 1-dimensional geometric primitives in 3D and more generally in nD. This is because, due to the graph structure, the presented framework is not dependent on any notion of dimension. However, it relies on the fact that geometric primitives are one



Shape	*	Comma	Watch	Spoon
$\alpha = 2$	$\#D$	166264	61740	125436
	$\#MinD$	4	48	1
	$\|MinD\|$	10	10	9
	$\|Greedy\|$	11	10	10

Fig. 10.16 Three sub-sampled digital shapes used to test the quality of our greedy algorithm. From *left* to *right*: "Comma", "Watch", "Spoon". $\alpha = \alpha' = 2$. (*Top*) The shapes. (*Middle*) Decompositions using the greedy algorithm. (*Bottom*) Statistics table: $\#D$, $\#MinD$ and $|MinD|$ have the same meaning as in Table 10.1; and $|Greedy|$ is the number of primitives of the greedy algorithm decompositions

dimensional to ensure that the total ordering of the elements of a shape is necessary and sufficient for the analysis. This property becomes false with 2D primitives such as planes since there is no more a total ordering on the elements adapted to the recognition of the primitives. Moreover, the predicate cover heavily relies on the inclusion order, which is a total order for 1D primitives but becomes a partial order for nD primitives. We should also add that the computation of the minimal number of planes which covers a digital object is an NP-Complete problem [23]. We refer the reader to the work in [3] for a first attempt to approach the problem.

A second open problem concerns overlapping shapes. Indeed, no overlapping is considered in the framework. However, in practice, it is necessary to try to recover a shape with a minimal number of subparts built from the geometric analysis of the primitives. For instance, the framework due to the total ordering does not consider that some parts might be matched in order to detect coherent substructure. Solving this problem requires to decompose a shape from the geometric analysis with the requirement that an element of the shape might be associated with several subparts.

A third open problem concerns the complexity of the analysis. It is important to notice that our approach has only one guarantee which is the quadratic bound on the number of valid subpaths in each predicate cover. But sometimes, only a linear number of such subpaths exists, for instance, for DSS. Thus, some properties on the predicate must be added to separate quadratic analysis form linear ones. For

instance, it is straightforward to see that the Sum-of-Squared-Error commonly used in polygonal approximations is a predicate which leads to a quadratic number of elements in the associated predicate cover. This explains why, in practice, polygonal approximation algorithms relying on this predicate are quadratic at least, when they do require optimality.

References

1. Andres, E.: Discrete circles, rings and spheres. Comput. Graph. **18**(5), 695–706 (1994)
2. Buzer, L.: Digital line recognition, convex hull, thickness, a unified and logarithmic technique. In: Proceedings of the 11th IWCIA, Berlin, Germany. Lecture Notes in Computer Science, vol. 4040, pp. 189–198 (2006)
3. Charrier, E., Lachaud, J.-O.: Maximal planes and multiscale tangential cover of 3D digital objects. In: Combinatorial Image Analysis, 14th International Workshop, IWCIA. Lecture Notes in Computer Science, vol. 6636, pp. 132–143. Springer, Berlin (2011)
4. Debled-Rennesson, I., Feschet, F., Rouyer-Degli, J.: Optimal blurred segments decomposition of noisy shapes in linear time. Comput. Graph. **30**(1), 30–36 (2006)
5. Debled-Rennesson, I., Reveillès, J.-P.: A linear algorithm for segmentation of digital curves. Int. J. Pattern Recognit. Artif. Intell. **9**(4), 635–662 (1995)
6. Faure, A., Buzer, L., Feschet, F.: Tangential cover for thick digital curves. Pattern Recognit. **42**, 2279–2287 (2009)
7. Faure, A., Feschet, F.: Robust decomposition of thick digital shapes. In: Proceedings of the 12th IWCIA, Buffalo, USA. Lecture Notes in Computer Science, vol. 4958, pp. 148–159 (2008)
8. Faure, A., Feschet, F.: Linear decomposition of planar shapes. In: 20th International Conference on Pattern Recognition, ICPR, pp. 1096–1099. IEEE Press, New York (2010)
9. Feschet, F.: Canonical representations of discrete curves. Pattern Anal. Appl. **8**(1–2), 84–94 (2005)
10. Feschet, F., Tougne, L.: On the min DSS problem of closed discrete curves. Discrete Appl. Math. **151**(1–3), 138–153 (2005)
11. Hajdu, A., Pitas, I.: Piecewise linear digital curve representation and compression using graph theory and a line segment alphabet. IEEE Trans. Image Process. **17**(2), 126–133 (2008)
12. Hilaire, X., Tombre, K.: Robust and accurate vectorization of line drawings. IEEE Trans. Pattern Anal. Mach. Intell. **28**(6), 890–904 (2006)
13. Kerautret, B., Lachaud, J.-O.: Curvature estimation along noisy digital contours by approximate global optimization. Pattern Recognit. **42**(10), 2265–2278 (2009)
14. Kerautret, B., Lachaud, J.-O.: Multi-scale analysis of discrete contours for unsupervised noise detection. In: Combinatorial Image Analysis, 13th International Workshop, IWCIA. Lecture Notes in Computer Science, vol. 5852, pp. 187–200. Springer, Berlin (2009)
15. Klette, R., Rosenfeld, A.: Digital Geometry: Geometric Methods for Digital Picture Analysis. Computer Graphics and Geometric Modeling. Morgan Kaufmann, San Francisco (2004)
16. Lachaud, J.-O., Vialard, A., de Vieilleville, F.: Fast, accurate and convergent tangent estimation on digital contours. Image Vis. Comput. **25**(10), 1572–1587 (2007)
17. Massod, A., Sarfraz, M.: An efficient technique for capturing 2D objects. Comput. Graph. **32**, 93–104 (2008)
18. Mokhtarian, F., Abbasi, S.: Shape similarity retrieval under affine transforms. Pattern Recognit. **35**(1), 31–41 (2002)
19. Nguyen, T.P., Debled-Rennesson, I.: A discrete geometry approach for dominant point detection. Pattern Recognit. **44**(1), 32–44 (2011)
20. Prasad, L., Rao, R.: A geometric transform for shape feature extraction. In: Vision Geometry 2000. Proceedings of SPIE, vol. 4117, pp. 222–233 (2000)

21. Reveilles, J.-P.: Geometrie discrete, calcul en nombres entiers et algorithmique. These d'Etat (1991)
22. Schlei, B.R.: A new computational framework for 2D shape-enclosing contours. Image Vis. Comput. **27**(6), 637–647 (2009)
23. Sivignon, I., Coeurjolly, D.: Minimal decomposition of a digital surface into digital plane segments is NP-hard. In: Discrete Geometry for Computer Imagery, 13th International Conference, DGCI. Lecture Notes in Computer Science, vol. 4245, pp. 674–685. Springer, Berlin (2006)
24. Vialard, A.: Geometrical parameters extraction from discrete paths. In: 6th International Workshop DGCI. Lecture Notes in Computer Science, vol. 1176, pp. 24–35. Springer, Berlin (1996)
25. Wagenknecht, G.: A contour tracing and coding algorithm for generating 2D contour codes from 3D classified objects. Pattern Recognit. **40**, 1294–1306 (2007)
26. Zhang, D., Lu, G.: Review of shape representation and description techniques. Pattern Recognit. **37**(1), 1–19 (2004)

Chapter 11
Shape from Silhouettes in Discrete Space

Atsushi Imiya and Kosuke Sato

Abstract Reconstruction of an object from a series of silhouettes is obtained through a binary geometric tomography technique since both the objects and the projections, which are measured as a series of silhouettes, are binary. In this paper, we formulate the method *Shape from Silhouettes* in two- and in three-dimensional discrete space. This approach to the problem derives an ambiguity theorem for the reconstruction of objects in the discrete space. The theorem shows that *Shape from Silhouettes* in discrete space results in an object which is over-reconstructed, if the object is convex. Furthermore, we show that in three-dimensional space, it is possible to reconstruct a class of non-convex objects from a set of silhouettes although in the plane a non-convex object is not completely reconstructible from projections.

11.1 Introduction

The reconstruction of three-dimensional shapes from measured data, such as range data, photometric information, and stereo image pairs, is called *Shape from X*. In this chapter, we deal with *Shape from Silhouettes*, a problem also called *Shape from Counter* [2], *Shape from Profile* [25, 26] in computer vision, and *Shape from Plane Probing* [5] in computational geometry. The method is a conventional technique for detection of shape models and shape reconstruction in computer graphics [13, 20]

A. Imiya (✉)
Institute of Media and Information Technology, Chiba University, Yayoi-cho 1-33, Inage-ku, Chiba 263-8522, Japan
e-mail: imiya@faculty.chiba-u.jp

K. Sato
School of Science and Technology, Chiba University, Yayoi-cho 1-33, Inage-ku, Chiba 263-8522, Japan
e-mail: Sato.Kosuke@dn.MitsubishiElectric.co.jp

Present address:
K. Sato
Information Technology Systems Dept. Intelligent Transport Systems Engineering Section, Mitsubishi Electric Corporation Kamakura Works, Kamimachiya 325, Kamakura 247-8520, Kanagea, Japan

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_11,

and robotics [16, 22, 25, 26]. Even though these reconstruction methods are mathematically formulated in the continuous framework [9, 10, 20], the reconstruction is achieved in discrete space. Moreover, little attention has been paid on the theoretical analysis of reconstruction algorithms in computer vision.

This chapter aims to introduce a complete discrete version of *Shape from Silhouettes*, a method allowing to reconstruct a discrete object from discrete silhouettes, that is, the camera observing the silhouettes is described as a voxel and the detectors measuring the silhouettes are described as a set of voxels in the three-dimensional discrete space.

In discrete space, a discrete object is reconstructed by line voting [12, 13], that is, it is considered as the intersection of all discrete lines defined by the camera voxels and the silhouette voxels. We prove that this approach leads to an ambiguous reconstruction. Furthermore, we show that a series of silhouettes measured using a camera moving along a circle on a plane is insufficient for the full reconstruction of the visible hull of an object. Although this type of measuring system is sometimes used in computer vision, our results show that we cannot reconstruct the full profile of an object using such a camera system even if the object is convex. Next, we show that we can fully reconstruct a convex object from a series of silhouettes measured through a general stereo system with some simple geometric assumptions.

The illumination problem [3] estimates the minimum and maximum number of view-points for the reconstruction of a convex body from its views from an appropriate set of these view-points. The illumination problem is equivalent to shape reconstruction from silhouettes or shadows. However, it is difficult in general to define the configuration of view-points for a given object. There are many results for the reconstruction of a convex polygon from its shadows (see, for example, [16, 17]). Laurentini [14, 15] was concerned with the geometric properties of silhouette-based shape reconstruction for polyhedra, and clarified the relation between the visible hull and the convex hull of a polyhedron. Tuy [27] proved that for a positive function defined in a finite closed convex region in three-dimensional Euclidean space, it is possible to reconstruct the function from line integrals measured by cone-beams, if the source of the line integrals moves on a pair of circles with the same radius lying on a mutually perpendicular planes which encircle the region. Obtaining the shape from perspective projections is related to the cone-beam reconstruction problem since it is possible to determine the boundary from line integrals. It is possible to decompose a three-dimensional object to a set of slices. The geometric relation between an object and its slices permits to decompose the shadows of a three-dimensional object to shadows of planar objects. Using these geometric relations we prove that a class of non-convex objects is reconstructible from a series of shadows.

11.2 Shape Reconstruction

In this section we describe the reconstruction procedure. We specifically consider the reconstruction of non-convex objects.

11.2.1 Silhouettes and Support Hyperplanes

Let $\mathbb{R}$ be the set of real numbers and $\mathbb{Z}$ be the set of integers; the closed finite subsets of $\mathbb{R}^n$ and $\mathbb{Z}^n$ are called *objects* and *discrete objects*, respectively.

In two- and three-dimensional Euclidean space, $\mathbb{R}^n$, $n = 2, 3$ we define the silhouette of a finite closed region $\mathbf{O}$ from a source $\boldsymbol{s} \in \mathbb{R}^n$ as

$$\Omega(\boldsymbol{s}) = \left\{\omega \middle| l(\boldsymbol{s}) \cap \mathbf{O} \neq \emptyset\right\} \tag{11.1}$$

for the half-line[1]

$$l(\boldsymbol{s}) = \left\{\boldsymbol{x} \middle| \boldsymbol{x} = \boldsymbol{s} + t\omega,\ \omega \in \mathbb{S}^{n-1},\ t \geq 0\right\}, \tag{11.2}$$

where $\mathbb{S}^{n-1}$ for $n = 2, 3$ is the unit circle/sphere in $\mathbb{R}^n$, $n = 2, 3$.

If for all $\boldsymbol{s}$ in $\mathbb{R}^n \setminus K$, where K is a finite convex region in $\mathbb{R}^n$ and $\Omega(\boldsymbol{s})$ is measured, we can reconstruct K as

$$K = \bigcap_{\boldsymbol{s} \in \mathbb{R}^n \setminus K} \left(\bigcap_{\omega \in \Omega(\boldsymbol{s})} \{\boldsymbol{x} | \boldsymbol{x} = \boldsymbol{s} + t\omega\} \right). \tag{11.3}$$

For $n = 2$, the method is equivalent to the reconstruction of K from a set of support lines as the intersection of half-planes is limited by the support lines.

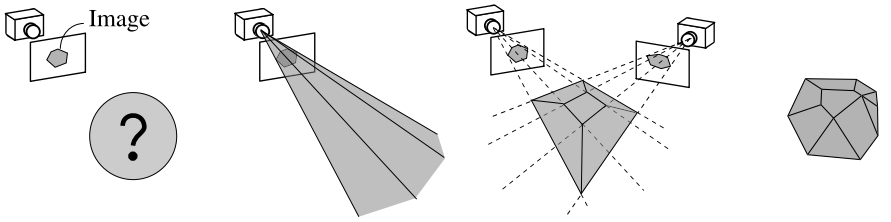
For $n = 3$ and the source $\boldsymbol{s}$, a set $\Omega(\boldsymbol{s})$ in $\mathbb{S}^2$ defines a convex cone. This convex cone defines a set of tangent planes to the cone. The reconstruction of K in $\mathbb{R}^3$ is defined by the support planes as the intersection of half-spaces divided by the support planes. For a finite convex object K in $\mathbb{R}^3$, if we can detect all planes which intersect K, we can obtain all rays which pass through K as the intersection of pairs of planes. These rays defines silhouettes. Therefore, this geometrical property implies that we can reconstruct a finite convex object in three-dimensional Euclidean space from the set of all planes which intersect the object.

Figures 11.1, 11.2, and 11.3 show a procedure to reconstruct an object from shadows. A shadow is a monochrome image on the imaging plane of the camera. A silhouette is the profile of an object derived from a shadow. Figure 11.2 shows the procedure to derive the silhouette from the shadow associated with a fixed light source. A silhouette is derived from a visual cone and the intersection of silhouettes is the visual hull. Therefore, when the number of source point increases, the difference between the object and its visual hull decreases.

Figure 11.4 shows the discrete method for the reconstruction of an object. First, the reconstructed object is expressed as a discrete object on the discrete plane or discrete space. Then, a shadow image of an imaging plane is processed as a discrete binary image.

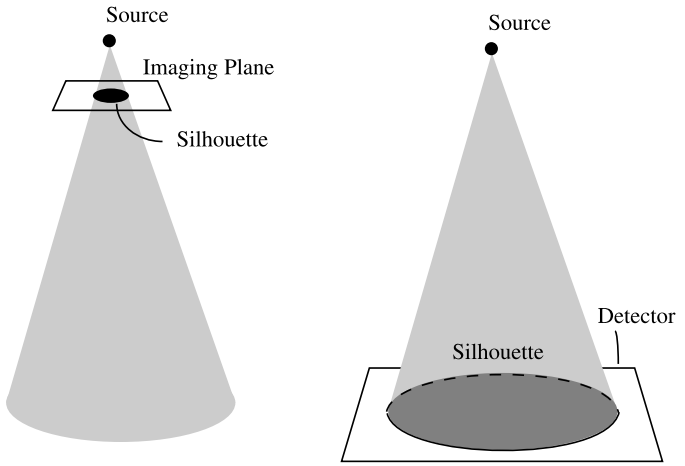
Figures 11.5 and 11.6 show geometric models obtained through *Shape from Silhouettes* in the plane and space, respectively.

[1] The cross-section of the cone $\boldsymbol{l}(\boldsymbol{s})$, $\boldsymbol{s} \in \Omega(\boldsymbol{s})$ with the hyperplane $\boldsymbol{s}^\top \boldsymbol{x} = d$ is geometrically defined as the silhouette generated by source $\boldsymbol{s}$.



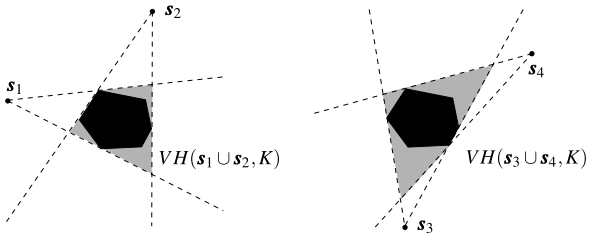
(a) Silhouette of an image. (b) The visual cone associated with a silhouette. (c) The visual hull derived as the intersection of two visual cones. (d) The visual hull.

Fig. 11.1 Reconstruction procedure. (**a**) Imaging system measures a silhouette on the imaging plane. (**b**) A silhouette on the imaging plane defines a visual cone. (**c**) The intersection of two visual cones contains an object. (**d**) A visual hull of an object is the intersection of many visual cones



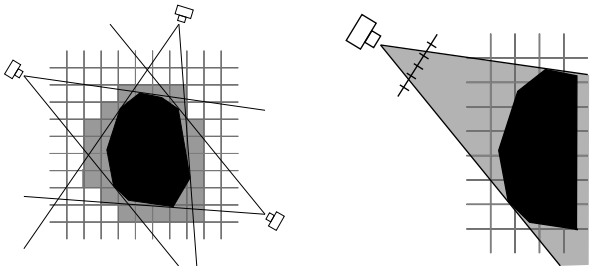
(a) Imaging model. (b) Source-detector model.

Fig. 11.2 Construction of a visual cone. The *silhouette* on the detector plane and the *source point* define a visual cone



(a) Visual hull and an object. (b) The intersection of visual cones.

Fig. 11.3 Geometric configuration of the visual hull and visual cones on the plane. (**a**) The *black convex polygon* is $VH(S, K)$. (**b**) The *gray region* containing $V(H, K)$ is the intersection of visual cones



(a) Discretization of an object.

(b) Discretization of a shadow.

Fig. 11.4 Discrete method for the reconstruction of an object.
(**a**) The reconstructed object is expressed as a discrete object in the discrete plane or space.
(**b**) A shadow image on an imaging plane is processed as a discrete binary image

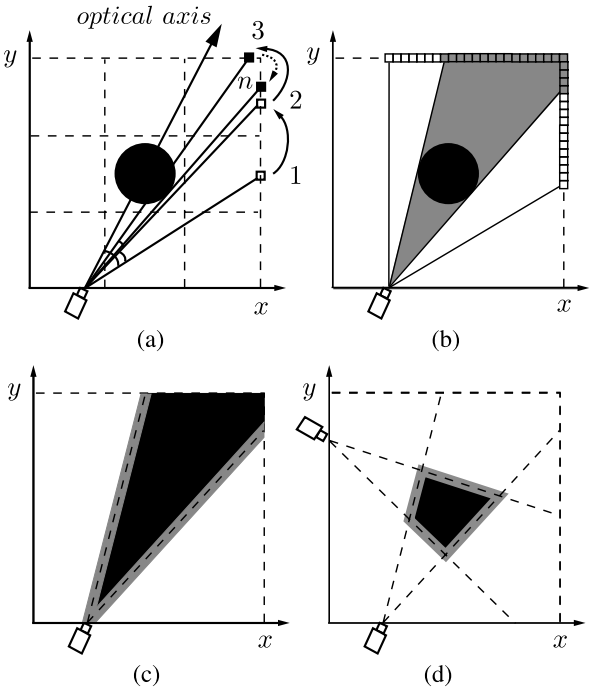


Fig. 11.5 Silhouettes and reconstruction of an object on the plane.
(**a**) Geometry of the detectors and the source for the planar problem.
(**b**) Silhouette of a convex object on a plane.
(**c**) Visible hull of a silhouette.
(**d**) Visible hull of silhouettes

11.2.2 Reconstruction of Non-convex Objects

In this section, we summarize the results of Ref. [10] on the reconstruction of non-convex objects from silhouettes. The following Theorem 1 shows a condition for the full reconstruction of a non-convex object.

Theorem 1 *From the set of silhouettes of an object observed from vertices lying on a sphere encircling the object, we can obtain a set of two-dimensional perspective projections of a slice from a point which moves on a circle encircling the object.*

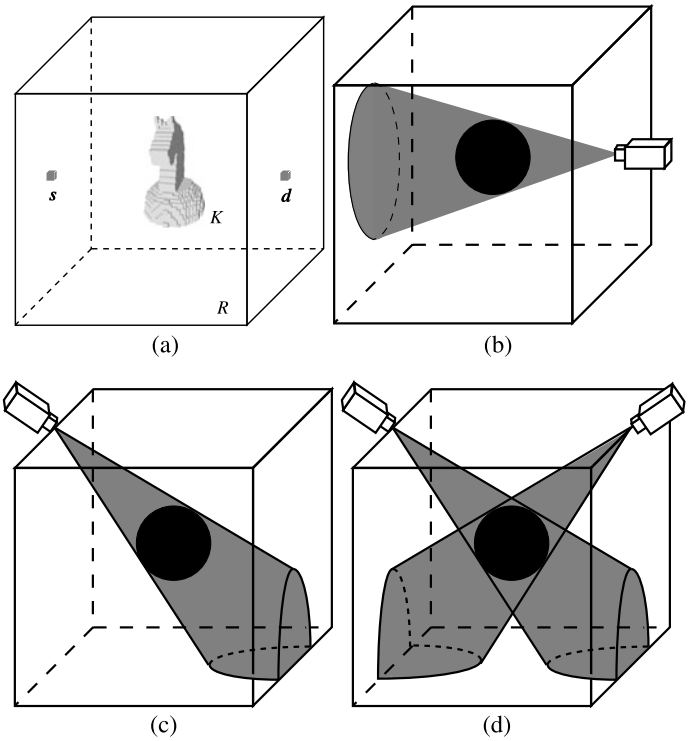


Fig. 11.6 Silhouette and reconstruction of an object in the space. (**a**) Geometry of the detectors and the source for the spatial problem. (**b**) Silhouette of a convex object in the space. (**c**) Visible hull of a silhouette. (**d**) Visible hull of silhouettes

Next, we define a class of non-convex objects.

Definition 1 For any point on the boundary, if there exists at least one unique convex slice-curve which contains this point, we call this object a *slice-convex object*.

A convex closed object is slice-convex. This geometric property of Definition 1 and Theorem 1 lead to the following Theorem 2.

Theorem 2 *A slice-convex object is uniquely reconstructible from the set of silhouettes observed from vertices which lie on the whole sphere encircling the object.*

Theorem 2 permits the reconstruction of a class of non-convex objects from silhouettes. Furthermore, in this expression, the axis of the reconstruction is not necessarily a straight line. The theorem also implies that a series of silhouettes measured using a camera moving along a circle on a plane is insufficient for the full reconstruction of the visible hull of an object. Although this type of measuring system is sometimes used in computer vision, our results show that we cannot reconstruct

the full profile of an object using such a camera system even if the object is convex. Moreover, this theorem allows to fully reconstruct a convex object from a series of silhouettes measured using a general stereo system since a pair of camera orbits defines a pairs of general stereo.

We have the following theorem for a slice-convex object $\boldsymbol{V}$ with respect to axis $\lambda \boldsymbol{v}_0$ for $|\boldsymbol{v}_0| = 1$ and $\lambda \neq 0$, setting $\boldsymbol{A}[\boldsymbol{v}]$ to be the reconstructed object with respect to the axis $\lambda \boldsymbol{v}$, for $\lambda \neq 0$.

Theorem 3 *For an object* $\boldsymbol{V}$, *the relation*

$$\boldsymbol{V} = \bigcap_{\boldsymbol{v} \in \mathbb{S}^2} \boldsymbol{A}[\boldsymbol{v}] \tag{11.4}$$

is satisfied if $\boldsymbol{V}$ *is slice-convex with respect to axis* $\lambda \boldsymbol{v}_0$.

If an object is defined as the common region of a finite number of slice-convex objects, that is, an object $\boldsymbol{V}$ is expressed as

$$\boldsymbol{V} = \bigcap_{\alpha=1}^{n} \boldsymbol{A}[\boldsymbol{a}_\alpha], \quad |\boldsymbol{a}_\alpha| = 1, \tag{11.5}$$

for $\lambda \neq 0$, where $\lambda \boldsymbol{a}_\alpha$ is the axis with respect to which the slices of an object are convex, we have the relation

$$\boldsymbol{V} = \bigcap_{\alpha=1}^{n} \boldsymbol{A}[\boldsymbol{a}_\alpha] \supseteq \bigcap_{\boldsymbol{v} \in \mathbb{S}^2} \boldsymbol{A}[\lambda \boldsymbol{v}] \supseteq \boldsymbol{V}. \tag{11.6}$$

This relation leads to Theorem 4.

Theorem 4 *A slice-convex object* $\boldsymbol{V}$ *is reconstructed as*

$$\boldsymbol{V} = \bigcap_{\boldsymbol{v} \in S^2} \boldsymbol{A}[\boldsymbol{v}]. \tag{11.7}$$

Theorems 3 and 4 show that it is possible to reconstruct a slice-convex object from silhouettes, even if we do not detect its axes, using the equation

$$\mathbf{O} = \bigcap_{\boldsymbol{s} \in A} l(\boldsymbol{s}, \mathbf{O}), \tag{11.8}$$

where A is a closed convex manifold encircling an object $\mathbf{O}$ and $l(\boldsymbol{s}, \mathbf{O})$ is a line which passes through $\boldsymbol{s}$ and satisfies the property

$$l(\boldsymbol{s}, \mathbf{O}) \cap \mathbf{O} \neq \emptyset. \tag{11.9}$$

Furthermore, if we can pre-detect the axes of a slice-convex object, we can reconstruct a three-dimensional non-convex object. This property shows the difference between the shape from silhouettes in the two-dimensional space and three-dimensional space, since, in three-dimensional space, the set of silhouettes does not allow us the reconstruction of non-convex objects. Figure 11.7 shows the geometrical relations of a silhouette in the space and slice-silhouettes in the space obtained from silhouettes in the three-dimensional space.

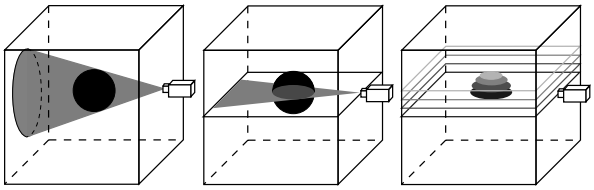


Fig. 11.7 Reconstruction of an object in a three-dimensional space using a two-dimensional method. (**a**) A three-dimensional silhouette of an object. (**b**) A two-dimensional silhouette as a slice of a three-dimensional silhouette. (**c**) A set of two-dimensional slices used to reconstruct a three-dimensional object

11.3 Mathematical Preliminaries

In this section we outline some mathematical facts and notations.

11.3.1 Connectivity of Pixels and Voxels

For $\boldsymbol{p} \in \mathbb{Z}^n$, $n = 2, 3$, $V(\boldsymbol{p}(x, y)) = [x - \frac{1}{2}, x + \frac{1}{2}] \times [y - \frac{1}{2}, y + \frac{1}{2}]$ and $V(\boldsymbol{p}(x, y, z)) = [x - \frac{1}{2}, x + \frac{1}{2}] \times [y - \frac{1}{2}, y + \frac{1}{2}] \times [z - \frac{1}{2}, z + \frac{1}{2}]$ are called a pixel and a voxel, respectively.

Definition 2 For two points $\boldsymbol{p} = (p_1, \ldots, p_n)^\top$ and $\boldsymbol{q} = (q_1, \ldots, q_n)^\top$ in $\mathbb{Z}^n$, if $|p_i - q_i| \leq 1$, $1 \leq i \leq n$ and $k \leq n - \sum_{i=1}^{n} |p_i - q_i| \leq n$, we call points $\boldsymbol{p}$ and $\boldsymbol{q}$ k-connected.

The order of connectivity k expresses the minimum dimension of the common parts of two pixels and voxels for $n = 2, 3$, respectively.

Definition 3 The k-neighborhood $N_k(\boldsymbol{p})$ of a discrete point is the set of all k-connected points. Furthermore, we set $A_k(\boldsymbol{p}) = N_k(\boldsymbol{p}) \setminus \boldsymbol{p}$.

For $n = 2, 3$, we have N_0- and N_1-neighborhoods, and N_0-, N_1-, and N_2-neighborhoods, respectively. The N_0 and N_1 neighborhoods on the discrete plane $\mathbb{Z}^2$ are equivalent to the 8-neighborhood $\mathbf{N}_8$ and 4-neighborhood $\mathbf{N}_4$, respectively, if we define the neighborhoods using the numbers of points connected to each point in $\mathbb{Z}^2$. In the discrete space $\mathbb{Z}^3$, N_0-, N_1-, and N_2-neighborhoods are equivalent to the 26-neighborhood $\mathbf{N}_{26}$, 18-neighborhood $\mathbf{N}_{18}$, and 6-neighborhood $\mathbf{N}_6$, respectively, if we define the neighborhoods using the numbers of connected points to each point in $\mathbb{Z}^3$.

Next, we define the order of a path between two points and a k-connected object.

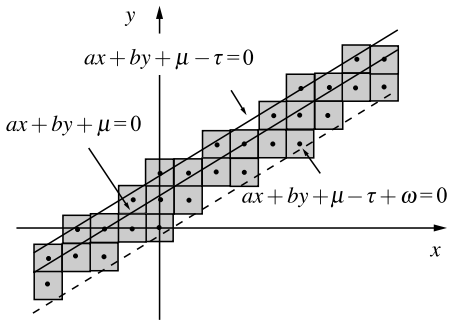


Fig. 11.8 Discrete planar line $L(a, b, \mu)$. Lattice points on a *line* are contained in $L(a, b, \mu)$, although the lattice points on the *dashed line* are not contained in $L(a, b, \mu)$

Definition 4 For a sequence of points $p = [\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n]$, if all pairs of points $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$ are k-connected, we call p a k-path. If there exists k-paths between all pair of points of M, M is called k-connected discrete object.

Using the paths, we define the connectivity of an object in the discrete space.

Definition 5 For a point $\boldsymbol{p}$ in a k-connected object M, if $M \setminus \boldsymbol{p}$ is not k-connected, we call M the minimum k-connected. Furthermore, if M is at least 0-connected, M is connected. If M is not at most 0-connected, then M is disconnected.

11.3.2 Discrete Linear Objects

11.3.2.1 Discrete Planar Lines

Definition 6 Setting $L(\boldsymbol{s}, \boldsymbol{d})$ to be a line on the Euclidean plane $\mathbb{R}^2$, the discrete line associated with the line $L(a, b, \mu)$ is the set of lattice points on $\mathbb{Z}^2$ which satisfy the double inequality

$$L(a, b, \mu) = \left\{(x, y) \in \mathbb{Z}^2 \middle| \tau \leq ax + by + \mu < \tau + \omega\right\}, \tag{11.10}$$

for $a > 0$, or $a = 0$ and $b > 0$, as shown in Fig. 11.8, where $a, b, \mu \in \mathbb{Z}$, $\omega \in \mathbb{N}$, and $\tau \in \mathbb{Q}$ for the sets of natural and rational numbers. Here ω defines the thickness of the line, and μ and τ define the intercept of the line.

By selecting τ and ω, we have three types of discrete lines on the plane.

Definition 7 The supercover of the line $L(\boldsymbol{s}, \boldsymbol{d})$ is the set of pixels which intersect the line $L(\boldsymbol{s}, \boldsymbol{d})$, that is,

$$\left\{p \in \mathbb{Z}^2 \middle| V(p) \cap L(\boldsymbol{s}, \boldsymbol{d}) \neq \emptyset\right\}. \tag{11.11}$$

Definition 8 The standard line of $L(\boldsymbol{s}, \boldsymbol{d})$ is the set of the 1-connected pixels which intersect the line.

Definition 9 The naive line of $L(\boldsymbol{s}, \boldsymbol{d})$ is the set of the 0-connected pixels which intersect the line.

Definitions 7, 8, and 9 give algebraic expressions of discrete lines in the plane.

Theorem 5 *The supercover of $L(\boldsymbol{s}, \boldsymbol{d})$ on $\mathbb{R}^2$ is the set of the solutions of the double Diophantine inequality*

$$\left\{(x, y)^\top \in \mathbb{Z}^2 \middle| -\frac{|a| + |b|}{2} \leq ax + by + \mu \leq \frac{|a| + |b|}{2}\right\} \tag{11.12}$$

as shown in Fig. 11.9(a).

Theorem 6 *The standard line of $L(\boldsymbol{s}, \boldsymbol{d})$ is the set of the solutions of the double Diophantine inequality*

$$\left\{(x, y)^\top \in \mathbb{Z}^2 \middle| -\frac{|a| + |b|}{2} \leq ax + by + \mu < \frac{|a| + |b|}{2}\right\} \tag{11.13}$$

for $a > 0$, or $a = 0$ and $b > 0$, as shown in Fig. 11.9(b).

Theorem 7 *The naive line of $L(\boldsymbol{s}, \boldsymbol{d})$ is the set of the solutions of the double Diophantine inequality*

$$\left\{(x, y)^\top \in \mathbb{Z}^2 \middle| -\frac{\max(|a|, |b|)}{2} \leq ax + by + \mu < \frac{\max(|a|, |b|)}{2}\right\} \tag{11.14}$$

for $a > 0$, or $a = 0$ and $b > 0$, as shown in Fig. 11.9(c).

By removing equality of the right hand side of the equation of the supercover, the definition of the standard line allows us to reduce the local thickness of a discrete line.

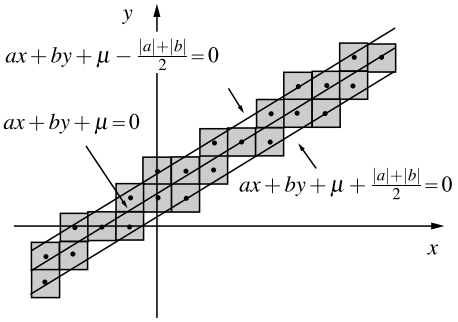
Theorem 8 *If the line $L(a, b, \mu)$, $(a \neq 0, b \neq 0) \in \mathbb{R}^2$ passes through the point $(m + \frac{1}{2}, n + \frac{1}{2})$, $m, n \in \mathbb{Z}$ the supercover contains all four pixels whose centers are $(m, n)^\top$, $(m + 1, n)^\top$, $(m, n + 1)^\top$, and $(m + 1, n + 1)^\top$ and the standard line contains the three pixels whose centers are $(m, n)^\top$, $(m, n + 1)^\top$, and $(m + 1, n \vee (n + 1))^\top$.*

11.3.2.2 Discrete Spatial Lines

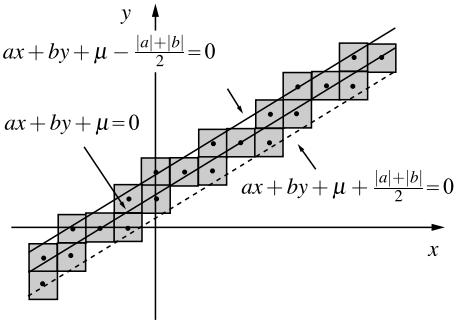
Definition 10 The supercover of the line $L(\boldsymbol{s}, \boldsymbol{d})$ is the set of voxels which intersect the line, that is,

$$\left\{p \in \mathbb{Z}^3 \middle| V(p) \cap L(\boldsymbol{s}, \boldsymbol{d}) \neq \emptyset\right\}. \tag{11.15}$$

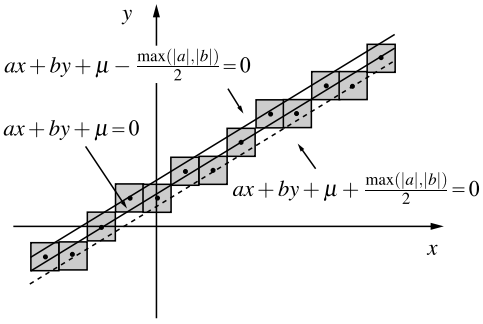
Definition 11 The standard line of $L(\boldsymbol{s}, \boldsymbol{d})$ is the set of the 2-connected voxels which intersect the line.



(a) The supercover.



(b) The standard line.



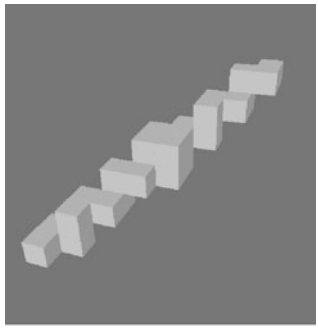
(c) The naive line.

Fig. 11.9 The supercover, the standard line, and the naive line $L(a, b, \mu)$ on $\mathbb{Z}^2$. The lattice points on a *line* are contained in $L(a, b, \mu)$, although the lattice points on the *dashed line* are not contained in $L(a, b, \mu)$

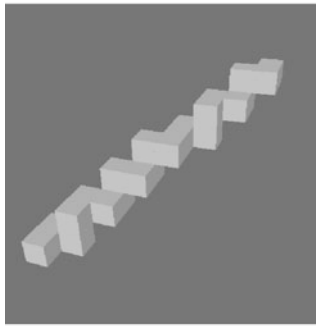
Definition 12 The naive line of $L(\boldsymbol{s}, \boldsymbol{d})$ is the set of the 0-connected pixels which intersect the line.

Figure 11.10 shows the three types of discrete lines in the space.

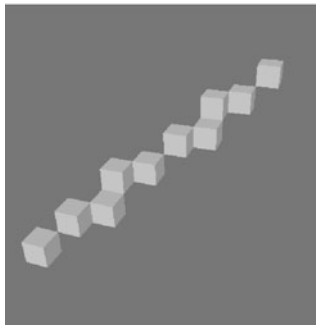
Definitions 10, 11, and 12 give algebraic expressions of discrete lines in the space.



(a) The supercover.



(b) The standard line.



(c) The naive line.

Fig. 11.10 The three types of discrete spatial lines of $L(s = (0, 0, 0)^\top, \boldsymbol{d} = (5, 7, 9)^\top)$

Theorem 9 *The supercover of the line $L(\boldsymbol{s}, \boldsymbol{d})$ is the set of the solutions of the double Diophantine inequality*

$$\begin{cases} -\frac{1}{2}\big(|a| + |b|\big) \leq ax + bz + \mu_1 \leq \frac{1}{2}\big(|a| + |b|\big), \\ -\frac{1}{2}\big(|a| + |c|\big) \leq ay + cz + \mu_2 \leq \frac{1}{2}\big(|a| + |c|\big), \\ -\frac{1}{2}\big(|b| + |c|\big) \leq cx - by + \mu_3 \leq \frac{1}{2}\big(|b| + |c|\big). \end{cases} \tag{11.16}$$

Theorem 10 *The standard line of $L(s, \boldsymbol{d})$ in $\mathbb{Z}^3$ is defined as follows.*

If for $a, b,$ *and* c, $a = b = 0$, $a = c = 0$, $b = c = 0$ *none of the following equalities hold*:

$$\begin{cases} -\frac{1}{2}\big(|a| + |b|\big) \le ax + bz + \mu_1 < \frac{1}{2}\big(|a| + |b|\big) \\ \quad (a > 0, \ or\, a = 0, \ b > 0) \\ -\frac{1}{2}\big(|a| + |c|\big) \le ay + cz + \mu_2 < \frac{1}{2}\big(|a| + |c|\big) \\ \quad (a > 0, \ or\, a = 0, \ c > 0) \\ -\frac{1}{2}\big(|b| + |c|\big) \le cx - by + \mu_3 < \frac{1}{2}\big(|b| + |c|\big) \\ \quad (c > 0, \ or\, c = 0, \ -b > 0); \end{cases} \tag{11.17}$$

If $a = b = 0$:

$$\begin{cases} -\frac{1}{2}|c| \le cz + \mu_2 < \frac{1}{2}|c| & (c > 0) \\ -\frac{1}{2}|c| \le cx + \mu_3 < \frac{1}{2}|c| & (c < 0); \end{cases} \tag{11.18}$$

If $a = c = 0$:

$$\begin{cases} -\frac{1}{2}|b| \le bz + \mu_1 < \frac{1}{2}|b| & (b > 0) \\ -\frac{1}{2}|b| \le -by + \mu_3 < \frac{1}{2}|b| & (b < 0); \end{cases} \tag{11.19}$$

If $b = c = 0$:

$$\begin{cases} -\frac{1}{2}|a| \le ax + \mu_1 < \frac{1}{2}|a| & (a > 0) \\ -\frac{1}{2}|a| \le ay + \mu_2 < \frac{1}{2}|a| & (a < 0). \end{cases} \tag{11.20}$$

Theorem 11 *The naive line of* $L(s, \boldsymbol{d})$ *in* $\mathbb{Z}^3$ *is defined as follows.*
If $\max(|a|, |b|, |c|) = |a|$:

$$\begin{cases} -\frac{1}{2}|a| \le ax + bz + \mu_1 < \frac{1}{2}|a| & (a > 0) \\ -\frac{1}{2}|a| \le ay + cz + \mu_2 < \frac{1}{2}|a| & (a < 0); \end{cases} \tag{11.21}$$

If $\max(|a|, |b|, |c|) = |b|$:

$$\begin{cases} -\frac{1}{2}|b| \le ax + bz + \mu_1 < \frac{1}{2}|b| & (b > 0) \\ -\frac{1}{2}|b| \le cx - by + \mu_3 < \frac{1}{2}|b| & (b < 0); \end{cases} \tag{11.22}$$

If $\max(|a|, |b|, |c|) = |c|$:

$$\begin{cases} -\frac{1}{2}|c| \le ay + cz + \mu_2 < \frac{1}{2}|c| & (c > 0) \\ -\frac{1}{2}|c| \le cx - by + \mu_3 < \frac{1}{2}|c| & (c < 0). \end{cases} \tag{11.23}$$

Figures 11.10(a), 11.10(b), and 11.10(c) show the three types of discrete lines in the space.

11.3.2.3 Discrete Planes

Setting $\boldsymbol{a} = (a, b, c)^\top \in \mathbb{Z}^3$ and $\mu \in \mathbb{Z}$, discrete planes are considered as an extension in $\mathbb{Z}^3$ of planar discrete lines.

Theorem 12 *The supercover of $P(\boldsymbol{a}, \mu)$ on $\mathbb{R}^3$ is the set of solutions of the double Diophantine inequalities*

$$\left\{(x, y, z)^\top \in \mathbb{Z}^3 \middle| -\frac{|a| + |b| + |c|}{2} \leq ax + by + cz + \mu \leq \frac{|a| + |b| + |c|}{2}\right\}. \tag{11.24}$$

Theorem 13 *The standard plane of $P(\boldsymbol{a}, \mu)$ is the set of the solutions of the double Diophantine inequality*

$$\left\{(x, y, z)^\top \in \mathbb{Z}^3 \middle| -\frac{|a| + |b| + |c|}{2} \leq ax + by + cz + \mu < \frac{|a| + |b| + |c|}{2}\right\}, \tag{11.25}$$

where $a > 0$ or ($a = 0$ and $b > 0$) or ($a = 0$, $b = 0$, and $c > 0$).

Theorem 14 *The naive plane corresponding to $P(\boldsymbol{a}, \mu)$ is the set of the solutions of the double Diophantine inequality*

$$\left\{(x, y, z)^\top \in \mathbb{Z}^3 \middle| -\frac{\max(|a|, |b|, |c|)}{2} \leq ax + by + cz + \mu < \frac{\max(|a|, |b|, |c|)}{2}\right\}, \tag{11.26}$$

where ($a > 0$) or ($a = 0$ and $b > 0$) or ($a = 0$, $b = 0$ and $c > 0$).

11.3.3 Discrete Polygons and Polyhedra

In this subsection, we define a polytope in the discrete space $\mathbb{Z}^n$ using discrete hyperplanes. For the plane

$$P(\boldsymbol{a}, \mu) = \left\{x_i \in \mathbb{Z} \middle| \sum_{i=1}^{n-1} a_i x_i + \mu = 0\right\}, \tag{11.27}$$

where $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)^\top \in \mathbb{Z}^n$, and $\mu \in \mathbb{Z}$, in $\mathbb{Z}^n$, we define a pair of half-spaces as

$$H^+\big(P(\boldsymbol{a}, \mu)\big) = \left\{\boldsymbol{x} \in \mathbb{Z}^n \middle| \sum_{i=1}^{n-1} a_i x_i + \mu \geq 0\right\}, \tag{11.28}$$

$$H^-\big(P(\boldsymbol{a}, \mu)\big) = \left\{\boldsymbol{x} \in \mathbb{Z}^n \middle| \sum_{i=1}^{n-1} a_i x_i + \mu \leq 0\right\}, \tag{11.29}$$

and

$$\bar{H}^{+}\big(P(\boldsymbol{a},\mu)\big)=\left\{\boldsymbol{x}\in\mathbb{Z}^{n}\left|\sum_{i=1}^{n-1}a_{i}x_{i}+\mu>0\right.\right\},\tag{11.30}$$

$$\bar{H}^{-}\big(P(\boldsymbol{a},\mu)\big)=\left\{\boldsymbol{x}\in\mathbb{Z}^{n}\left|\sum_{i=1}^{n-1}a_{i}x_{i}+\mu<0\right.\right\}.\tag{11.31}$$

We define a convex polyhedron in the discrete space $\mathbb{Z}^n$ as

$$M=\bigcap_{j=1}^{m}H^{-}\big(P(\boldsymbol{a}_{j},\mu_{j})\big),\tag{11.32}$$

which is equivalent to

$$M=\big\{\boldsymbol{x}\in\mathbb{Z}^{n}\big|\boldsymbol{x}\in M\oplus N_{0}\big\}.\tag{11.33}$$

For the extension $\oplus$ of points in $\mathbb{Z}^n$ with N_k, we have the following relations.

$$H^{+}\oplus N_{0}\big(P(\boldsymbol{a},\mu)\big)=\left\{\boldsymbol{x}\in\mathbb{Z}^{n}\left|\sum_{i=1}^{n-1}a_{i}x_{i}+\mu+\sum_{i=1}^{n-1}|a_{i}|\geq 0\right.\right\}\tag{11.34}$$

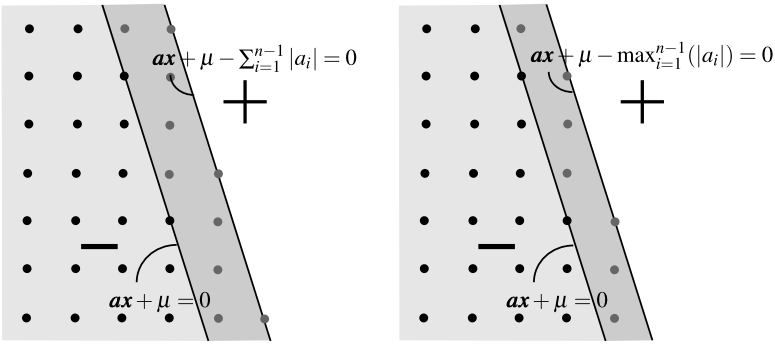
$$H^{-}\oplus N_{0}\big(P(\boldsymbol{a},\mu)\big)=\left\{\boldsymbol{x}\in\mathbb{Z}^{n}\left|\sum_{i=1}^{n-1}a_{i}x_{i}+\mu-\sum_{i=1}^{n-1}|a_{i}|\leq 0\right.\right\}\tag{11.35}$$

and

$$H^{+}\oplus N_{n-1}\big(P(\boldsymbol{a},\mu)\big)=\left\{\boldsymbol{x}\in\mathbb{Z}^{n}\left|\sum_{i=1}^{n-1}a_{i}x_{i}+\mu+\max_{i=1}^{n-1}\big(|a_{i}|\big)\geq 0\right.\right\}\tag{11.36}$$

$$H^{-}\oplus N_{n-1}\big(P(\boldsymbol{a},\mu)\big)=\left\{\boldsymbol{x}\in\mathbb{Z}^{n}\left|\sum_{i=1}^{n-1}a_{i}x_{i}+\mu-\max_{i=1}^{n-1}\big(|a_{i}|\big)\leq 0\right.\right\}.\tag{11.37}$$

Figures 11.11(a) and 11.11(b) show the extension of H^- by N_0, N_{n-1} on the two-dimensional discrete plane.



(a) $H^- \oplus N_0$. (b) $H^- \oplus N_{n-1}$.

Fig. 11.11 Extension of H^- by N_0, N_{n-1} on the two-dimensional discrete plane

Since

$$M \oplus N_0 = \bigcap_{j=1}^{m} H^-\left(L\left(\boldsymbol{a}_j, \mu_j + \sum_{i=1}^{n-1}|a_{j,i}|\right)\right) \tag{11.38}$$

$$M \oplus N_{n-1} \subseteq \bigcap_{j=1}^{m} H^-\left(L\left(\boldsymbol{a}_j, \mu_j + \max_{i=1}^{n-1}(|a_{j,i}|)\right)\right), \tag{11.39}$$

where $\boldsymbol{a}_j = \{\boldsymbol{a}_{j,1}, \ldots, \boldsymbol{a}_{j,n-1}\}$, we have the relations such that

$$M \oplus \bigoplus_{h=1}^{k} N_0 = \bigcap_{j=1}^{m} H^-\left(L\left(\boldsymbol{a}_j, \mu_j + k\sum_{i=1}^{n-1}|a_{j,i}|\right)\right) \tag{11.40}$$

$$M \oplus \bigoplus_{h=1}^{k} N_{n-1} \subseteq \bigcap_{j=1}^{m} H^-\left(L\left(\boldsymbol{a}_j, \mu_j + k\max_{i=1}^{n-1}(|a_{j,i}|)\right)\right), \tag{11.41}$$

where $k \in \mathbb{Z}, k > 0$.

For a discrete convex polytope M, since $P \oplus N_0 \setminus P$ is the kernel of $P \oplus N_0$, the kernel of $P \oplus \bigoplus_{h=1}^{k} N_0$ is

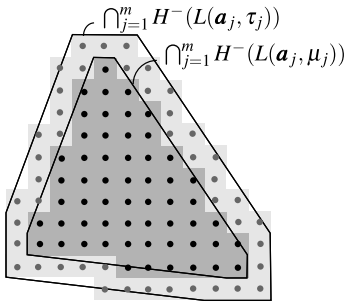
$$U_k = \left(P \oplus \bigoplus_{h=1}^{k} N_0\right) \setminus \left(P \oplus \bigoplus_{h=1}^{k-1} N_0\right), \tag{11.42}$$

where $k > 0, k \in \mathbb{Z}$. Figures 11.12(a), 11.12(b), and 11.12(c) show $P \oplus N_0$, $P \oplus N_{n-1}$, and U_1 on $\mathbb{Z}^n$ for $n = 2$, respectively.

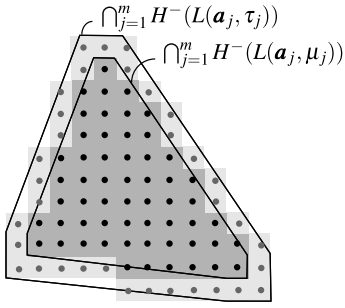
As an analytical expression of $(P \oplus \bigoplus_{h=1}^{k} N_0)$, we have next Lemma 1.

Lemma 1 *Setting*

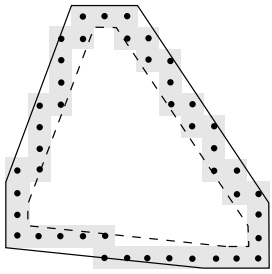
$$H_1 = \bigcap_{j=1}^{m} H^-\left(L\left(a_j, b_j, \mu_j + k\left(|a_j| + |b_j|\right)\right)\right), \tag{11.43}$$



(a) $P \oplus N_0$.



(b) $P \oplus N_{n-1}$.



(c) U_1.

Fig. 11.12 Extension of a discrete polygon by N_0 and N_1 on the discrete plane. In (**a**) and (**b**), $\tau_j = \mu_j + \sum_{i=1}^{n-1} |a_{j,i}|$ and $\tau_j = \mu_j + \max_{i=1}^{n-1}(|a_{j,i}|)$, respectively

$$H_2 = \bigcup_{j=1}^{m} \bar{H}_{\mathbb{Z}}^{-}\big(L\big(a_j, b_j, \mu_j + (k-1)\big(|a_j| + |b_j|\big)\big)\big), \tag{11.44}$$

for U_k defined by Eq. (11.42)*, the relation*

$$U_k = H_1 \cap H_2 \tag{11.45}$$

is satisfied.

11.4 Shape Reconstruction in Discrete Space

In this section we explain and illustrate the reconstruction process in 3D space.

11.4.1 Reconstruction of Space and Object

In $\mathbb{Z}^n$, instead of the pair of the source $\boldsymbol{s}$ and $\Omega(\boldsymbol{s})$ for a line, we detect the pair of $\boldsymbol{s}$ and $\boldsymbol{d}$. Furthermore, in $\mathbb{Z}^2$, we assume that detector pixels lie on the edges E of the square D^2 whose four vertices are $(0,0)^\top$, $(3n-1,0)^\top$, $(3n-1,3n-1)^\top$, and $(0,3n)^\top$, and that a discrete object exists in the square R whose vertices are $(n-1,n-1)^\top$, $(2n-1,n)^\top$, $(2n-1,2n-1)^\top$, and $(n-1,2n-1)^\top$. We assume that our object is a 4-connected simple object. Moreover, the source pixel moves on the edges of the square D. Therefore, we can detect a set of discrete half-planes which are divided by line segments connecting two pixels on D, namely, the source $\boldsymbol{s}=(s_1,s_2)^\top$ and the detector $\boldsymbol{d}=(d_1,d_2)^\top$. Furthermore, in $\mathbb{Z}^3$, we assume that the detectors are voxels on a cube D^3 whose vertices are $(0,0,0)^\top$, $(3n-1,0,0)^\top$, $(3n-1,3n-1,0)^\top$, $(0,3n-1,0)^\top$, $(0,0,3n)^\top$, $(3n-1,0,3n-1)^\top$, $(3n-1,3n-1,3n-1)^\top$, and $(0,3n-1,3n-1)^\top$, and the object is enclosed in a cubic region R whose vertices are $(n-1,n-1,n-1)^\top$, $(n-1,2n-1,n-1)^\top$, $(2n-1,2n-1,n-1)^\top$, $(n-1,2n-1,n-1)^\top$, $(n-1,n-1,2n-1)^\top$, $(n-1,2n-1,2n-1)^\top$, $(2n-1,2n-1,2n-1)^\top$, and $(n-1,2n-1,2n-1)^\top$. The source $\boldsymbol{s}$ moves on the faces F of D^3. We assume that our object is 6-connected simple object.

Setting $\boldsymbol{l}(\boldsymbol{s},\mathbf{O})$ to be a line passing through a fixed source $\boldsymbol{s}$, we define a set of voxels $\boldsymbol{d}$ on D^3 such that

$$\boldsymbol{l}(\boldsymbol{s},\mathbf{O})\cap\mathbf{O}=\emptyset \tag{11.46}$$

for an object $\mathbf{O}$. The set of voxels $\boldsymbol{d}$ is the silhouette of object $\mathbf{O}$ with respect to the source $\boldsymbol{s}$.

Figure 11.13 shows discrete models obtained through *Shape from Silhouettes* in the plane and space, respectively.

11.4.2 Convex Hull and Visual Hull in Discrete Space

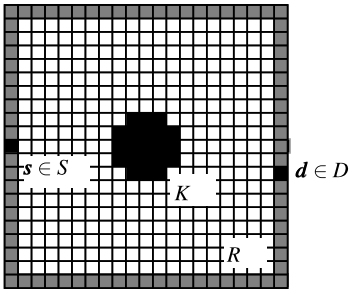
Definition 13 For a source $\boldsymbol{s}$ and $\boldsymbol{t}\in D$, where $D:=D^n$ for $n=2,3$, the silhouette from the source $\boldsymbol{s}$ is

$$T(\boldsymbol{s},D,K)=\left\{\boldsymbol{t}\middle|L(\boldsymbol{s},\boldsymbol{t})\cap K\neq\emptyset,\boldsymbol{t}\in D\right\}. \tag{11.47}$$

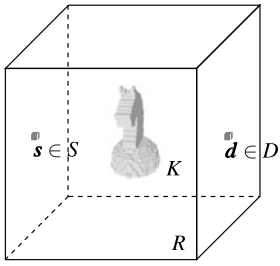
Definition 14 $\partial T(\boldsymbol{s},D,K)$ such that

$$\partial T_*(\boldsymbol{s},D,K)=\left\{\boldsymbol{t}\middle|A\not\subseteq T(\boldsymbol{s},D,K),A=N_0(\boldsymbol{t})\cap D,\boldsymbol{t}\in T(\boldsymbol{s},D,K)\right\} \tag{11.48}$$

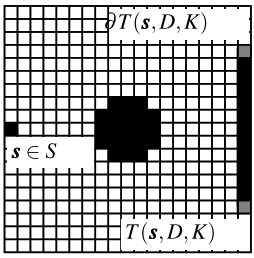
is the boundary of the silhouette $T(\boldsymbol{s},D,K)$.



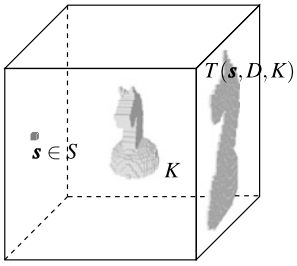
(a) Object in 2D discrete space.



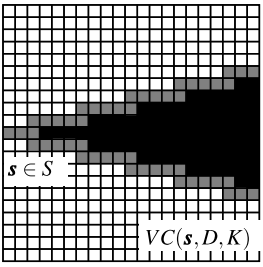
(b) Object in 3D discrete space.



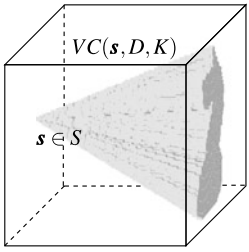
(c) Silhouette in 2D discrete space.



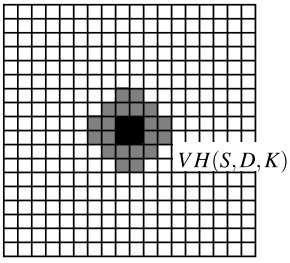
(d) Silhouette in 3D discrete space.



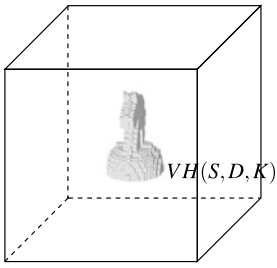
(e) Visual cone in 2D discrete space.



(f) Visual cone in 3D discrete space.



(g) Visual hull in 2D discrete space.



(h) Visual hull in 3D discrete space.

Fig. 11.13 Reconstruction in discrete space. Form *top* to *down*, environment, silhouette, visual cone, and visual hull for the two-dimensional discrete space (*left*) and the three-dimensional discrete space are shown (*right*)

Algorithm 1: GENERATION_OF_VISUAL_HULL $VH(S, D, K)$

Input: $temp(\emptyset, D, K) = R$
Input: $T(\boldsymbol{s}_j, D, K)$, $1 \leq j \leq |S|$
Result: $VH(S, D, K)$
for $j \leftarrow 1$ **to** $|S|$ **do**
  compute $VC(\boldsymbol{s}_j, D, K)$;
  $temp(\bigcup_{i=1}^{j}\{\boldsymbol{s}_i\}, D, K) = temp(\bigcup_{i=1}^{j-1}\{\boldsymbol{s}_i\}, D, K) \cap VC(\boldsymbol{s}_j, D, K)$;
  $VH(S, D, K) = temp(\bigcup_{i=1}^{|S|}\{\boldsymbol{s}_i\}, D, K)$;

Lemma 2 *Let $T_{Sup}(\boldsymbol{s}, D, K)$, $T_{Sta}(\boldsymbol{s}, D, K)$, and $T_{Nai}(\boldsymbol{s}, D, K)$ be the silhouettes defined by the supercover, standard, and naive lines, respectively. We have the relation*

$$T_{Nai}(\boldsymbol{s}, D, K) \subseteq T_{Sta}(\boldsymbol{s}, D, K) \subseteq T_{Sup}(\boldsymbol{s}, D, K). \tag{11.49}$$

We define the labels for discrete points as

$$L(\mathbf{p}, \boldsymbol{s}, D, K) = \begin{cases} 1, & (\text{if } \mathbf{p} \in L(\boldsymbol{s}, \boldsymbol{t}) \text{ and } \mathbf{p} \notin L(\boldsymbol{s}, \hat{\boldsymbol{t}})), \\ 2, & (\text{if } \mathbf{p} \in L(\boldsymbol{s}, \hat{\boldsymbol{t}})), \\ 3, & (\text{otherwise}). \end{cases} \tag{11.50}$$

Using these labels, we define the visual cone and visual hull.

Definition 15 $VC(\boldsymbol{s}, D, K)$ and $\partial VC(\boldsymbol{s}, D, K)$, such that

$$VC(\boldsymbol{s}, D, K) = \left\{ \boldsymbol{p} \middle| L(\boldsymbol{p}, \boldsymbol{s}, D, K) = 1, 2 \right\} \tag{11.51}$$

$$\partial VC(\boldsymbol{s}, D, K) = \left\{ \boldsymbol{p} \middle| L(\boldsymbol{p}, \boldsymbol{s}, D, K) = 2 \right\}, \tag{11.52}$$

are denoted as the visual cone and the boundary of the visual cone, respectively.

For the visual cone, we have the relation

$$VC(\boldsymbol{s}, D, K) = \bigcup_{\boldsymbol{t} \in T(\boldsymbol{s}, D, K)} L(\boldsymbol{s}, \boldsymbol{t}). \tag{11.53}$$

Algorithm 1 shows an algorithm for the generation of visual hull $VC(\boldsymbol{s}, D, K)$ from $T(\boldsymbol{s}, D, K)$ using the geometric property of Eqs. (11.51) and (11.53).

Lemma 3 *For the visual cones $VC_{Sup}(\boldsymbol{s}, D, K)$, $VC_{Sta}(\boldsymbol{s}, D, K)$, and $VC_{Nai}(\boldsymbol{s}, D, K)$ defined by the supercover, standard, and naive lines, respectively, we have the relation*

$$VC_{Nai}(\boldsymbol{s}, D, K) \subseteq VC_{Sta}(\boldsymbol{s}, D, K) \subseteq VC_{Sup}(\boldsymbol{s}, D, K). \tag{11.54}$$

Lemma 4 *A visual cone $VC(\boldsymbol{s}, D, K)$ satisfies the relation*

$$K \subset VC(\boldsymbol{s}, D, K). \tag{11.55}$$

Lemmas 3 and 4 are obvious from the geometrical properties of the definitions of discrete lines.

Definition 16 $VH(\boldsymbol{s}, D, K)$ and $\partial VH(\boldsymbol{s}, D, K)$, such that

$$VH(S, D, K) = \{\boldsymbol{p} | L(\mathbf{p}, S, D, K) = 1, 2\}, \tag{11.56}$$

$$\partial VH(S, D, K) = \{\boldsymbol{p} | L(\mathbf{p}, S, D, K) = 2\}, \tag{11.57}$$

are called visual hull and boundary of the visual hull, respectively.

Furthermore, we set

$$VH(S, D, K) = \bigcap_{\boldsymbol{s} \in S} VC(\boldsymbol{s}, D, K). \tag{11.58}$$

Lemma 5 *For the visual hulls* $VC_{Sup}(\boldsymbol{s}, D, K)$, $VC_{Sta}(\boldsymbol{s}, D, K)$, *and* $VC_{Nai}(\boldsymbol{s}, D, K)$ *defined by supercover, standard, and naive lines, we have the relation*

$$VH_{Nai}(S, D, K) \subseteq VH_{Sta}(S, D, K) \subseteq VH_{Sup}(S, D, K). \tag{11.59}$$

Lemma 6 $VH(S, D, K)$ *satisfies the relation*

$$K \subseteq VH(S, D, K). \tag{11.60}$$

These two propositions are obvious from the geometrical properties of the definitions of discrete lines. From Lemmas 5 and 6 the next Lemma 7 is derived.

Lemma 7 *For a pair of discrete objects* K_1 *and* K_2, *if* $K_1 \subseteq K_2$, *their visual hulls satisfy the relation*,

$$VH(S, D, K_1) \subseteq VH(S, D, K_2). \tag{11.61}$$

Proof For a source $\boldsymbol{s}$, $VC(\boldsymbol{s}, D, K)$ is the set of all discrete lines $L(\boldsymbol{s}, \boldsymbol{d})$ which intersect K. Therefore, the relation $K_1 \subseteq K_2$ implies the relation $VC(\boldsymbol{s}, D, K_1) \subseteq VC(\boldsymbol{s}, D, K_2)$. □

Furthermore, Lemma 7 implies Theorem 15.

Theorem 15 *For sets of sources* S_1, S_2, *if* $S_1 \subseteq S_2$ *the visual hulls satisfy the relation*

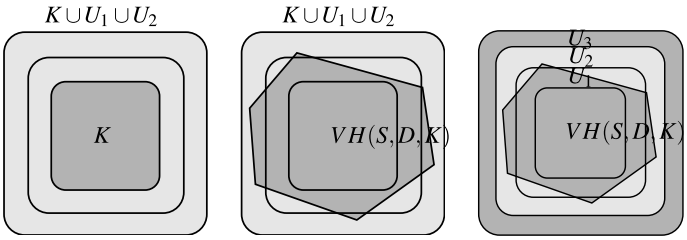
$$VH(S_1, D, K) \subseteq VH(S_2, D, K). \tag{11.62}$$

Therefore, for the reconstructed object, we have Theorem 16.

Theorem 16 *The relation*

$$VH(S, D, K) \subseteq (K \oplus N_0 \oplus N_0) \tag{11.63}$$

is satisfied for all types of discrete lines in two- and in three-dimensional discrete space.



(a) $K \cup U_1 \cup U_2$ in gray and K in dark gray.

(b) $K \cup U_1 \cup U_2$ in gray and $VH(S,D,K)$in dark gray.

(c) Illustration proof of $U_3 \cap VH(S,D,K) = \emptyset$.

Fig. 11.14 Illustration of upper bound of reconstruction. $VH(S, D, K)$ has a distance at most 2 from K with 0-connectivity

Theorem 16 can be read as the next Theorem 17.

Theorem 17 *If K is the discretization of a finite convex region in $\mathbb{R}^2$, $\hat{K}$ satisfies the relation*

$$\hat{K} \setminus K \subset (K \oplus \mathbf{N}_8 \oplus \mathbf{N}_8) \setminus K, \tag{11.64}$$

where $\mathbf{N}_8$ is the 8-neighborhood of the origin.

For these voxels, we have Theorem 18.

Theorem 18 *If K is the discretization of a finite convex region in $\mathbb{R}^3$, $\hat{K}$ satisfies the relation*

$$\hat{K} \setminus K \subset (K \oplus \mathbf{N}_{26} \oplus \mathbf{N}_{26}) \setminus K, \tag{11.65}$$

where $\mathbf{N}_{26}$ is the 26-neighborhood of the origin.

Figure 11.14 illustrates the upper bound of reconstruction. In Fig. 11.14, $VH(S, D, K)$ has distance at most 2 from K with 0-connectivity.

11.4.3 Proof of Theorem 16

Form the relation of Eq. (11.49), we prove the relation of Eq. (11.63) for an object reconstructed using the supercover.

Using Eq. (11.42), Eq. (11.63) becomes $VH(S, D, K) \subseteq K \cup U_1 \cup U_2$. This relation implies that none of the elements of U_3 is are included in the visual hull, that is,

$$\forall \boldsymbol{p} \in U_3, \quad \boldsymbol{p} \notin VH(S, D, K). \tag{11.66}$$

The configurations of points of a two-dimensional discrete object are illustrated in Fig. 11.14(c). Therefore, the proof is obtained by clarifying the condition that satisfies the relation $\boldsymbol{p} \notin VH(S, D, K)$.

Lemma 8 *Iff a point* $\boldsymbol{p}$ *satisfies the relation* $\boldsymbol{p} \notin VH(S, D, K)$ *for a source* $\boldsymbol{s}$, *a line which intersects* $\boldsymbol{p}$ *does not intersect* K, *that is*,

$$\boldsymbol{p} \notin VH(\boldsymbol{s}, D, K) \iff \exists \boldsymbol{s} \in S, \quad VC(\boldsymbol{s}, D, \boldsymbol{p}) \cap K = \emptyset. \tag{11.67}$$

Proof Equation (11.58) implies that $VH(S, D, K)$ is the set of points which are included in $VC(\boldsymbol{s}, D, K)$ for all $\boldsymbol{s}$. Therefore, for a point $\boldsymbol{s}$, iff $\boldsymbol{p} \notin VH(S, D, K)$, there exists at least $VC(\boldsymbol{s}, D, K)$ which does not include $\boldsymbol{p}$, that is,

$$\boldsymbol{p} \notin VH(S, D, K) \iff \exists \boldsymbol{s} \in S, \quad \boldsymbol{p} \notin VC(\boldsymbol{s}, D, K). \tag{11.68}$$

Furthermore, Eq. (11.53) implies that $L(\boldsymbol{s}, \boldsymbol{d}) \subseteq VC(\boldsymbol{s}, D, K)$ if $L(\boldsymbol{s}, \boldsymbol{d})$ and K intersect. Therefore, if there exists at least one $L(\boldsymbol{s}, \boldsymbol{d})$ which contains $\boldsymbol{p}$ and intersects K, $\boldsymbol{p} \in VC(\boldsymbol{s}, D, K)$ is satisfied. Conversely, the relation $\boldsymbol{p} \notin VC(\boldsymbol{s}, D, K)$ implies that all lines which contains $\boldsymbol{s}$ and $\boldsymbol{p}$ do not intersect K, that is,

$$\boldsymbol{p} \notin VC(\boldsymbol{s}, D, K) \iff VC(\boldsymbol{s}, D, \boldsymbol{p}) \cap K = \emptyset. \tag{11.69}$$

Equations (11.68) and (11.69) imply Eq. (11.67). □

11.4.3.1 Two-Dimensional Case

Figure 11.15 presents an illustration of the proof of the upper bound theorem in case of a two-dimensional reconstructed object.We prove some propositions.

Lemma 9 *For a source* $\boldsymbol{s}$ *and a point* $\boldsymbol{p}$, $VC(\boldsymbol{s}, D, \boldsymbol{p})$ *and* K *have no common point if there exists a line* $L(\boldsymbol{s}, \boldsymbol{d})$ *between the point* $\boldsymbol{p}$ *and the object* K *and it does not intersect either* $\boldsymbol{p}$ *or* K, *that is*,

$$\begin{array}{ll} K \subset H^-\big(L(\boldsymbol{s}, \boldsymbol{d})\big), & K \cap L(\boldsymbol{s}, \boldsymbol{d}) = \emptyset \\ \boldsymbol{p} \in H^+\big(L(\boldsymbol{s}, \boldsymbol{d})\big), & \boldsymbol{p} \notin L(\boldsymbol{s}, \boldsymbol{d}) \end{array} \quad \Rightarrow \quad VC(\boldsymbol{s}, D, \boldsymbol{p}) \cap K = \emptyset \tag{11.70}$$

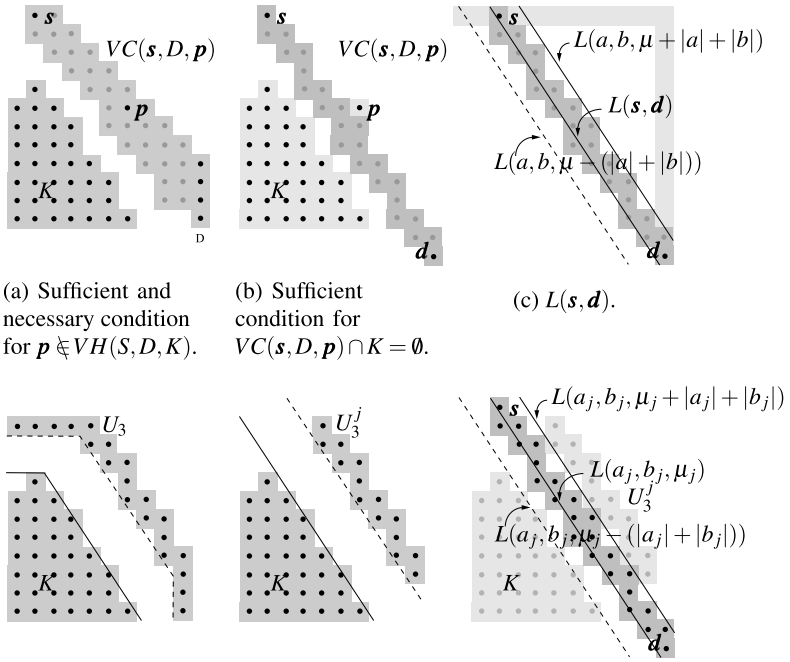
as shown in Fig. 11.15(b).

Proof Equation (11.68) implies that a point $\boldsymbol{d}$ on $L(\boldsymbol{s}, \boldsymbol{d})$ which satisfies Eq. (11.70) is not included in $T(\boldsymbol{s}, D, K)$ and $T(\boldsymbol{s}, D, \boldsymbol{p})$. Since $\boldsymbol{d}$ exists between $T(\boldsymbol{s}, D, K)$ and $T(\boldsymbol{s}, D, \boldsymbol{p})$, $T(\boldsymbol{s}, D, K) \cap T(\boldsymbol{s}, D, \boldsymbol{p}) = \emptyset$. Therefore, $VC(\boldsymbol{s}, D, \boldsymbol{p})$ and K do not intersect. □

Lemma 10 *There exists at least one discrete line* $L(\boldsymbol{s}, \boldsymbol{d})$ *which is contained in* $\bar{H}^-(L(a, b, \mu - (|a| + |b|))) \cap H^-(L(a, b, \mu + |a| + |b|))$, *that is*,

$$\begin{aligned} \exists L(\boldsymbol{s}, \boldsymbol{d}), \quad L(\boldsymbol{s}, \boldsymbol{d}) \subset \bar{H}^-\big(L\big(a, b, \mu - \big(|a| + |b|\big)\big)\big) \\ \cap H^-\big(L\big(a, b, \mu + |a| + |b|\big)\big). \end{aligned} \tag{11.71}$$

This configuration is shown in Fig. 11.15(c).



(a) Sufficient and necessary condition for $\boldsymbol{p} \notin VH(S,D,K)$.

(b) Sufficient condition for $VC(\boldsymbol{s},D,\boldsymbol{p}) \cap K = \emptyset$.

(c) $L(\boldsymbol{s},\boldsymbol{d})$.

(d) The relation between K and U_3.

(e) The relation between K and U_3^j.

(f) $L(\boldsymbol{s},\boldsymbol{d})$ which does not crossU_3^j and K.

Fig. 11.15 Figures illustrating the proof of the upper bound theorem of a two-dimensional reconstructed object

Proof Since $\bar{H}^-(L(a,b,\mu - \frac{|a|+|b|}{2})) \cap H^-(L(a,b,\mu + \frac{|a|+|b|}{2}))$ is equivalent to the width of the standard line, $\bar{H}^-(L(a,b,\mu - \frac{|a|+|b|}{2})) \cap H^-(L(a,b,\mu + \frac{|a|+|b|}{2}))$ is the set of connected points with 4-connectivity. The set $B(R)$ whose source and detector are S and D, respectively is the closed line with 4-connectivity. Therefore, $\bar{H}^-(L(a,b,\mu - \frac{|a|+|b|}{2})) \cap H^-(L(a,b,\mu + \frac{|a|+|b|}{2}))$ and $B(R)$ share a pair of points $\boldsymbol{s}$ and $\boldsymbol{d}$. Then, the width of the supercover implies the relation $\bar{H}^-(L(a,b,\mu - (|a| + |b|))) \cap H^-(L(a,b,\mu + |a| + |b|))$. Equation (11.32) and Lemma 1 imply the relations

$$K = \bigcap_{j=1}^{m} H^-\big(L(a_j,b_j,\mu_j)\big) \tag{11.72}$$

$$U_3 \subset \bigcup_{j=1}^{m} \bar{H}^-\big(L\big(a_j,b_j,\mu_j + 2\big(|a_j| + |b_j|\big)\big)\big) \tag{11.73}$$

as shown in Fig. 11.15(d). For an edge $L(a_j,b_j,\mu_j)$, we have the relations

$$K \subset H^-\big(L(a_j,b_j,\mu_j)\big) \tag{11.74}$$

$$U_3^j \subset \bar{H}^-\big(L\big(a_j, b_j, \mu_j + 2\big(|a_j| + |b_j|\big)\big)\big), \tag{11.75}$$

as shown in Fig. 11.15(e), where $1 \le j \le m$, $j \in \mathbb{Z}$ and $U_3^j = U_3 \cap H^-(L(a_j, b_j, \mu_j + 2(|a_j| + |b_j|)))$. From Eqs. (11.74) and (11.75) we derive the conclusion that

$$\bar{H}^-\big(L(a_j, b_j, \mu_j)\big) \cap H^-\big(L\big(a_j, b_j, \mu_j + 2\big(|a_j| + |b_j|\big)\big)\big) \tag{11.76}$$

does not have a common point with either K or U_3^j. Furthermore, Lemma 10 implies that there exists a line $L(\boldsymbol{s}, \boldsymbol{d})$ which includes points defined by Eq. (11.76).The line $L(\boldsymbol{s}, \boldsymbol{d})$ satisfies Lemma 9 if $\boldsymbol{p} \in U_3^j$. Therefore, Lemma 9 implies that all points in $\boldsymbol{p}$ in U_3^j satisfy the relation

$$\exists \boldsymbol{s} \in S, \quad VC(\boldsymbol{s}, D, \boldsymbol{p}) \cap K = \emptyset \tag{11.77}$$

as shown in Fig. 11.15(f). Moreover, since

$$U_3 = \bigcup_{j=1}^{m} U_3^j \tag{11.78}$$

all points of U_3 satisfy Eq. (11.77). □

11.4.3.2 Three-Dimensional Case

The following two propositions are used to prove the three-dimensional case.

Lemma 11 *If a plane $P(a, b, c, \mu)$ containing a source $\boldsymbol{s}$ exists between the point $\boldsymbol{p}$ and the object K, and $P(a, b, c, \mu)$ does not contain either $\boldsymbol{p}$ or K, that is,*

$$\begin{aligned} &K \subset H^-\big(P(a, b, c, \mu)\big), \quad K \cap P(a, b, c, \mu) = \emptyset \\ &\boldsymbol{p} \in H^+\big(P(a, b, c, \mu)\big), \quad \boldsymbol{p} \notin P(a, b, c, \mu) \\ &\quad \Rightarrow \quad VC(\boldsymbol{s}, D, \boldsymbol{p}) \cap K = \emptyset, \end{aligned} \tag{11.79}$$

where $\boldsymbol{s} \in P(a, b, c, \mu)$, then $VC(\boldsymbol{s}, D, \boldsymbol{p})$, and K have no common point.

Lemma 12 *There exists at least a plane $P(a', b', c', \mu')$ which is contained in $\hat{H}$*

$$\begin{aligned} \hat{H} = \bar{H}^-\big(P\big(a, b, c, \mu - \big(|a| + |b| + |c|\big)\big)\big) \\ \cap H^-\big(L\big(a, b, c, \mu + |a| + |b| + |c|\big)\big) \end{aligned} \tag{11.80}$$

and $\boldsymbol{s} \in P(a', b', c', \mu')$, that is, the following relation holds:

$$\exists P\big(a', b', c', \mu'\big), \quad P\big(a', b', c', \mu'\big) \subset \hat{H}. \tag{11.81}$$

Equation (11.32) and Lemma 1 imply the relation

$$K = \bigcap_{j=1}^{m} H^-\big(P(a_j, b_j, c_j, \mu_j)\big) \tag{11.82}$$

$$U_3 \subset \bigcup_{j=1}^{m} \bar{H}^{-}\big(P\big(a_j, b_j, c_j, \mu_j + 2\big(|a_j| + |b_j| + |c_j|\big)\big)\big). \tag{11.83}$$

Furthermore, for an edge $L(a_j, b_j, c_j, \mu_j)$, the relations

$$K \subset H^{-}\big(P(a_j, b_j, c_j, \mu_j)\big) \tag{11.84}$$

$$U_3^j = \bar{H}^{-}\big(P\big(a_j, b_j, c_j, \mu_j + 2\big(|a_j| + |b_j| + |c_j|\big)\big)\big) \tag{11.85}$$

hold, where $1 \leq j \leq m$, $j \in \mathbb{Z}$ and $U_3^j = U_j \cap H^{-}(P(a_j, b_j, c_j, \mu_j + 2(|a_j| + |b_j| + |c_j|)))$. Equations (11.84) and (11.85) imply that

$$\bar{H}^{-}\big(P(a_j, b_j, c_j, \mu_j)\big) \cap H^{-}\big(P\big(a_j, b_j, c_j, \mu_j + 2\big(|a_j| + |b_j| + |c_j|\big)\big)\big) \tag{11.86}$$

does not share any point with K and U_3^j. Lemma 12 implies that there exist $P(a', b', c', \mu')$, which is included in the set defined by Eq. (11.86). This plane $P(a', b', c', \mu')$ satisfies the statement if $\boldsymbol{p} \in U_3^j$. Therefore, any point in U_3^j

$$U_3 = \bigcup_{j=1}^{m} U_3^j \tag{11.87}$$

satisfies the relation

$$\exists \boldsymbol{s} \in S, \quad VC(\boldsymbol{s}, D, \boldsymbol{p}) \cap K = \emptyset. \tag{11.88}$$

11.4.4 Non-convex Case

In this section, we prove an upper-bound theorem for non-convex objects in discrete space using the convex hull in the discrete space.

Definition 17 Setting $CH(N)$ to be the convex hull of a non-convex object N in $\mathbf{R}^n$, the supercover of $CH(N')$ is the convex hull of a polytope, where N' is the collection of grid points in N.

The supercovers N_1 and N_2 of the sets of points in N_1' and N_2' satisfies the relation

$$(N_1 \subseteq N_2) \quad \Longrightarrow \quad \big(N_1' \subseteq N_2'\big). \tag{11.89}$$

Since $K \subseteq CH(K)$, we have the relation

$$K \subseteq CH(K), \tag{11.90}$$

and the visual hulls satisfy the relation

$$VH(S, D, K) \subseteq VH\big(S, D, CH(K)\big). \tag{11.91}$$

Moreover, since Theorem 16 implies the relation $VH(S, D, CH(K)) \subseteq (CH(K) \oplus N_0 \oplus N_0)$, we have the following theorem.

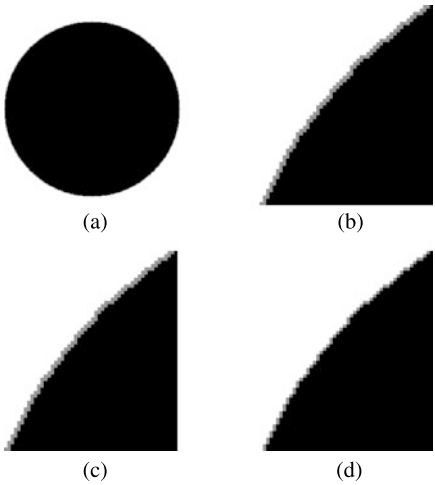


Fig. 11.16 Reconstruction of planar convex objects. (**a**) Reconstructed object using supercovers. (**b**) Enlarged part of the reconstructed object using supercovers. (**c**) Enlarged part of the reconstructed object using standard lines. (**d**) Enlarged part of the reconstructed object using naive lines

Theorem 19 *If an object K is non-convex, then $VH(S, D, K)$ satisfies the relation*

$$VH(S, D, K) \subseteq \big(CH(K) \oplus N_0 \oplus N_0\big), \tag{11.92}$$

for all three types of discrete line.

For two-dimensional discrete objects, we have the following theorem.

Theorem 20 *For two-dimensional non-convex discrete objects, we have the relation*

$$CH(K) \subseteq VH(S, D, K) \subseteq \big(CH(K) \oplus N_0 \oplus N_0\big). \tag{11.93}$$

11.5 Examples

In this section, we show four examples.

1. Reconstruction of a convex object in the plane, which shows ambiguity properties.
2. Reconstruction of a slice-convex object as the intersection of objects reconstructed using a two-dimensional method on each slice.
3. Comparison of three-dimensional reconstructed objects using the supercover, standard, and naive lines.
4. Reconstruction of convex and non-convex objects. An object in the visible hull is reconstructed.

Figure 11.16 shows the reconstruction of a convex object in 2D. Figures 11.16(a), 11.16(b), 11.16(c) and 11.16(d) show a reconstructed object using supercovers, an expanded part of the reconstructed object using supercovers, an expanded part of the reconstructed object using standard lines, and an expanded part of the reconstructed object using naive lines, respectively.

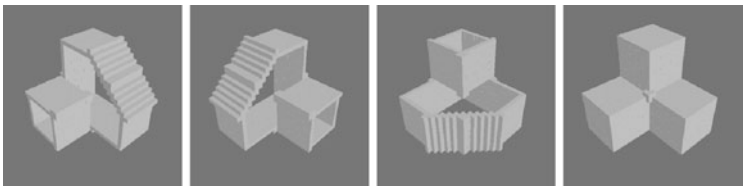


Fig. 11.17 Reconstruction of a non-convex object as an intersection of slice-convex objects. (**a**), (**b**) and (**c**) are reconstructed as an axial convex with respect to the x-, y- and z-axes, respectively. (**d**) is the reconstructed object defined as the intersection of the three objects in (**a**), (**b**) and (**c**)

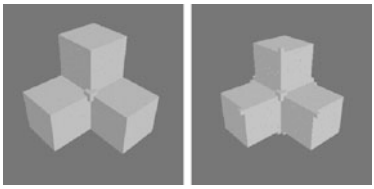
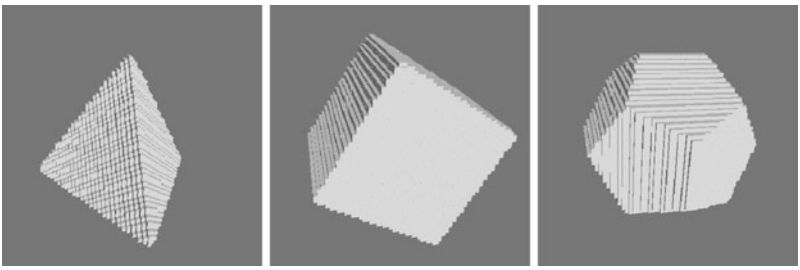


Fig. 11.18 Reconstruction of a non-convex object. (**a**) Multi-slice Method and (**b**) Shape from Silhouette

Figure 11.17 shows a process for the reconstruction of a non-convex object as the intersection of slice-convex objects. Figures 11.18(a) and 11.18(b) show the reconstruction of a non-convex object using multi-slice method and the method of *Shape from Silhouettes*, respectively. Figures 11.19(a), 11.19(b), and 11.19(c) show reconstructed objects using supercovers, standard lines, and naive lines, respectively.

Figures 11.20(a), 11.20(b), and 11.20(c) show the convex polyhedra used for performance evaluation. Table 11.1 shows the number of voxels (card$\{K\} = |K|$) of discrete geometric polyhedra; a tetrahedron, a cube, and dodecahedron in discrete space. The numbers of over-reconstructed voxels are shown in Tables 11.2 and 11.3 for the three types of spatial discrete lines. Furthermore, Fig. 11.22 shows the

Fig. 11.19 Reconstruction of a non-convex object using the method of Shape from Silhouette. (**a**) The reconstructed object using supercovers. (**b**) The reconstructed object using standard lines. (**c**) The reconstructed object using naive lines



(a) Tetrahedron. (b) Cube. (c) Dodecahedron.

Fig. 11.20 Testing convex polyhedra for evaluation of over-reconstruction

Table 11.1 Numbers of voxels (card$\{K\}$) of discrete geometric polyhedra

	card$\{K\}$
Tetrahedron	17825
Cube	73759
Dodecahedron	65193

Table 11.2 Numbers of over-reconstructed voxels (card$\{VH(S, D, K)\backslash K\}$)

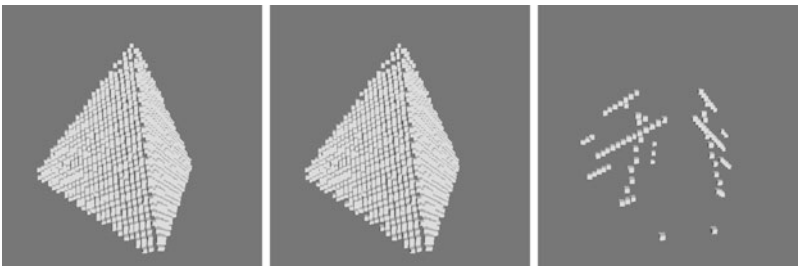
	Supercover	Standard	Naive
Tetrahedron	1275	1252	86
Cube	151	133	0
Dodecahedron	320	300	0

reconstruction of a non-convex polyhedra. Table 11.4 shows the number of voxels (card$\{K\}$) of three discrete non-convex polyhedra. Geometric properties of the numbers of over-reconstructed voxels are shown in Tables 11.5, 11.6, and 11.7 for the three types of spatial discrete lines. Figure 11.21 shows over-reconstruction of the discrete tetrahedron $VH(S, D, K)\backslash K$.

Figure 11.23 shows the reconstruction of the discrete objects Knight and Human from the IAPR TC18 test [1] data and the Hilbert curve of order 3 with step 5. Table 11.8 shows the number of voxels (card$\{K\}$) of three discrete data objects. The numbers of over-reconstructed voxels are shown in Tables 11.9 and 11.10 for the three types of spatial discrete lines.

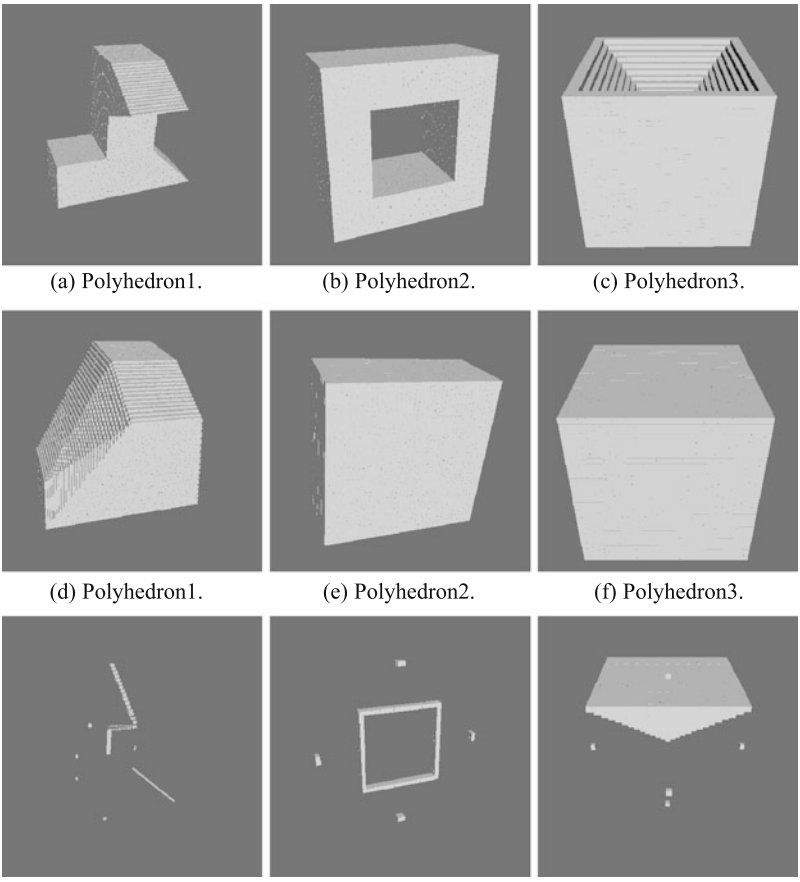
Table 11.3 card$\{(K \oplus N_0 \oplus N_0)\backslash VH(S, D, K)\}$. If the entry of the table is 0, $VH(S, D, K) \subseteq K \oplus N_0 \oplus N_0$

	Supercover	Standard	Naive
Tetrahedron	0	0	0
Cube	0	0	0
Dodecahedron	0	0	0



(a) Supercover. (b) Standard. (c) Naive.

Fig. 11.21 Over-reconstruction of the discrete tetrahedron $VH(S, D, K) \backslash K$



(a) Polyhedron1. (b) Polyhedron2. (c) Polyhedron3.

(d) Polyhedron1. (e) Polyhedron2. (f) Polyhedron3.

(g) Polyhedron1. (h) Polyhedron2. (i) Polyhedron3.

Fig. 11.22 Numerical evaluation of non-convex objects. From *top* to *bottom*: Discrete objects K; Convex hull $CH(K)$ of K; Over-reconstruction $VH_{Sup}(S, D, K) \backslash K$

Table 11.4 Numbers of voxels in (card$\{K\}$)

	card$\{K\}$
Polyhedron1	18792
Polyhedron2	146568
Polyhedron3	26137

Table 11.5 Numbers of over-reconstructed voxels (card$\{VH(S, D, K)\backslash K\}$)

	Supercover	Standard	Naive
Polyhedron1	90	85	0
Polyhedron2	168	72	0
Polyhedron3	3660	3660	3654

11.6 Discussion

In this section we consider cases of reconstruction of different shapes. At the end of the section we consider some open problems.

11.6.1 Reconstruction of Spatial String

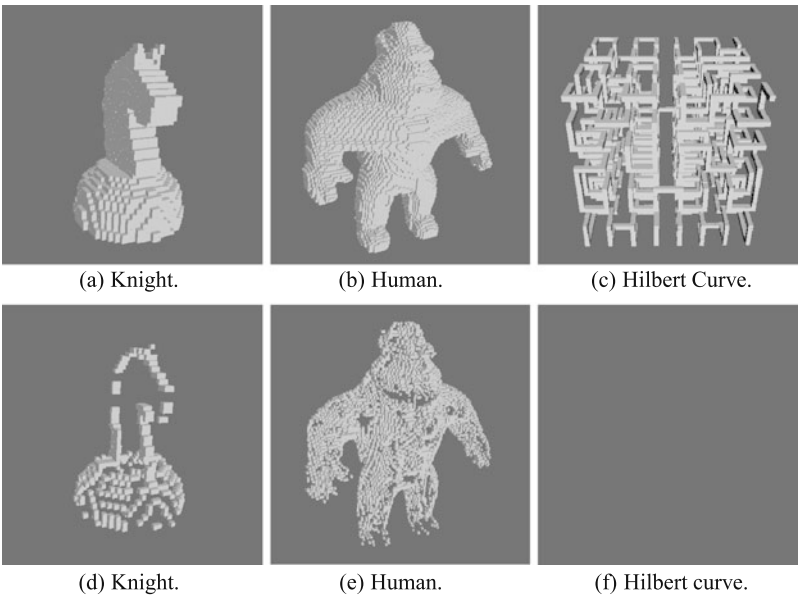
The reconstruction results for non-convex objects show that the Hilbert curve is completely reconstructed from its silhouettes in the discrete space. This property of the Hilbert curve suggests that we can use the method of *Shape from Silhouettes* for the reconstruction of string-like objects in the space, since string-like objects are locally multi-axial convex. DNA and proteins are biological string-like objects

Table 11.6 card$\{(CH(K) \oplus N_0 \oplus N_0)\backslash VH(S, D, K)\}$. If the entry of the table is 0, $VH(S, D, K) \subseteq CH(K) \oplus N_0 \oplus N_0$

	Supercover	Standard	Naive
Polyhedron1	0	0	0
Polyhedron2	0	0	0
Polyhedron3	0	0	0

Table 11.7 card$\{(K \oplus N_0 \oplus N_0)\backslash VH(S, D, K)\}$. If the entry of the table is 0, $VH(S, D, K) \subseteq K \oplus N_0 \oplus N_0$

	Supercover	Standard	Naive
Polyhedron1	0	0	0
Polyhedron2	0	0	0
Polyhedron3	1331	1331	1330



(a) Knight. (b) Human. (c) Hilbert Curve.

(d) Knight. (e) Human. (f) Hilbert curve.

Fig. 11.23 Performance evaluation by test object sets. The images in the *top row* are the discrete objects K and the images in *bottom row* are the over-reconstruction$(VH_{Sup}(S, D, K)\backslash K)$

Table 11.8 Numbers of voxels of discrete objects (card$\{K\}$)

	card$\{K\}$
Knight	6513
Human	70413
Hilbert curve	2556

Table 11.9 Numbers of over-reconstructed voxels (card$\{VH(S, D, K)\backslash K\}$)

	Supercover	Standard	Naive
Knight	396	216	18
Human	5187	5010	1213
Hilbert curve	0	0	0

Table 11.10 card$\{(K \oplus N_0 \oplus N_0)\backslash VH(S, D, K)\}$. If 0, then $VH(S, D, K) \subseteq K \oplus N_0 \oplus N_0$

	Supercover	Standard	Naive
Knight	0	0	0
Human	0	0	0
Hilbert curve	0	0	0

whose silhouettes are measured by electron microscopy computerized tomography [6, 7, 11, 19]. Therefore, these string-like objects in the space are reconstructed using reconstruction techniques for the traditional x-ray tomography, which are based on the Radon transform [8, 18, 21]. Our results show that for the reconstruction of the shape of DNA and the proteins, the method of *Shape from Silhouettes* will help to remove artifacts in the reconstructed objects caused by the numerical computation of the inverse Radon transform.

11.6.2 Shape Carving

From Theorem 16, it follows that in shape carving and visible voting, smoothing and weighting, respectively, there are operations to yield K' such that

$$\left|\hat{K} \Delta K'\right| > \left|R' \Delta R\right|, \quad K \subseteq K' \subset \hat{K}, \tag{11.94}$$

where

$$A \Delta B = (A \cap \overline{B}) \cup (\overline{A} \cap B) \tag{11.95}$$

and $|A|$ is the number of elements in the set A.

For the reconstruction, we define the modified naive and modified standard lines as

$$\left\{(x, y) \middle| -\frac{1}{2}|\boldsymbol{a}|_\infty \le \boldsymbol{a}^\top \boldsymbol{x} + \mu < \frac{1}{2}|\boldsymbol{a}|_\infty \right\}, \tag{11.96}$$

$$\left\{(x, y) \middle| -\frac{1}{2}|\boldsymbol{a}|_1 \le \boldsymbol{a}^\top \boldsymbol{x} + \mu < \frac{1}{2}|\boldsymbol{a}|_1 \right\}. \tag{11.97}$$

In $\mathbb{Z}^3$,

$$\begin{aligned} -\frac{1}{2}\big(|a| + |c|\big) &\le ax + bz + \mu_1 < \frac{1}{2}\big(|a| + |c|\big), \\ -\frac{1}{2}\big(|a| + |b|\big) &\le ay + cz + \mu_2 < \frac{1}{2}\big(|a| + |b|\big), \\ -\frac{1}{2}\big(|b| + |c|\big) &\le cx - by + \mu_3 < \frac{1}{2}\big(|b| + |c|\big), \end{aligned} \tag{11.98}$$

is the modified standard line of the line

$$\begin{aligned} ax + bz + \mu_1 &= 0, \\ ay + cz + \mu_2 &= 0, \\ cx - by + \mu_3 &= 0 \end{aligned} \tag{11.99}$$

in $\mathbb{R}^3$. The modified lines used for reconstruction eliminate some pixels/voxels in $\{(K \oplus N_0 \oplus N_0) \setminus K\}$ since these lines are symmetrical around the continuous line.

11.6.3 Approximation of the Hybrid Model

In practice, we deal with the case $\boldsymbol{s} \in \mathbb{R}^n$ and $\boldsymbol{d} \in \mathbb{Z}^n$. We call this setting a hybrid model of shape reconstruction. For the case $n = 2$, $\boldsymbol{s} = (s_1, s_2)^\top$, there exist a pair of rational numbers q/p and r/p which approximate s_1 and s_2, respectively. Therefore, we can approximate the pair $\boldsymbol{s} \in \mathbb{R}^2$ and $\boldsymbol{d} \in \mathbb{Z}^2$ with the pair $\overline{\boldsymbol{s}} \in \mathbb{Q}^2$ and $\boldsymbol{d} \in \mathbb{Z}^2$, where $\mathbb{Q}$ is the set of all rational numbers. The line which passes through $\overline{\boldsymbol{s}} \in \mathbb{Q}^2$ and $\boldsymbol{d} \in Z^2$ is represented as

$$Ax + By + M = 0, \tag{11.100}$$

where A, B, and M are integers. However, $gcd(A, B) = g \geq 1$. Therefore, if the thickness of a discrete line is defined by A/g and B/g, for example, the supercover is expressed as

$$|Ax + By + M| \leq \frac{1}{2}\left(\frac{|A|}{g} + \frac{|B|}{g}\right). \tag{11.101}$$

This representation allows us to apply the algorithm presented earlier to the hybrid model.

11.6.4 Open Problems

For the square $D^2(k)$ in $\mathbb{Z}^2$ whose vertices are

$$(-k, -k)^\top, (-k, k)^\top, (k, k)^\top, (k, -k)^\top, \tag{11.102}$$

if the source moves in the region

$$D_m^2 = \bigcup_{k=n}^{m} D^2(k), \tag{11.103}$$

we have many more lines than in the case that the source moves on the $D(3n)$, which is the geometry we introduced in Sect. 11.4.1. For this geometry, the error bound for the ambiguity theorem is an open problem. The error bound for the spatial case is also an open problem.

The above problem, related to dealing with different sizes of reconstruction space, is equivalent to selecting the resolutions of objects in the space and the silhouettes on the imaging planes. Since the size of the neighborhood N_0 becomes relatively small in higher resolutions, we can have the following conjecture.

Conjecture 1 *The over-reconstruction* $(K \oplus N_0 \oplus N_0)$ *converges to zero, that is,*

$$\lim_{\Delta \to 0} \left\{(K \oplus N_0 \oplus N_0) \setminus K\right\} = \emptyset, \tag{11.104}$$

$$\lim_{\Delta \to 0} \left|\left\{(K \oplus N_0 \oplus N_0) \setminus K\right\}\right| = 0, \tag{11.105}$$

where Δ *is the size of the length of a pixel and a voxel in the plane and in the space, respectively.*

To analyze the mathematical properties of this conjecture, we have to use the theory of multi-resolution discrete geometry [4, 23, 24].

11.7 Conclusions

In this paper, we formulated the method of *Shape from Silhouettes* in two- and three-dimensional discrete space. This approach to the problem implied an ambiguity theorem for the reconstruction of objects in discrete space. Furthermore, we showed that in three-dimensional space, it is possible to reconstruct a class of non-convex objects from a set of silhouettes although in the plane a non-convex object is unreconstructible from projections.

Shape reconstruction from a series of silhouettes is a conventional technique for the detection of shape models and shape reconstruction in computer graphics, computer vision [13], and robotics [16, 22, 25, 26]. Although these reconstruction methods are mathematically formulated in the continuous framework [9, 20], the reconstruction is achieved in discrete space. Therefore, we dealt with the *Shape from Silhouette* problem in discrete space and showed the upper-bound of over-reconstruction of objects.

Acknowledgements This research was supported by the grant "Computational anatomy for computer-aided diagnosis and therapy: Frontiers of medical image sciences" funded by the Grant-in-Aid for Scientific Research on Innovative Areas, MEXT, Japan, the Grants-in-Aid for Scientific Research funded by Japan Society of the Promotion of Sciences, Japan. A part of the research is based on the Master's thesis of Kosuke Sato submitted to the School of Science and Technology, Chiba University 2007.

References

1. http://tc18.liris.cnrs.fr/
2. Aloimonos, J.: Visual shape computation. Proc. IEEE **76**, 899–916 (1988)
3. Boltyanski, V., Martin, H., Soltan, P.S.: Excursions into Combinatorial Geometry. Springer, Berlin (1997)
4. Chollet, A., Wallet, G., Fuchs, L., Andres, A., Largeteau-Skapin, G.: Ω-Arithmetization: a discrete multi-resolution representation of real functions. In: IWCIA 2009. Lecture Notes in Computer Science, vol. 5852, pp. 316–329 (2009)
5. Dobkin, D.P., Edelsbrunner, H., Yap, C.K.: Probing convex polytopes. In: Proc. 18th ACM Symposium on Theory of Computing, pp. 424–432 (1986)
6. Frank, J. (ed.): Electron Tomography: Methods for Three-Dimensional Visualization of Structures in the Cell, 2nd edn. Springer, Berlin (2006)
7. Herman, G.T., Kuba, A. (eds.): Advances in Discrete Tomography and Its Applications. Applied and Numerical Harmonic Analysis. Birkhäuser, Boston (2007)
8. Herman, G.T., Kuba, A. (eds.): Discrete Tomography: Foundations, Algorithms, and Applications. Applied and Numerical Harmonic Analysis. Birkhäuser, Boston (1999)
9. Imiya, A., Kawamoto, K.: Shape reconstruction from an image sequences. In: Lecture Notes in Computer Science, vol. 2059, pp. 677–686 (2001)

10. Imiya, A., Kawamoto, K.: Mathematical aspects of shape reconstruction from an image sequence. In: Proc. 1st Intl. Symp. 3D Data Processing Visualization and Transformations, pp. 632–635 (2002)
11. Jordan, E.K., Kirby, R.M., Silvia, C., Peters, T.: Through a new looking glass*: mathematically precise visualization. SIAM News **43**(5) (2010). Archive: http://www.siam.org/news/archives.php
12. Kawamoto, K., Imiya, A.: Detection of spatial points and lines by random sampling and voting process. Pattern Recognit. Lett. **22**, 199–207 (2001)
13. Kutulakos, K., Seitz, S.M.: A theory of shape by space carving. In: Proceedings of 7th ICCV, vol. 1, pp. 307–314 (1999)
14. Laurentini, A.: The visual hull concept for silhouette-bases image understanding. IEEE Trans. Pattern Anal. Mach. Intell. **16**, 150–163 (1994)
15. Laurentini, A.: How a 3D shape can be understood from 2D silhouettes. IEEE Trans. Pattern Anal. Mach. Intell. **17**, 88–195 (1995)
16. Li, R.S.-Y.: Reconstruction of polygons from projections. Inf. Process. Lett. **28**, 235–240 (1988)
17. Lindembaum, M., Bruckstein, A.: Reconstructing a convex polygon from binary perspective projections. Pattern Recognit. **23**, 1343–1350 (1990)
18. Natterer, F.: The Mathematics of Computerized Tomography. Wiley, New York (1986)
19. Plitzko, J.M., Baumeister, W.: Cryoelectron tomography (CET). In: Hawkes, P.W., Spence, J.C.H. (eds.) Science of Microscopy, 2nd edn., pp. 535–604. Springer, Berlin (2006)
20. Prince, J.L., Willsky, A.S.: Reconstructing convex sets from support line measurements. IEEE Trans. Pattern Anal. Mach. Intell. **12**, 377–389 (1990)
21. Radon, J.: Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten. Ber. Verh. Sächs. Akad. Wiss. **69**, 262–277 (1917)
22. Rao, A.S., Goldberg, Y.K.: Shape from diameter: recognizing polygonal parts with parallel-jaw gripper. Int. J. Robot. Res. **13**, 16–37 (1994)
23. Said, M., Lachaud, J.-O., Feschet, F.: Multiscale discrete geometry. In: DGCI 2009. Lecture Notes in Computer Science, vol. 5810, pp. 118–131 (2009)
24. Said, M., Lachaud, J.-O., Feschet, F.: Multiscale analysis of digital segments by intersection of 2D digital lines. In: Proc. ICPR 2010, pp. 4097–4100 (2010)
25. Skiena, S.S.: Interactive reconstruction via geometric probing. Proc. IEEE **80**, 1364–1383 (1992)
26. Skiena, S.S.: Probing convex polygon with half-planes. J. Algorithms **12**, 359–374 (1991)
27. Tuy, H.K.: An inversion formula for cone-beam reconstruction. SIAM J. Appl. Math. **43**, 546–552 (1983)

Chapter 12
Combinatorial Maps for 2D and 3D Image Segmentation

Guillaume Damiand and Alexandre Dupas

Abstract This chapter shows how combinatorial maps can be used for 2D or 3D image segmentation. We start by introducing combinatorial maps and we show how they can be used to describe image partitions. Then, we present a generic segmentation algorithm that uses and modifies the image partition represented by a combinatorial map. One advantage of this algorithm is that one can mix different criteria and use different image features which can be associated with the cells of the partition. In particular, it is interesting that the topological properties of the image partition can be controlled through this approach. This property is illustrated by the computation of classical topological invariants, known as Betti numbers, which are then used to control the number of cavities or the number of tunnels of regions in the image partition. Finally, we present some experimental results of 2D and 3D image segmentation using different criteria detailed in this chapter.

12.1 Introduction

In the image analysis framework, image segmentation is one of the main issues, and probably the most discussed one in the literature. Image segmentation methods were subject of numerous research papers. Surveys on existing approaches can be found, for example, in [22–24, 32, 36]. Image segmentation is usually the first step before any algorithm for computer vision, such as object recognition, is applied. The segmentation operation consists of grouping the elements of an image, usually known as pixels or voxels (for 2D and 3D images, respectively), in homogeneous areas called regions. Each region is uniform regarding some properties based on

G. Damiand (✉)
LIRIS, UMR5205, Université de Lyon, CNRS, 69622 Lyon, France
e-mail: guillaume.damiand@liris.cnrs.fr

A. Dupas
Unit 698, Inserm, 75018 Paris, France
e-mail: alexandre.dupas@gmail.com

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_12,

intensity (gray levels), texture, or colors. The set of regions forms a partition of the image elements, and thus any pixel or voxel of the image belongs to exactly one region.

For efficiency purposes, some segmentation algorithms need an effective representation of the image partition and operations so that the partition can easily be modified. The cost related to the partition modification is closely connected to the data structure related to the particular segmentation algorithm. One of the first data structures described in the literature is the region adjacency graph, called RAG [33]. Regions of the image are represented by the vertices of a graph, and the edges are linking each pair of vertices if the two corresponding regions are adjacent. However, a RAG does not describe all the relations between regions in 2D images, and in higher dimensions it is even more so. To overcome this issue, other solutions are available in the literature. In dual graphs approach [28, 35], two graphs are used: a RAG is coupled with its dual. While dual graphs solve some of the issues of RAG, this solution is not complete, and moreover it is meant to only represent 2D images partitions.

Several solutions based on 2D combinatorial maps have been proposed [5–7, 9, 14, 19], and then extended in 3D [4, 8]. These models have advantages that justify their use. First, combinatorial maps fully describe the topology of the image partition in regions, that is, the representation of all cells of the partition and all the incidence and adjacency relations between these cells. Second, combinatorial maps allow efficient algorithms to retrieve information and modify the partition. Finally, combinatorial maps are defined in any dimension allowing to easily generalize the algorithms. For these reasons, combinatorial map based models have been used in many works in image segmentation [2, 3, 13, 15, 26].

The objective of this chapter is to present all the basic knowledge required to implement a generic 2D and 3D image segmentation algorithm based on combinatorial maps. We start in Sect. 12.2 by introducing combinatorial maps and the related definitions. Then, we give several notions about 2D and 3D images, and finally we define topological maps, which are specific combinatorial maps describing a partition of an image in regions. In Sect. 12.3 we give the generic segmentation algorithm based on the global merging operation. The algorithm is generic because it is defined in any dimension, and it takes as parameter two functions allowing to control its behavior depending on the image properties. To illustrate the genericity, we introduce four different segmentation criteria based on different information. As our interest in combinatorial maps is to fully describe the topology of image partitions, we use this asset and propose a segmentation criterion based on a topological invariant: the Betti numbers. In Sect. 12.4, we show how to compute Betti numbers in the topological map framework, and we propose two segmentation criteria that allow, with the generic segmentation algorithm, to explore a segmentation method that take into account some topological features. Finally, Sect. 12.5 presents some experiments of 2D and 3D image segmentation using the different criteria proposed in this chapter.

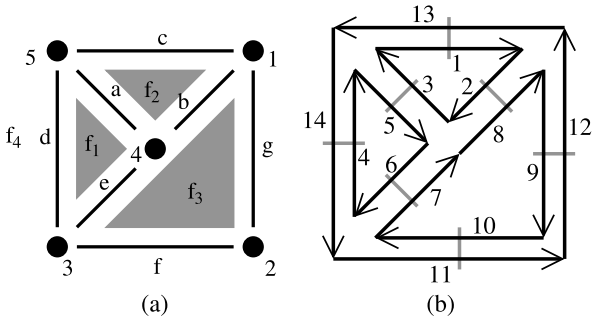


Fig. 12.1 (**a**) Example of 2D orientable subdivision. The object is composed of five 0-cells (numbered from *1* to *5*), seven 1-cells (labeled from a to g) and four 2-cells (labeled f_1 to f_4). Edges a and b are adjacent since they share vertex *4*, and faces f_1 and f_2 are adjacent along edge a. Thus, vertex *4* is incident to edge a, and edge a is incident to face f_1. By transitivity, vertex *4* is incident to face f_1. (**b**) Corresponding 2D combinatorial map. Darts are represented by numbered *black segments* ending with *arrows*. Two darts linked by β_1 are drawn consecutively (for example, $\beta_1(1) = 2$) and two darts linked by β_2 are drawn parallel to each other in reverse orientation connected by a little *gray segment* (for example, $\beta_2(1) = 13$)

12.2 Topological Maps

In this section, we introduce the basic notions leading to the definition of a topological model used to represent 2D and 3D image partition. We start by introducing combinatorial maps that describe subdivided objects in any dimension. Then, we recall the basic notions related to images (pixels, voxels, adjacency, regions, inter-elements). Last, we present 2D and 3D topological maps which are specific combinatorial maps that describe 2D and 3D image subdivisions.

12.2.1 Combinatorial Maps

An nD combinatorial map is a model representing an nD subdivided orientable object by describing all its cells, and all the neighborhood relations between these cells. We denote by *i-cell* a cell in dimension i: 0-cells are called *vertices*, 1-cells *edges*, 2-cells *faces*, and 3-cells *volumes*. Neighborhood relations are defined on the basis of the *incidence* and *adjacency* relations. Two cells c_1 and c_2 are adjacent if they have the same dimension i, and if they share a common $(i-1)$-cell c. In this case, c is said to be incident to c_1 and to c_2. Incidence relation is symmetric: if c_1 is incident to c_2, then c_2 is incident to c_1. Moreover, the incidence relation is extended by transitivity: two cells c_1 and c_2 are incident if there is a path of cells starting from c_1 to c_2 such that each couple of consecutive cells are incident (see Fig. 12.1(a) for an example in 2D).

Any orientable nD subdivided object cannot be described by an nD combinatorial maps: only quasi-manifold orientable objects without boundary can. Quasi-manifold means that an object consists only of $(n-1)$D quasi-manifold orientable

objects glued together along $(n-1)$-cells. Note that in 2D, quasi-manifolds are manifolds, but this is no more true in higher dimension. "Orientable" means that it is possible to define a global orientation "left" and "right" in each point of the object. Lastly, "without boundary" means that each $(n-1)$-cell is without boundary, and that the boundary of each n-cell is fully described by $(n-1)$-cells.

An nD combinatorial map is defined as a set of basic elements, called *darts*, with one to one mappings defined onto the set of darts. Each dart describes a part of a 0-cell, a 1-cell, ..., an n-cell. The mappings β allow to link together darts and thus group the darts that describe the same cell. Definition 1 gives the definition of nD combinatorial maps (see [29, 30] for definitions and more details on combinatorial maps).

Definition 1 (*nD combinatorial map*) An *nD combinatorial map* (or an n-map) is a $n+1$-tuple $M = (D, \beta_1, \ldots, \beta_n)$ where:

1. D is a finite set of darts;
2. β_1 is a *permutation*[1] on D;
3. $\forall i\colon 2 \leq i \leq n\colon \beta_i$ is an *involution*[2] on D;
4. $\forall i, j\colon 1 \leq i < i+2 \leq j \leq n\colon \beta_i \circ \beta_j$ is an involution.[3]

Intuitively, given a dart of an n-map, β_1 gives the next dart of the same face, and β_i gives the dart of the adjacent i-cell (see example in Fig. 12.1(b)). The property that the represented object is without boundary ensures that any dart is linked to another dart by β_i, which is a prerequisite of the permutation property. Moreover, the quasi-manifold property ensures that there are at most two i-cells along each $(i-1)$-cell belonging to the same $(i+1)$-cell, which explains why β_i is an involution. We denote by β_0 the permutation β_1^{-1}. Note that this is only a notation and not a new permutation.

The last line of the definition ($\beta_i \circ \beta_j$ is an involution) ensures the quasi-manifold property. This condition guarantees that when two darts of two i-cells are linked by β_{i+1}, all the darts of the two cells are also linked two by two by β_{i+1}. Intuitively, this means that two i-cells are either disjointed or completely identified, but they cannot be partially identified.

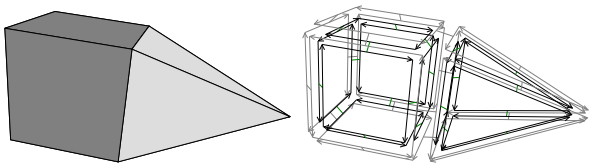
Now, thanks to darts and the β relations, we can retrieve the cells of the subdivision which are implicitly represented in combinatorial maps by sets of darts and by the *orbit* notion.

Definition 2 (*Orbit*) Let $\Phi = \{f_1, \ldots, f_k\}$ be a finite set of permutations on D. We denote by $\langle\Phi\rangle$ 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 with respect to Φ is defined by $\langle\Phi\rangle(d) = \{\phi(d) | \phi \in \langle\Phi\rangle\}$.

[1] A *permutation* on a set D is a one to one mapping from D onto D.

[2] An *involution* f on a set D is a one to one mapping from D onto D such that $f = f^{-1}$.

[3] $\beta_i \circ \beta_j$ is the composition of both permutations: $(\beta_i \circ \beta_j)(x) = \beta_i(\beta_j(x))$.



(a) (b)

Fig. 12.2 (**a**) Example of 3D orientable subdivision. The object is composed by nine 0-cells, sixteen 1-cells, ten 2-cells and three 3-cells (the *cube*, the *pyramid*, plus an unbounded volume not drawn). (**b**) Corresponding 3D combinatorial map having 144 darts. The 64 darts describing the unbounded volumes are drawn in *gray thin lines*

Intuitively, the orbit $\langle \Phi \rangle(d)$ is the set of darts that can be reached from d by using any combination of permutations in Φ. Each i-cell of an nD combinatorial map is obtained by a specific orbit:

Definition 3 (i*-cell*) Let $M = (D, \beta_1, \ldots, \beta_n)$ an n-map, $d \in D$, and $i \in \{0, \ldots, n\}$. The i-cell incident to d, denoted by $c_i(d)$, is:

- if $i = 0$: $\langle \beta_1 \circ \beta_2, \ldots, \beta_1 \circ \beta_n \rangle(d)$;
- otherwise: $\langle \beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_n \rangle(d)$.

Due to the definition of cells as sets of darts, the incident and adjacency relations on cells can easily be tested. Two distinct cells c_1 and c_2 are *incident* if $c_1 \cap c_2 \neq \emptyset$, and two distinct i-cells c_1 and c_2 are *adjacent* if there are two darts $d_1 \in c_1$ and $d_2 \in c_2$ satisfying $d_1 = \beta_i(d_2)$ (or $d_2 = \beta_i(d_2)$ in the case of 1-cells).

We can see an example of 2D combinatorial map in Fig. 12.1(b). In 2D, a 2-map is a triplet $M = (D, \beta_1, \beta_2)$ and the last line of the definition ($\beta_i \circ \beta_j$ is an involution) does not apply. Face f_3 (2-cell) corresponds to $\langle \beta_1 \rangle(7) = \{7, 8, 9, 10\}$, edge a (1-cell) corresponds to $\langle \beta_2 \rangle(3) = \{3, 5\}$ and vertex 1 (0-cell) corresponds to $\langle \beta_1 \circ \beta_2 \rangle(13) = \{2, 9, 13\}$. Edge b and face f_3 are incident since $b = \{2, 8\} \cap f_3 = \{7, 8, 9, 10\} \neq \emptyset$, and f_2 and f_3 are adjacent since $2 \in f_2$, $8 \in f_3$ and $2 = \beta_2(8)$.

One main advantage of combinatorial maps is their definition in any dimension. We can see an example in 3D in Fig. 12.2. A 3D combinatorial map is a 4-tuple $M = (D, \beta_1, \beta_2, \beta_3)$ such that $\beta_1 \circ \beta_3$ is an involution. A 3-cell (volume) corresponds to $\langle \beta_1, \beta_2 \rangle(d)$; a 2-cell (face) to $\langle \beta_1, \beta_3 \rangle(d)$; a 1-cell (edge) to $\langle \beta_2, \beta_3 \rangle(d)$; and a 0-cell (vertex) to $\langle \beta_1 \circ \beta_2, \beta_1 \circ \beta_3 \rangle(d)$.

12.2.2 Removal Operations

The basic operations used to simplify a combinatorial map are the *removal operations*. These operations were first defined on generalized maps [10, 11] and then transposed to combinatorial maps [20, 21]. An i-removal operation allows to remove an i-cell while possibly merging the two incident $(i + 1)$-cells around the

removed cell. There are several removal operations since we can remove an i-cell in an n-map for any i: $0 \leq i < n$.

However, removing an i-cell is not always possible: there is a constraint that the i-cell must satisfy: the notion of *removable cell*. Intuitively the removable constraint ensures that there are at most two $(i+1)$-cells around the removed cell. Otherwise it is not possible to automatically decide how to modify the different $(i+1)$-cells while removing the i-cell.

Definition 4 (*Removable cell*) An i-cell c in an n-map is *removable* if $0 \leq i < n$, and if $i = n-1$ or $\forall d \in c$, $\beta_{i+1} \circ \beta_{i+2}(d) = \beta_{i+2} \circ \beta_{i+1}^{-1}(d)$.

As explained above, the notion of being removable is related to the number of $(i+1)$-cells around the removed cell which is the notion of degree of a cell.

Definition 5 (*Cell degree*) Let c be an i-cell in an n-map, with $0 \leq i < n$. The *degree* of c is the number of $(i+1)$-cells incident to c.

We can easily prove that if an i-cell c is removable, then its degree is at most 2 (i.e. equal to 1 or 2 since it is not possible to have a degree equal to 0 by definition of cells).

Now, we give a generic definition of removal operations. This is Definition 6 which is valid for any removable i-cell with $0 < i < n$. Intuitively, the definition "modifies" only β_i relations for the neighboring darts of the removed i-cell.

Definition 6 (*i-removal operation*) Let $M = (D, \beta_1, \ldots, \beta_n)$ be an n-map, and c a removable i-cell, with $0 < i < n$. The combinatorial map $M' = (D', \beta'_1, \ldots, \beta'_n)$ obtained by removing c from M is defined by:

- $D' = D \setminus c$;
- $\forall j$: $1 \leq j \leq n$: $\forall d \in D'$:
  - if $j = i$ and $d \in \beta_i^{-1}(c) \setminus c$: $\beta'_j(d) = (\beta_j \circ \beta_{j+1})^k(\beta_j(d))$, with k the smaller positive integer such that $(\beta_j \circ \beta_{j+1})^k(\beta_j(d)) \notin c$;
  - otherwise: $\beta'_j(d) = \beta_j(d)$.

$\beta_i^{-1}(c)$ is the set of darts $\{\beta_i^{-1}(d) | d \in c\}$. This is the set of darts which are neighbors of c by β_i. In the removal definition, only darts of $\beta_i^{-1}(c) \setminus c$ have their β_i modified since all the darts of c are removed by the operation. For all these darts, the new β'_j are defined by $\beta'_j(d) = (\beta_j \circ \beta_{j+1})^k(\beta_j(d))$. Intuitively, we jump over the removed darts until we obtain a dart that does not belong to c.

We can see in Fig. 12.3 two examples of removal operations in a 3D combinatorial map. First, we want to remove a 2-cell represented by the darts drawn in bold black in Fig. 12.3(a). The darts, neighbors of the removed darts by β_2 and drawn in bold gray, are the only darts modified by the operation, for example $\beta'_2(1) = \beta_2 \circ \beta_3 \circ \beta_2(1) = 2$. Second, we want to remove the edge represented

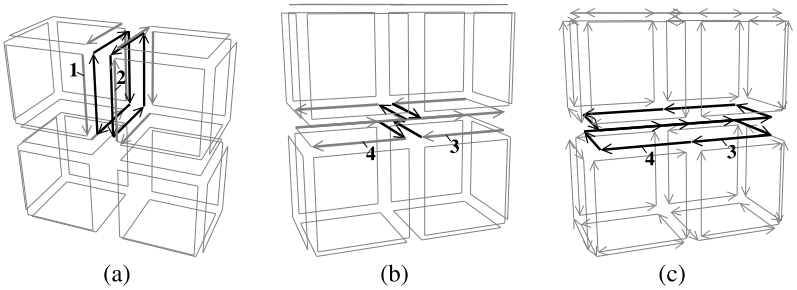


Fig. 12.3 Example of some removals operations in a 3D combinatorial map (only partially drawn). (**a**) The initial configuration from which we want to remove the 2-cell drawn in *bold black*. (**b**) The resulting map obtained after the 2-removal. The two volumes incident to the removed 2-cell are merged. Now we want to remove the 1-cell drawn in *bold black*. (**c**) The resulting map obtained after the 1-removal. The two faces incident to the removed edge are merged

by the darts drawn in bold black in Fig. 12.3(b). Only neighbor darts of these darts by β_0 are modified by the operation, for example $\beta'_1(3) = \beta_1 \circ \beta_2 \circ \beta_1(3) = 4$. Note that this edge is removable in the combinatorial map of Fig. 12.3(b) (in the sense of Definition 4) but not in the combinatorial map of Fig. 12.3(a) because there are more than two 2-cells incident to this edge.

We have a specific case for 0-removal operation, that is the removal of a vertex. This is due to the inhomogeneous definition of combinatorial maps where β_1 is a permutation while other β are involutions. We can see in Definition 7 the main difference with the generic definition. Indeed, we need to modify all the β links of neighbor darts of the removed cell. Moreover, the definition of the modified relation is different.

Definition 7 (0-*removal operation*) Let $M = (D, \beta_1, \ldots, \beta_n)$ an n-map, and c a removable 0-cell. The combinatorial map $M' = (D', \beta'_1, \ldots, \beta'_n)$ obtained by removing c from M is defined by:

- $D' = D \setminus c$;
- $\forall j\colon 1 \leq j \leq n\colon \forall d \in D'$:
  - if $d \in \beta_j^{-1}(c) \setminus c\colon \beta'_j(d) = \beta_j((\beta_1)^k(d))$,
    with k the smaller positive integer such that $\beta_j((\beta_1)^k(d)) \notin c$;
  - otherwise: $\beta'_j(d) = \beta_j(d)$.

Here, contrary to the general case, we do not need to only modify β_i, but all the β. This is due to the definition of cells. Indeed, given an i-cell c, with $0 < i < n$, we know that for any dart $d \in c$, $\beta_j(d) \in c$, $\forall j \neq i$. This property ensures that only β_i has to be modified, but this is no more true for vertices.

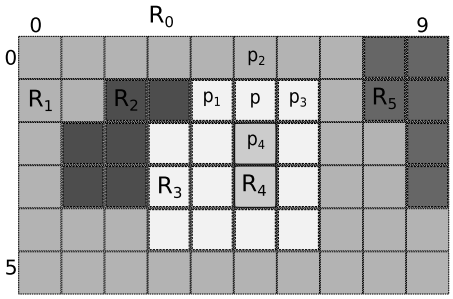


Fig. 12.4 Example of a 2D labeled image having 10 pixels in x axis, and 6 pixels in y axis. Pixel $p = (5, 1)$ belongs to region R_3. Pixel p is 4-adjacent to pixels $p_1 = (4, 1)$, $p_2 = (5, 0)$, $p_3 = (6, 1)$ and $p_4 = (5, 2)$. The image contains five regions (labeled from R_1 to R_5), plus the infinite region R_0. R_4 is enclosed into region R_3, and regions R_2 and R_3 are enclosed into region R_1

12.2.3 Images, Regions and Inter-elements

In this chapter we are interested in 2D and 3D images segmentation, and thus we now recall some usual notions. A pixel (resp. voxel) is an element of the discrete space $\mathbb{Z}^2$ (resp. $\mathbb{Z}^3$), associated with a value (for example a color or a gray level). A 2D image is a set of pixels and a 3D image is a set of voxels.

Two pixels $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ are 4-adjacent if $|x_1 - x_2| + |y_1 - y_2| = 1$. Two voxels $v_1 = (x_1, y_1, z_1)$ and $v_2 = (x_2, y_2, z_2)$ are 6-adjacent if $|x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2| = 1$. A 4-path (resp. 6-path) between two pixels (resp. voxels) b and e is a sequence of pixels (resp. voxels) $(b = e_1, \ldots, e_k = e)$ such that any couple of consecutive pixels (resp. voxels) of the path are 4-adjacent (resp. 6-adjacent).

A set of pixels S (resp. voxels) is 4-connected (resp. 6-connected) if there is a 4-path (resp. 6-path) between any couple of pixels (resp. voxels) of S with all the elements of the path belonging to S.

A region R is a maximal set of 4-connected pixels (resp. 6-connected voxels) having the same label. To avoid having a specific process for the image borders, we consider an infinite region, usually called R_0, that surrounds the image (i.e. this region is the complement of the image). If a region R_j is completely surrounded by a region R_i we say that R_j is *enclosed* in R_i.

We can see in Fig. 12.4 an example of a 2D labeled image and the illustrations of the main notions.

In the interpixel or intervoxel framework [25, 27], pixels or voxels are not the only considered elements. We also consider all the elements of a cellular decomposition of the paving of $\mathbb{Z}^n$. In 2D, pixels are unit squares, linels are unit segments separating two squares, and pointels are points at the extremity of linels. In 3D, voxels are unit cubes, surfels are unit squares separating two voxels, and linels and pointels definitions are similar to the 2D case (see example in Fig. 12.5).

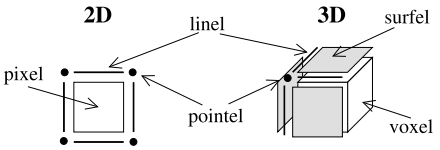


Fig. 12.5 Interpixel and intervoxel elements

12.2.4 Topological Maps

A topological map is a combinatorial map describing an image. For this reason, topological maps are not as general as combinatorial maps since they have some particular properties implied by the specifics of the described partitions. We present here only the main principle of topological maps. Interested readers can found more details in the referenced papers.

Given a 2D labeled image, the problem is to describe the partition in regions using a combinatorial map. We want to describe the multi-adjacency relations between regions, to retrieve all the regions adjacent to a given one, and to know how many times two regions are adjacent. Thus, we have to build a combinatorial map where each edge corresponds exactly to a maximal set of linels between two regions [1, 9]. However, using a combinatorial map is not enough to represent all the information contained in the image. We need to add an interpixel matrix to represent the geometry of the regions, and a tree of regions to describe the enclosed relations between the regions. This is the notion of topological map given in Definition 8.

Definition 8 (*2D topological map*) Given a 2D labeled image, its 2D topological map is a data structure composed of three parts:

- A minimal 2D combinatorial map describing the partition of regions: the external boundary and each cavity of each region is described by one cycle of darts, linked by β_1. Note that the infinite region is a special case since it does not have an external boundary but only one internal boundary. Each edge of the combinatorial map corresponds to a maximal frontier between two regions;
- An interpixel matrix containing all the linels that belong to a region boundary, and all the pointels having a degree greater than two;
- An enclosed tree of regions describing all the enclosed relations between the regions.

The 2D combinatorial map is minimal in number of cells which means there is no combinatorial map that describes the same partition in regions with a smaller number of cells. This minimal property ensures that we represent each maximal frontier between two regions by exactly one edge in the map. We can see in Fig. 12.6 an example of a 2D labeled image and the corresponding 2D topological map.

The topological map definition can be extended in 3D by using a 3D minimal combinatorial map, an intervoxel matrix, and an enclosed tree of regions. This definition is given in Definition 9 and detailed in [8].

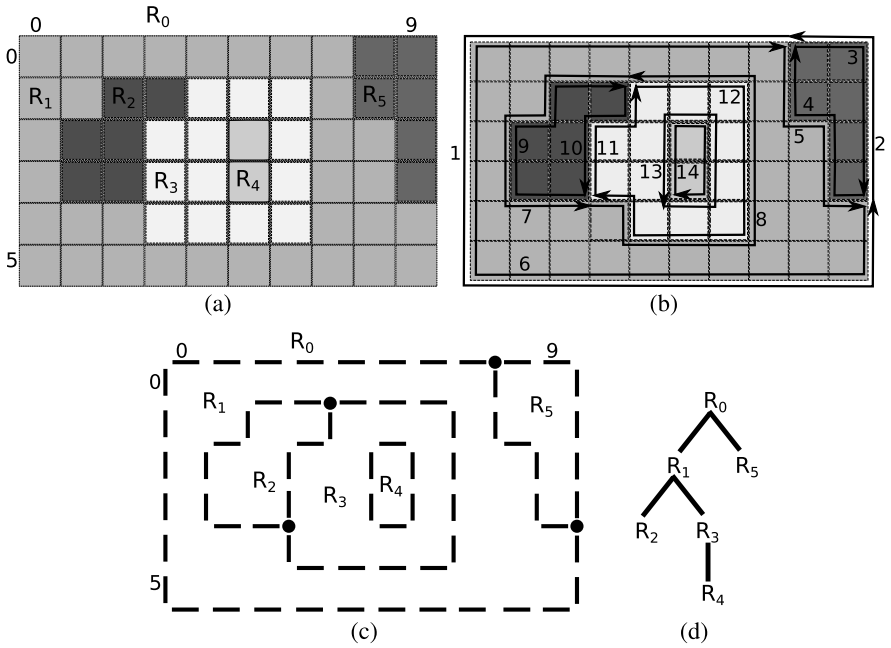


Fig. 12.6 Example of 2D topological map. (**a**) A 2D labeled image. (**b**) The minimal combinatorial map describing the image. Darts are numbered from *1* to *14*. Regions R_2, R_4, and R_5 have only one external boundary, thus only one β_1 cycle of darts. Regions R_1 and R_3 have one cavity, and thus two β_1 cycles of darts, one for their external boundary, and one for their cavity. (**c**) The interpixel matrix. (**d**) The enclosed tree of regions

Definition 9 (*3D topological map*) Given a 3D labeled image, its 3D topological map is a data structure composed of three parts:

- A minimal 3D combinatorial map describing the partition of regions: the external boundary and each cavity of each region is described by volumes (i.e., orbits $\langle \beta_1, \beta_2 \rangle$). As in 2D, the infinite region is a special case since it does not have an external boundary; it has only one internal boundary. Each face of the combinatorial map corresponds to a maximal frontier between two regions, and each edge corresponds to a maximal junction between faces;
- An intervoxel matrix containing all the surfels that belong to a region boundary, all the linels having a degree greater than two, and all the pointels having a degree greater than two;
- An enclosed tree of regions describing all the enclosed relations between regions.

Figure 12.7 shows an example of a 3D labeled image and the corresponding topological map.

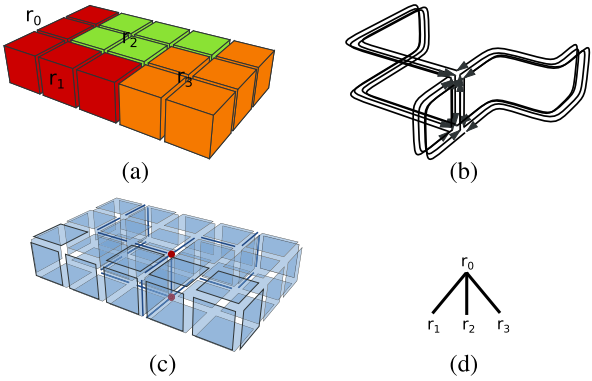


Fig. 12.7 An example of 3D topological map. (**a**) A 3D labeled image. (**b**) The minimal combinatorial map describing the image. (**c**) The intervoxel matrix. (**d**) The enclosed tree of regions

12.3 Image Segmentation Algorithm

Now, we present a bottom-up segmentation algorithm based on topological maps [15]. The guiding principle consists in successively merging adjacent regions satisfying a given criterion starting with an initial partition represented by a topological map. The initial partition can be a partition where each pixel/voxel is in its own region, a partition where pixels/voxels having same color are grouped, a partition which is an over-segmentation of the initial image, or any other kind of partition.

A bottom-up approach is chosen here since it allows to incrementally update regions characteristics without the need of running through all the pixels/voxels of the region. Such incremental update is not possible for top-down or mixed approaches.

12.3.1 The Global Merging Algorithm

The principle of the global merging algorithm consists in merging each couple of adjacent regions of a given topological map satisfying a given criterion. To optimize the algorithm and minimize topological maps modifications, we decompose the process in two steps:

1. Symbolic merging: regions are "merged" by using union-find trees; no modification is made on the topological map;
2. Effective merging: the topological map is modified to group together all regions that belong to the same union-find tree.

For the symbolic merging, we use an union-find tree forest [34] to partition the set of regions in disjointed sets. Each region has a reference to its father in a union-find tree. To handle trees, we use two functions: $find(r)$ which, given a region r, finds the root of the union-find tree, and $union(r_1, r_2)$ which, given two different regions r_1 and r_2 merges their trees. Before starting the symbolic merging, we initialize the pointer of each region to the region itself. This means that each region r is in its own union-find tree since we have $find(r) = r$. During the symbolic merging, an external

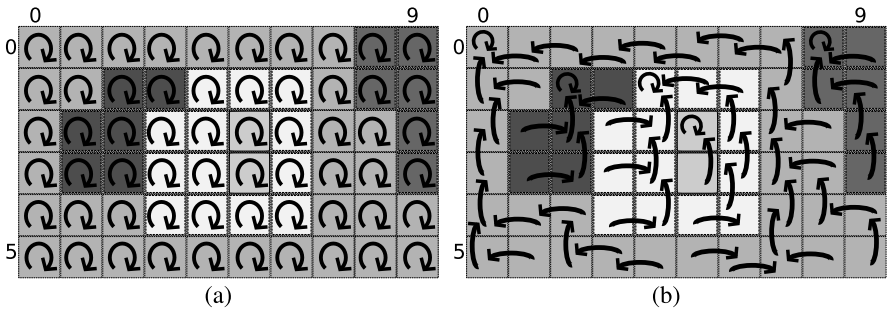


Fig. 12.8 Illustration of the symbolic merging. The father relation of union-find trees is represented by *bold black arrows*. (**a**) An initial forest of union-find trees where each pixel belongs to its own region; thus each pixel is the root of its own union-find tree. (**b**) Result of the symbolic merging of each couple of 4-adjacent pixels having same label (in this example we do not use the two heuristics to simplify the figure). Each pixel has an unique father, and thanks to these union-find tree, we can simply test if two pixels belong to the same region just by testing if $find(p_1) = find(p_2)$

process successively merges couples of regions. The external process can be driven by the user who selects some regions to merge during an interactive session, or as we will see in the next section, by a segmentation algorithm. The merging of the two union-find trees containing regions r_1 and r_2 is simply done by modifying the father pointer of the root of one tree to the root of the other tree. To improve the complexity of this step, we use the two following heuristics:

- We put the smaller tree (in terms of the number of regions) as a child of the largest one during the *union* function;
- We compress the path from region r and its root r' during the $find(r)$ function; for that we assign the father of all the regions between r and its root to r'.

Thanks to these two heuristics, it is proved in [34] that union and find operations on disjoint sets represented by trees can be considered as constant time operations. Figure 12.8 shows a simple example of the symbolic merging.

We use the union-find trees during the second step of the global merging algorithm. The objective of this step, presented in Algorithm 1, is to merge all the regions that belong to a same union-find tree in the topological map. The principle of this algorithm is to test for each couple of adjacent regions r_1 and r_2 whether they belong to the same union-find tree. In this case we remove all the $(n-1)$-cells that separate the two regions.

We can efficiently test each couple of adjacent regions thanks to the relations represented in the topological map. We run through all the darts of the map. Each dart d belongs to its region r_1, and each dart $\beta_n(d)$ belongs to a region r_2 adjacent to r_1. The `foreach` loop allows testing all the couples of adjacent regions in a time linear to the number of darts. Moreover, if two regions are adjacent k times, they are separated by k different $(n-1)$-cells. If the two regions belong to the same union-find tree, all these $(n-1)$-cells are removed since during the `foreach` loop, we will consider one dart for each of these cells, and for each of these darts the condition $find(region(d)) = find(region(\beta_n(d)))$ will be satisfied.

Algorithm 1: Effective merging step

Input: An nD topological map T;
Disjointed sets partitioning the regions of T.
Result: T is modified so that all regions belonging to the same union-find tree are merged in an unique region.

let *toSimplify* be an empty set of darts;
foreach *dart d in T* **do**
  if $find(region(d)) = find(region(\beta_n(d)))$ **then**
    $toSimplify \leftarrow toSimplify \cup$ one dart per $(n-2)$-cell incident to $c_{n-1}(d)$;
    remove $c_{n-1}(d)$;
simplify(*toSimplify*);
recompute the enclosed tree of regions;

If we remove an $(n-1)$-cell, we might need to simplify the topological map because it does not possess the minimal number of cells property anymore. Indeed, the degree of each $(n-2)$-cell incident to the remove cell is decreased by one, and thus these cells can possibly be simplified. To solve this issue, during the effective merging, we keep one dart per each $(n-2)$-cell incident to a removed $(n-1)$-cell, and these cells will be tested during a post-processing simplification step. This simplification step is specialized for 2D and 3D topological maps since in 2D we only need to test and possibly simplify vertices, while in 3D we need to test and possibly simplify edges, then test and possibly simplify vertices (see [9] and [8] for details and Fig. 12.9 for an example in 2D).

12.3.2 The Segmentation Algorithm

In our approach, the segmentation algorithm, given in Algorithm 2, is only a specific case of the global merging algorithm. In fact, as we have seen in the previous section, it is enough to control the symbolic merging step to propose a segmentation algorithm. Then, the effective merging step will modify the initial partition and will produce the topological map representing the result of the segmentation.

Algorithm 2 is a generic segmentation algorithm taking as parameters an nD topological map which describes an initial partition, two functions giving a weight for each $(n-1)$-cell, and a criterion to determine whether an $(n-1)$-cell must be removed.

The initial topological map can be either initialized with a region for each image element, or it can be any initial partition, for example obtained by a pre-segmentation step. The two functions allow to tune the segmentation algorithm by using any kind of information associated with cells and regions of the topological map (see Sect. 12.3.3 for some examples of such functions).

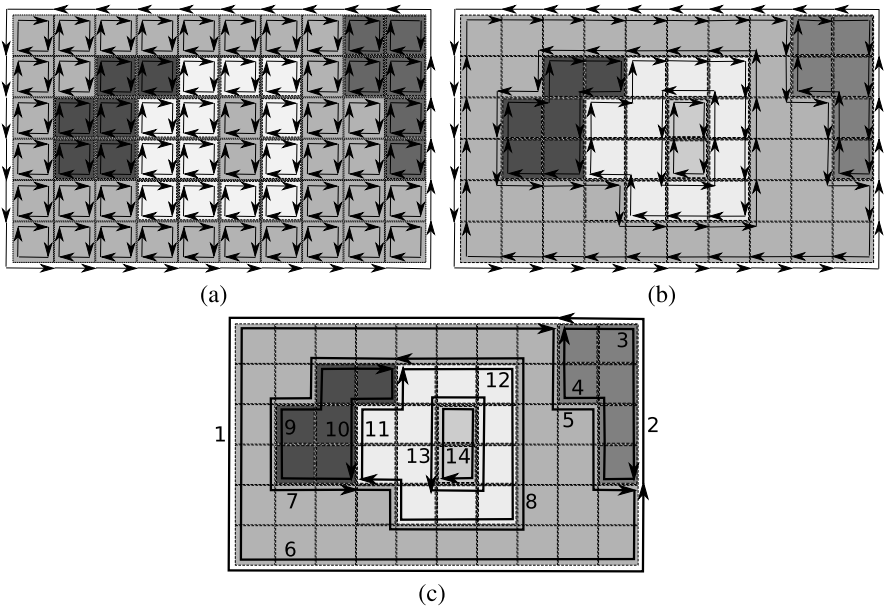


Fig. 12.9 Example of effective merging. (**a**) A topological map describing a 2D labeled image where each pixel is in its own region. (**b**) The combinatorial map obtained after the first step of the effective merging (removal of 1-cells) given the disjoint sets shown in Fig. 12.8(b). This map is not minimal since some vertices are removable. (**c**) The combinatorial map obtained after the simplification step. This is the 2D topological map of the partition in regions given by the disjoint sets

Algorithm 2: Generic segmentation algorithm

Input: An nD topological map T associated with an image I;
A function $weight(c)$ giving a weight of each $(n-1)$-cell;
A boolean function $criterion(d)$ returning true if cell $c_{(n-1)}(d)$ must be removed.

Result: T represents the optimal segmentation of I for the *weight* and *criterion* functions.

$L \leftarrow$ sorted list of one dart per $(n-1)$-cell of T according to *weight*;
foreach *dart* $d \in L$ **do**
  $r_1 \leftarrow region(b)$; $r_2 \leftarrow region(\beta_n(b))$;
  if $r_1 \neq r_2$ ***and*** $criterion(d)$ **then**
    $r \leftarrow$ union(r_1, r_2);
    update region r;
effective merging of regions of T;

Before processing the regions, we start by sorting the list of $(n-1)$-cells according to the *weight* function. This allows us to start with the consideration of a couple of regions that are close with respect to the given *weight*. In Sect. 12.5, we show that this approach gives a better result that processing edges randomly.

The main loop of the algorithm consists of testing all couples of adjacent regions, and in merging them if the given criterion returns a true value. The merging is only made in the symbolic way using the union find trees. The last line of the algorithm is the effective merging step presented in Algorithm 1.

The algorithm is generic since it is defined in any dimension, and can be tuned easily by modifying the weight and criterion functions. Its time complexity is linear in number of darts of the topological map, times the complexity of the criterion function. As we will see in the next section, most of our criteria have constant time complexity, and thus the segmentation algorithm becomes linear in number of darts of the map.

12.3.3 Different Criteria of Segmentation

The main interest of our approach is the genericity and the possibility to mix different criteria associated with a different type of cells (for example a colorimetric criteria associated with the regions, and a gradient associated with the edges). To illustrate these interests, we present here four segmentation criteria, based on:

- The range of gray levels in the regions;
- The gradient of the $(n-1)$-cells separating regions;
- External/internal contrasts of the regions and $(n-1)$-cells;
- The size of the regions and the gradient of $(n-1)$-cells.

All these criteria are defined for nD topological maps, and use different kind of information (cells, adjacency and incidence relations, geometrical information, etc.). Moreover, the information used can often be initialized without additional complexity cost during the topological map construction, and can often be used and updated in constant time which results in efficient segmentation algorithms.

12.3.3.1 Range of Gray Levels

The first version is a basic criteria based on gray levels of pixels. We associate with each region an interval $[g_{min}, g_{max}]$ of min gray level and max gray level of all the pixels contained in the region. The value of each interval is initialized during the construction of the topological map (without modifying the complexity of the construction).

The weight function is given in Algorithm 3. We define the weight of the removal of a cell $c_{n-1}(d)$ by the difference between the length of the new interval of the two merged regions around $c_{n-1}(d)$ and the maximum length of the two original

intervals of r_1 and r_2. Thanks to this definition, the weight is zero if one interval is included into another, or if the two interval are equal. The weight increases if the two region intervals move away.

Algorithm 3: Weight function based on gray level ranges

Input: A dart d of an nD topological map.
Result: The weight of the removal of $c_{n-1}(d)$.

$r_1 \leftarrow region(d)$; $r_2 \leftarrow region(\beta_n(d))$;
$dist_0 \leftarrow \max(r_1.g_{max} - r_1.g_{min}, r_2.g_{max} - r_2.g_{min})$;
$dist_1 \leftarrow \max(r_1.g_{max}, r_2.g_{max}) - \min(r_1.g_{min}, r_2.g_{min})$;
return $dist_1 - dist_0$;

The removal criterion given in Algorithm 4 returns true if the length of the new interval after the merging of the two regions around $c_{n-1}(d)$ is smaller than a threshold τ given by the user.

Algorithm 4: Removal criterion based on gray level ranges

Input: A dart d of an nD topological map;
A threshold τ.
Result: `true` iff $c_{n-1}(d)$ must be removed.

$r_1 \leftarrow region(d)$; $r_2 \leftarrow region(\beta_n(d))$;
$dist_1 \leftarrow \max(r_1.g_{max}, r_2.g_{max}) - \min(r_1.g_{min}, r_2.g_{min})$;
return $dist_1 < \tau$;

Since we associate an interval with each region, we have a direct access to each information associated with the darts ($region(b)$, $\beta_n(d)$, $r.g_{min}$ and $r.g_{max}$) and thus the two algorithms have a constant time complexity.

Moreover, when two regions r_1 and r_2 are merged during the symbolic merging, we can easily, and in constant time, update the interval of the region r which is the union of r_1 and r_2: we only have to set $r.g_{min} \leftarrow \min(r_1.g_{min}, r_2.g_{min})$, and $r.g_{max} \leftarrow \max(r_1.g_{max}, r_2.g_{max})$.

12.3.3.2 Gradient on $(n-1)$-Cells

Another criterion often used in image segmentation is an image gradient. We associate with each $(n-1)$-cell c a value corresponding to the gradient of c. More precisely, for each inter-element i belonging to c the gradient is the sum of the absolute difference of the gray levels of the two image elements around i. As for the

previous criterion, the gradient of each $(n-1)$-cell is initialized during the construction of the topological map without modifying the complexity of the construction. However, gradients are associated with $(n-1)$-cells and not with the regions. This illustrate an interesting feature of topological maps as we can associate information with any cell and possibly mix information associated with different type of cells.

The first approach to use the gradient in the segmentation algorithm is to consider the gradient of each cell $c_{n-1}(d)$ as its weight. Then, the merging criterion returns true if the gradient of $c_{n-1}(d)$ is smaller than a threshold τ given by the user. The problem of this approach is that it does not account for the fact that if we remove an $(n-1)$-cell, it will result in the merging of the two regions around this cell, and thus it will remove all the $(n-1)$-cells between these two regions. If we only consider the current $(n-1)$-cell, the merging criterion can return true because its associated gradient is small, even if merging the two incident regions will remove other $(n-1)$-cells with stronger gradients.

To solve this problem, we define the weight function given in Algorithm 5 which takes into account all the $(n-1)$-cells which will be removed if we merge the two regions incident to the considered cell $c_{n-1}(d)$.

Algorithm 5: Weight function based on gradients

Input: A dart d of an nD topological map.
Result: The weight of the removal of $c_{n-1}(d)$.

```
res ← 0;
r2 ← region(β_n(d));
foreach dart d' ∈ c_n(d) do
    if d' not marked treated then
        if region(β_n(d')) = r2 then
            res ← res + c_{n-1}(d').gradient;
        mark treated all the darts of c_{n-1}(d');
return res;
```

In this algorithm, we run through all the darts $d' \in c_n(d)$ to sum all the gradients of all the $(n-1)$-cells separating the same two regions.

The removal criterion given in Algorithm 6 returns true if the sum of all the gradients of all the $(n-1)$-cells separating the two regions is smaller than a threshold τ given by the user. This test can be achieved by reusing the *weight* function and comparing its value with the threshold.

Due to these definitions, all the $(n-1)$-cells that separate the same couple of regions have the same weight and thus give the same answer for the removal criterion. Moreover, when two regions are merged during the symbolic merging, there is no modification to apply to gradient values. Lastly, during the simplification step of the

Algorithm 6: Removal criterion based on gradients

Input: A dart d of an nD topological map;
A threshold τ.
Result: `true` iff $c_{n-1}(d)$ must be removed.

return $weight(d) < \tau$;

effective merging, if two $(n-1)$-cells are merged, the gradient of the new $(n-1)$-cell is the sum of the two gradients of the original cells. Note that the complexity of the weight and removal criterion algorithms is linear in number of darts of $c_n(d)$.

12.3.3.3 External and Internal Contrasts

This method is based on the segmentation algorithm proposed by Felzenszwalb and Huttenlocher in [17, 18] which uses a criterion based on intensity differences between neighboring pixels in 2D images. In the original work, authors used region adjacency graphs to represent the image partition. In [15] we transposed this method for topological maps and extended it to 3D, but as with the previous criteria, we can extend this criterion in any dimension thanks to the topological maps.

To define this criterion, we associate with each region an internal contrast, called *int*, which is the minimal gray level difference between two adjacent image elements of the region. We also associate an external contrast (called *ext*) with each $(n-1)$-cell c, which is the minimal value for each inter-element i belonging to c of the absolute difference of the gray level of the two image elements around i. These contrasts are initialized during the construction of the topological map.

Thanks to these two values, we can define the weight function given in Algorithm 7 which is equal to the external contrast of the considered $(n-1)$-cell.

Algorithm 7: Weight function based on contrasts

Input: A dart d of an nD topological map.
Result: The weight of the removal of $c_{n-1}(d)$.

return $c_{n-1}(d).ext$;

According to the original work of Felzenszwalb and Huttenlocher [17, 18], the removal criterion given in Algorithm 8 returns true if the external contrast of the considered $(n-1)$-cell is smaller than the minimal internal contrast of the two incident regions. These two internal contrasts are weighted by a threshold function f allowing the user to control the degree to which the external variation can actually be larger than the internal variations. We can use any defined positive function for f. In practice, we use the same function than the one proposed by the authors of the

original method: $f(r) = k/|r|$ with $|r|$ the size of region r (i.e., its number of image elements), and k a constant defined by the user and allowing to tune the algorithm.

Algorithm 8: Removal criterion based on contrasts

Input: A dart d of an nD topological map;
A threshold function f.
Result: `true` iff $c_{n-1}(d)$ must be removed.

$r_1 \leftarrow region(d)$; $r_2 \leftarrow region(\beta_n(d))$;
return $c_{n-1}(d).ext \leq \min(r_1.int + f(r_1), r_2.int + f(r_2))$;

The two algorithms have a constant time complexity since we have a direct access to all the information. Moreover, when two regions are merged in region r during the symbolic merging, it is proved in [17] that the internal contrast of r is equal to the external contrast of the removed cell, while there is no modification on external contrasts. Thus, we can update the contrast in constant time during the symbolic merging.

12.3.3.4 Size of Regions

We present now the last criterion which illustrates one more time the interest of our generic approach. Indeed, thanks to topological maps, we can use different cells and associate different type of information with these cells, but we can also mix colorimetric criteria and geometrical ones. To illustrate this possibility, we present a criterion which mixes the size or regions and the range of gray levels given in Sect. 12.3.3.1. Thanks to these two pieces of information, the proposed criterion allows removing all the small regions (i.e., regions with size smaller than a given threshold) by merging them to the closer regions in their neighborhood (closer in the sense of range of gray levels).

For the weigh function, we use the function already given in Algorithm 3 based on the distance between the ranges of the two regions, and the range of the region after the union. This allows to start processing the $c_{n-1}(d)$ cells that separate closer regions.

For the removal criteria, we use Algorithm 9 which do not use range of gray levels, but which is based on region sizes. This criterion returns true if one of the two regions incident to the considered $(n-1)$-cell is smaller than a given threshold. Thanks to this criterion, we ensure that at the end of the process, all the regions have their size greater than the threshold. As we will see in our experiments, this process can be used as post-processing step to remove small regions that are often due to noise in the original image.

The range of gray levels and the size of regions can be computed during the topological map extraction, and can both be updated in constant time after the merging of two regions during the symbolic merging.

Algorithm 9: Removal criterion based on sizes

Input: A dart d of an nD topological map;
A threshold τ.
Result: `true` iff $c_{n-1}(d)$ must be removed.

$r_1 \leftarrow region(d)$; $r_2 \leftarrow region(\beta_n(d))$;
return $size(r_1) < \tau$ *or* $size(r_2) < \tau$;

12.4 Betti Numbers and Topological Criteria

We presented a generic segmentation algorithm that is parametrized by two functions *weight*, and *criterion*. The algorithm controls the symbolic merging step of the global merge operation to produce the optimal segmentation of the image. We also introduced several criteria based on intensity values of the image, or on simple geometrical property like the region size. While such kind of criteria are commonly used to define segmentation algorithm, topological features are equally important to retrieve useful information from the image segmentation. The presence of a cavity in a 2D region representing a plain object might indicate of a defect in the production factory. Tunnels in a 3D region might be an indication of a segmentation error if, for instance, the segmented object is known for not having any. Thus, we want to define criteria that account for topological features, and that can be used in conjunction with geometric or colorimetric criteria.

In computational topology, topological invariants characterize some of the topological feature we want to take into account. One of the most well known invariants is the Euler characteristic χ of a 2D surface which is linked to the genus of the surface. The genus allows to discriminate between sphere-like surfaces and torus-like surfaces. Definition 10, found in [31], defines the value of the Euler characteristic as the alternating sum of number of cells for cellular complexes. For instance, in 2D, the Euler characteristic is equal to the sum of the number of faces and the number of vertices minus the number of edges.

Definition 10 The Euler characteristic $\chi(r)$ of a region r in an nD space is defined as: $\chi(r) = \sum_{i=0}^{n}(-1)^i \#c_i$, with $\#c_i$ is the number of i-cells belonging to r.

While the Euler characteristic allows to uniquely qualify a topological property for 2D surfaces, it is not the same for 3D volumes like regions in a 3D topological map. Other invariants are, for instance, Betti numbers that express some topological features like the number of cavities or the number of tunnels of a region, and work both in 2D and 3D.

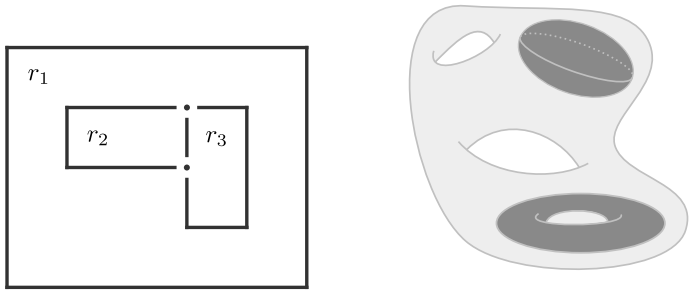


Fig. 12.10 Example of Betti numbers. (**a**) In 2D for region r_1: $b_0 = 1$, $b_1 = 1$; (**b**) In 3D: $b_0 = 1$, $b_1 = 3$, $b_2 = 2$

12.4.1 Betti Numbers

In computational topology, Betti numbers are the rank of the homology group generators, an advanced topological invariant. From a practical point of view, Betti numbers of an object represent the number of *holes* in each dimension. The first Betti number, noted b_0, counts connected components of the object. In 2D, the second Betti number, b_1 counts the number of cavities in the object. In 3D, the second Betti number, b_1 counts tunnels and the third Betti number, b_2 is equal to the number of cavities of the object. For closed oriented nD objects, as regions in a 2D or 3D image, Betti numbers b_k with $k > n$ are equal to zero. For instance, non-zero Betti numbers of the 2D region r_1 in Fig. 12.10(a) are $b_0 = 1$ and $b_1 = 1$ and the non-zero Betti numbers of the 3D object presented in Fig. 12.10(b) are $b_0 = 1$, $b_1 = 3$ and $b_2 = 2$.

Definition 11 of [31], establishes a relation between Betti numbers and the Euler characteristic of an object. The relation gives an alternative approach to compute the Euler characteristic.

Definition 11 The Euler characteristic $\chi(r)$ of a region r in nD is defined as the alternating sum of Betti numbers: $\chi(r) = \sum_{i \leftarrow 0}^{n} (-1)^i b_i(r)$, where the $b_i(r)$ are the Betti numbers of region r.

Computing Betti numbers allows to use the number of connected components, the number of tunnels (in 3D), or the number of cavities to serve as criteria during a segmentation step. In the following sections, we present how Betti numbers can be computed in topological maps and how they can be used as segmentation criteria.

12.4.2 Computation Algorithms Using Topological Maps

In this section, we present the computation of Betti numbers using information provided by the topological map representing an image partition. We avoid the complex

computation of homology group generators as we only require the rank of these groups. The goal is to compute Betti numbers in 2D and 3D image partitions using the practical definition of Betti numbers. Thus, depending on the dimension of the topological map, we count the number of connected components, the number of tunnels, and the number of cavities to obtain the Betti numbers.

In the following, the computation algorithms of Betti numbers of region r of topological map M are explained. First, we detail the algorithms in the 2D case. Second, we present the modifications needed to handle 3D image partitions.

12.4.2.1 Computation of Betti Numbers in 2D Image Partition

The number of connected components of region r in a 2D image partition is equal to the first Betti number $b_0(r)$. By definition of topological maps, a region is a 4-connected set of pixels. Thus, each region has only one connected component. The first Betti number is constant and equal to one: $b_0(r) = 1$ for all region r.

The number of cavities of a region r in a 2D image partition is equal to the second Betti number $b_1(r)$. Note that the set of enclosed regions of region r fills the cavities of region r, and each 8-connected component of the enclosed regions fills exactly one cavity of region r. Counting the number of connected components of enclosed regions allows to retrieve the number of cavities, and thus the second Betti number. In the topological map framework, a tree of region represents the enclosed relation.

The tree of regions is organized so that each connected component $R_{enclosed}$ of regions enclosed in a region r is represented in the tree by a representative region which is in direct relation with r. For instance, r has a direct enclosed relation with $r_i \in R_{enclosed}$, and the other regions within $R_{enclosed} \setminus r_i$ can be retrieved using the connected component relation. Thus, in a 2D topological map, the second Betti number $b_1(r)$ of region r is obtained by counting the number of regions having a direct enclosed relation with r.

12.4.2.2 Computation of Betti Numbers in 3D Image Partition

There are analogies between the definition of the two first Betti numbers in 2D and the definition of the first and third Betti numbers in 3D. By definition of a 3D topological map, regions are 6-connected sets of voxels. Thus, as in 2D, there is only one connected component for each region implying that the first Betti number is constant and equal to one: $b_0(r) = 1$ for any region r.

The third Betti number $b_2(r)$ counts the number of cavities of a region r following the same principle as the second Betti number in a 2D topological map. The 3D region tree represents the enclosed relation, and regions enclosed within region r fill up the cavities of r. The tree of regions is organized as the tree of region in a 2D topological map. Thus, in a 3D topological map, counting the number of regions in direct enclosed relation with region r gives $b_2(r)$, the third Betti number of region r.

12.4.2.3 Computation of the Second Betti Number in 3D Image Partition

In 3D, the second Betti number $b_1(r)$ counts the number of tunnels of region r. As there is no simple way to determine the number of tunnels of a region in 3D topological maps, we use the relation between the Betti numbers and the Euler characteristic given by Definition 11.

The computation of the Euler characteristic using the alternated sum of volumes, faces, edges, and vertices is only valid if the volume is represented by a cellular complex composed of i-cells homeomorphic to i-balls. As the topological map is not a full cellular complex since it only represents cells that belong to the border of the regions of the partition, the represented cells of a region do not satisfy the prerequisite for direct computation of χ using Definition 10.

In [12], the authors present the computation of the sum of Euler characteristics of each border of region r represented by a 3D topological map. We call this sum Euler characteristic of the border of region r denoted $\chi'(r)$. The computation of the alternating sum of cells belonging to the border of a region is possible since the topological map represents all cells belonging to the border of each region.

In [16], the authors define the notion of implicit cells that allows to convert a region represented in a 3D topological map as a cellular complex. Using the fact that the regions in topological maps are composed of only one connected component of voxels, the authors prove Proposition 1 which gives a relation between χ and χ' for region r represented by a 3D topological map.

Proposition 1 *The Euler characteristic $\chi(r)$ is linked to the Euler characteristic of the border $\chi'(r)$ by the relation $\chi(r) = \chi'(r)/2$ with r – a region of a 3D topological map.*

In Proposition 2, we use Definition 11 and Proposition 1 to obtain the second Betti number of region r of a 3D topological map as a function of the Euler characteristic of the border of r, the number of connected components of r, and the number of cavities of r. The complete proof of Proposition 2 can be found in [16].

Proposition 2 *In a 3D topological map, the second Betti number $b_1(r)$ of region r is given by $b_1(r) = b_0(r) + b_2(r) - \chi'(r)/2$.*

We have defined the computation formulas of Betti numbers for regions represented by 2D and 3D topological maps. But these formulas do not allow to compute the Betti numbers for a set of regions as the ones defined during the segmentation process in the symbolic step of the global merging algorithm. To propose such a feature, we introduce in Sect. 12.4.3 the incremental computation algorithms of Betti numbers.

12.4.3 Incremental Computation Algorithms

In Sect. 12.4.2, we have defined simple formulas that allow to compute the Betti numbers of any region in 2D and 3D topological maps. To use Betti numbers as topological criteria during segmentation operation, we need incremental computation algorithms that efficiently update Betti numbers during merge operations.

Symbolic regions are a composition of regions merged in a same disjoint-set of regions during the symbolic step of the global merging algorithm. Incremental computation algorithms have to handle symbolic regions as if they already had been merged together. We use special border coverage algorithms to this purpose removing the need to take a special care of such configurations.

In 2D and 3D image partitions represented by topological maps, the number of connected components per region is constant and equals to one. The value of the first Betti number $b_0(r)$ never changes. The second Betti number in 2D and the third Betti number in 3D count the cavities of the region. In Sect. 12.4.3.1, we propose an incremental approach that updates the number of cavities during the merging of two regions from the symbolic merge step. In Sect. 12.4.2.3, the number of tunnels in a 3D image partition represented by a topological map cannot be obtained directly. To overcome this issue, we give in Sect. 12.4.3.2 algorithms used to incrementally compute the Euler characteristic of the border, the number of connected components, and the number of cavities so that we can use the formula provided by Proposition 2 linking these values to the value of the second Betti number in 3D topological maps.

12.4.3.1 Incremental Computation of the Number of Cavities

The incremental computation of the number of cavities in a topological map during the merging of two regions r_1 and r_2 consists in computing the number of connected components of the enclosed regions after the merge which is the number of internal borders of a region. Let region r_1 and region r_2 be adjacent. In the following, we suppose that region r_1 is not enclosed in region r_2. This leads to three possible configurations (we can assume this because if it is not the case we only have to swap the notations of r_1 and r_2):

1. Merging r_1 and r_2 does not change the number of cavities;
2. Merging r_1 and r_2 removes a connected component of enclosed region, if r_2 fills completely a cavity of r_1;
3. Merging r_1 and r_2 adds new connected components of enclosed regions (either by surrounding new components of enclosed regions or splitting an existing connected component into several chunks).

The number of components of the border of a region is linked to the number of cavities: as there is only one connected component of regions, the number of cavities is equal to the number of borders minus one, the external border. To compute the changes in the number of borders, we use Proposition 3 proved for the 3D case in [16] and also valid in 2D. In the proposition, k is the number of new internal borders.

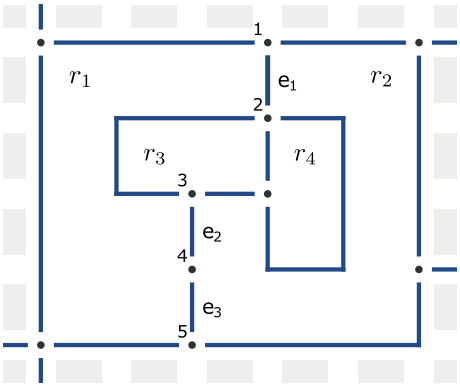


Fig. 12.11 Example of inner cells in a 2D topological map. The inner cells of the union of regions $r_1 \cup r_2$ are the edges e_1, e_2, e_3, and the vertex *4*. Vertices *1*, *2*, *3*, and *5* belong to the boundary of the inner border and thus are not qualified as inner cells

Proposition 3 *Let r_1 and r_2 be two adjacent regions such that $r_1 < r_2$. We have $\#borders(r_1 \cup r_2) = \#borders(r_1) + \#borders(r_2) + k - 2$, where k is the number of borders containing part of the external border of r_2 and that are not between r_1 and r_2.*

We have now a formula linking the number of cavities of a region with the number of borders of the union of two regions. Let us details the algorithm that we use to effectively compute the number of new borders. First, we define the notion of inner border of two regions r_1 and r_2 as the border composed by the cells that lies between r_1 and r_2. We implement traversal algorithms allowing to run through darts of the border ignoring inner borders. That algorithm is a slightly modified version of the classical border traversing algorithm with the exception that if the next dart to be traversed belongs to an inner border, we ignore the corresponding cell (an edge in 2D), and proceed with the next suitable cell (the next non-inner edge around the vertex in 2D). The principle is to cover the border as if region r_1 and r_2 are merged.

An example presenting inner cells in a 2D topological map is presented in Fig. 12.11. The border traversing algorithm, starting in the top left corner of r_1, and going to the right comes to vertex 1. The classical coverage algorithm would go through the edge e_1. Since we want to cover the border as if $r_1 \cup r_2$ is a unique region, the coverage algorithm proceeds with the next edge that is not marked as inner. Thus the next edge is the one leading to the top right corner.

Now, we implement the border counting mechanism which allows to compute the number of borders of the union of two adjacent regions r_1 and r_2. The algorithm first runs through the external border of r_1 and marks as inner and processed each dart d such as $region(\beta_n(d))$ equals r_2. Then, we count the number of connected components of cells that contains a dart of r_1 without traversing any inner border. Using the previously computed and stored value for the number of borders of region r_1 and region r_2, the algorithm returns the number of borders of the union of r_1 and r_2.

Figure 12.12 illustrates the incremental border counting algorithm during the evaluation of the union of regions $r_1 \cup r_2$ on a 2D topological map. In the figure, the border counting algorithm, starting from a dart belonging to region r_2 and edge e_1

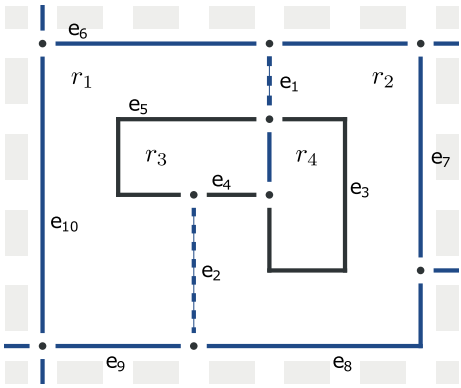


Fig. 12.12 Illustration of the incremental computation of the number of cavities for the union of regions $r_1 \cup r_2$. Edges e_1 and e_2 are inner cells for $r_1 \cup r_2$. For the purpose of this example, we suppose the starting dart belongs to region r_2 and to the edge e_1. The two borders discovered by the algorithm are the internal border composed of edges e_3, e_4, e_5, and the external border composed of edges e_6 through e_{10}

allows to discover two borders. The new internal border is surrounding regions r_3 and r_4 and the external border surrounding regions r_1 and r_2.

Now we can use this algorithm to compute incrementally the number of cavities of two regions that are going to be merged. The differences between the number of cavities in 2D regions and the number of cavities in 3D regions are the cells used to describe the border of a region. In 2D topological maps, border cells are vertices and edges whereas in 3D the border cells are vertices, edges, and faces. The core algorithm does not change except for the orbit used in the border traversal algorithms. In the 2D case, the orbit $\langle \beta_1 \rangle$ gives the darts of the border of a region and the involution β_2 is used to retrieve the region on the other side of the border. In the 3D case, the orbit $\langle \beta_1, \beta_2 \rangle$ describes the border of a region and a dart of the region on the other side of the border is obtained by the involution β_3.

The time complexity of the incremental computation of the number of cavities of the union of regions $r_1 \cup r_2$ depends on the number of darts of the external border of r_2 and the number of darts of the border of r_1 in contact with region r_2.

12.4.3.2 Incremental Computation of the Number of Tunnels in 3D

The incremental computation of the number of tunnels of a region r represented in a 3D topological map consists in computing incrementally each member of the formula presented in Proposition 2. The number of connected components is constant, and since the number of cavities can be obtained incrementally using the method proposed in Sect. 12.4.3.1, we detail here the incremental computation of the Euler characteristic of the border of the union of r_1 and r_2, denoted $\chi'(r_1 \cup r_2)$.

In a 3D topological map, a border is a surface composed of faces, edges, and vertices. There may be one or several surfaces between two adjacent regions; these

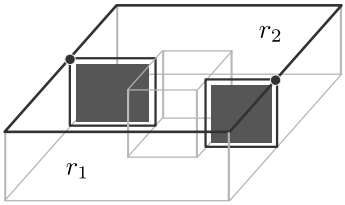


Fig. 12.13 Illustration of the incremental computation of the number of tunnels for the union of regions $r_1 \cup r_2$ in a 3D topological map. Edges drawn in *gray* show the shape of the region. The edges drawn in *black* are actual cells represented in the topological map. The faces drawn in *dark gray* are inner faces, $\chi(inner(r_1, r_2))$ is equal to two (two faces). The values of $\chi'(r_1)$ and $\chi'(r_2)$ are equal to two (two vertices, three edges, and three faces). The resulting value for $\chi'(r_1 \cup r_2)$ is equal to zero which corresponds to the expected value for a torus surface. The number of tunnels obtained after using the formula is one

are the inner surfaces. We call inner cells the cells that entirely belong to the inner surfaces: that is cells such that the region of each dart representing the cell is either r_1 or r_2. Proposition 4 gives an incremental computation formula to obtain the Euler characteristic of the border of the union of two regions r_1 and r_2 from the Euler characteristic of the border of the two regions and the Euler characteristic of the inner surfaces between the two regions. Proof of Proposition 4 is available in [16].

Proposition 4 $\chi'(r_1 \cup r_2) = \chi'(r_1) + \chi'(r_2) - 2\chi(inner(r_1, r_2))$.

Figure 12.13 presents an example of incremental computation of the number of tunnels for the union of regions $r_1 \cup r_2$ detailing the incremental computation of χ'. The resulting region has one tunnel and zero cavity.

We can ignore cells that partially belong to inner surfaces: they form the border of the inner surfaces, and there is an equal number of edges and vertices. Since we are only interested in the Euler characteristic and not in the actual number of cells used to represent the union of regions r_1 and r_2 border, we can ignore these cells: the Euler characteristic of the border of inner surfaces is always zero.

The algorithm that computes incrementally $\chi'(r_1 \cup r_2)$ for the union of two regions r_1 and r_2 is defined as follow. The prerequisite for the incremental computation of χ' is to initialize the χ' value for each region in the first partition represented by a topological map. This is done either by counting cells and using the algorithm proposed in Sect. 12.4.2, or by using the incremental approach during the extraction of the topological map as presented in [12]. The χ' value is stored for each region.

The incremental algorithm is divided in two processing loops. In the first loop, we run through darts of the surface of r_1 that are in contact with r_2. Then each dart belonging to an inner face is marked accordingly. We also count the number of inner faces that are discovered. In the second loop, we count the number of the inner vertices and the number of the inner edges. Using the fact that the darts belonging to inner faces are marked, we conclude that a cell is inner if all darts used to represent the cell are marked as inner. In the final step, we compute the Euler characteristic of the inner border. We retrieve the χ' value for both region r_1 and region r_2, and we

use the formula from Proposition 4 to obtain the Euler characteristic of the border of the union of the two regions around the dart d. Then, with the incrementally computed value for the number of cavities, we use the formula from Proposition 2 to obtain the number of tunnels.

The algorithm has a linear time complexity with the number of darts belonging to $\langle \beta_1, \beta_2 \rangle(d)$ (a subset of the darts used to represent cells of the border of region r_1). In fact, each dart of the orbit is processed at most four times during the course of the algorithm. Darts belonging to the inner cells of $r_1 \cup r_2$ are also covered during the cell counting part of the algorithm, but their number is linearly proportional to the number of darts in the orbit.

12.4.4 Implementation of Topological Criteria in the Segmentation

The incremental computation of Betti numbers allows to anticipate the number of tunnels and cavities during a region merging process. Thus, we can use the number of cavities, or the number of tunnels as a removal criterion during the segmentation.

Weight functions can be defined according to topological properties, for instance, one can prioritize cells whose removal do not change the topology of the regions. However, due to region merging, a cell removal that does not change topological features such as the number of cavities in a simple two regions merging might do so after several other removals. It seems unlikely that such a weight function has any interest for the segmentation part, and thus we keep weight functions given in Sect. 12.3.3 based on intensity values of the image.

We propose to define two new criteria that handle topological properties like:

1. Forbidding topological changes for regions;
2. Reducing the number of cavities.

These criteria can be mixed with other criteria such as the ones presented in Sect. 12.3.3 to control some colorimetric properties of the resulting regions. In such a case, the idea is to check successively the different criteria and allowing the merge if all criteria return a true value.

The first criterion, known as constant topology criterion and given in Algorithm 10, returns true if all the Betti numbers remain constant. That is, if each Betti number of the resulting region is equal to the sum of the same Betti numbers of the two initial regions (except for the number of connected components which is constant).

The second criterion, given in Algorithm 11, returns true if the total number of cavities is reduced. That is if the number of cavities of the resulting region is smaller than the sum of the numbers of cavities of the initial regions.

Since we have a direct access to each information associated with darts ($region(b)$, $\beta_n(d)$) and to each Betti number ($b_i(r)$), the two algorithms have a

Algorithm 10: Removal criterion based on constant Betti numbers

Input: A dart d of an nD topological map.
Result: `true` iff $c_{n-1}(d)$ must be removed.

$r_1 \leftarrow region(d)$; $r_2 \leftarrow region(\beta_n(d))$;
for $i \leftarrow 1$ *to* n **do**
  $b \leftarrow$ compute incrementally b_i for $r_1 \cup r_2$;
  if $b \neq b_i(r_1) + b_i(r_2)$ **then return** *false*;
return *true*;

Algorithm 11: Removal criterion based on decreasing number of cavities

Input: A dart d of a 3D topological map.
Result: `true` iff $c_2(d)$ must be removed.

$r_1 \leftarrow region(d)$; $r_2 \leftarrow region(\beta_3(d))$;
$C \leftarrow$ compute incrementally the number of cavities of $r_1 \cup r_2$;
return $C \leq \#cavities(r_1) + \#cavities(r_2)$;

time complexity equal to the time complexity of the incremental computation algorithms which are linearly proportional to the number of darts used to represent the cells of the border of the regions.

The incremental computation method implies that, if two regions r_1 and r_2 are merged during the symbolic merging, we update in constant time the Betti numbers. This is achieved by storing the values computed during the evaluation of the topological criteria and using them to update the values of the symbolic region $r_1 \cup r_2$ if the two regions are merged.

12.5 Experimental Results

We present some segmentation results obtained using the segmentation algorithm proposed in the previous section. First, we present some results obtained using different criteria with the generic segmentation algorithm and topological maps. Then, we introduce topological constraints inside the segmentation algorithm to obtain partitions that constraint the number of cavities or tunnels.

12.5.1 Generic Criteria

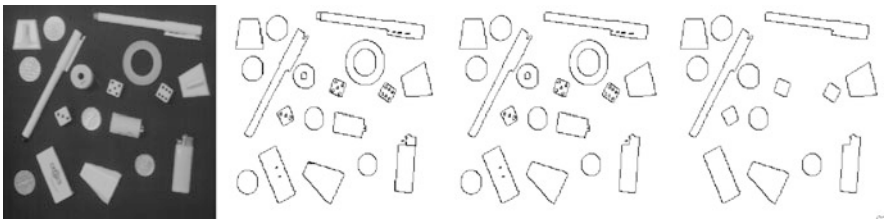
We show some segmentation results obtained using the different criteria presented in Sect. 12.3.3. Figure 12.14 shows the segmentation of a classical 2D image using the

Fig. 12.14 Sample partitions using the range and contrast criteria. (**a**) Original "fishing boat" image; (**b**) Partition of the "fishing boat" image using the range criterion (range = 100); (**c**) Partition of the "fishing boat" image using the range criterion with a random weight function (range = 100); (**d**) Slice of a simulated body PET scan (luminosity and contrast have been edited); (**e**) Slice of the simulated PET 3D image segmented using the contrast criterion with the removal of small regions ($k = 2500$, minimum size for regions 500 voxels)

Fig. 12.15 Partitions obtained from a 2D image using the small region criterion. (**a**) Original image; (**b**) Partition obtained using the range criterion with a threshold of 50; there are many small regions; (**c**) Partition obtained from the previous partition, by using the segmentation algorithm with the small region threshold set to 25. There are no more small regions of size less than 25 in the partition

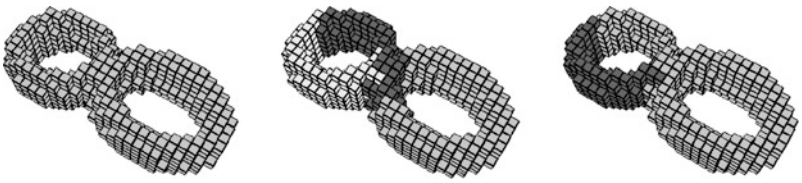
intensity range criterion, and the segmentation of simulated body positron emission tomography (PET) 3D image using the contrast criterion. We also apply removal of small regions to obtain a cleaner partition. In Fig. 12.14(c), we process edges in a random order, that is without a meaningful weight function. The obtained partition has more regions and some regions, which have a similar intensity (like background regions) cannot be merged together due to the intensity range constraint.

The small region criterion allows removing small regions that do not appear to be relevant to the obtained partition. An example of the use of such a criterion is show in Fig. 12.15 where the initial partition Fig. 12.15(b), obtained using the range criterion, contains lots of small regions. Using the small region removal criterion in conjunction with a weight function that orders the edges by the intensity difference between the two adjacent regions, we obtain the partition presented Fig. 12.15(c) where the regions are larger.



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

Fig. 12.16 Segmentation of a 2D image using a topological constraint to limit the number of cavities. (**a**) Original image; (**b**) Initial partition obtained with a range criterion (threshold set to 100); (**c**) Partition obtained from (**b**) if the number of cavities is constant with a range threshold set to 150; (**d**) Partition obtained from (**b**) if the number of cavities decreases with a range threshold set to 150



(a) (b) (c)

Fig. 12.17 Segmentation of a 3D object using a topological constraint to limit the number of tunnels. (**a**) Original partition of the object without a topological constraint on the number of tunnels per region, the region has two tunnels; (**b**) Partition obtained if the number of tunnels is minimized: the same object is represented by three regions without a tunnel; (**c**) Partition obtained if the allowed number of tunnels does not excess one: the same object is represented by two regions, one having a tunnel while the other has none

12.5.2 Constraint on Betti Numbers

In Fig. 12.16, we show the impact of topological constraints applied to a segmentation of the 2D image presented in Fig. 12.16(a). We use the range criterion to obtain an initial segmentation shown in Fig. 12.16(b). In this segmentation there are regions with cavities. Starting with the initial partition, increasing the range threshold, and constraining the number of cavities by a constant, we obtain the segmentation proposed in Fig. 12.16(c). If the cavity count is allowed to decrease from the cavity count in the initial partition, the result is slightly different, as shown in Fig. 12.16(d). Some objects disappeared as the regions representing them are merged with the background. Moreover, some small cavities, like dots in the dice, are removed while they are preserved if the number of cavities remains constant.

Figure 12.17 illustrates the results obtained by a segmentation algorithm that controls the number of tunnels in objects. Without topological constraint, the segmentation using the intensity range criteria produces a 2-torus (a region with two tunnels) as seen in Fig. 12.17(a). If no tunnels are allowed, the segmentation result uses three regions to represent the same object, none of them having a tunnel

(Fig. 12.17(b)). Finally, if one tunnel per region is allowed, the segmentation result uses two regions to represent the object: one of them has a tunnel and the other one has none. This is due to the ordering of the border of the segmentation algorithm that merges the regions (Fig. 12.17(c)).

12.6 Open Problems and Discussion

In this chapter, we presented an efficient and generic image segmentation algorithm based on topological maps. The algorithm is generic because it is defined in any dimension, and it is controlled by two functions that change its behavior. In order to specify the segmentation process, we define several segmentation criteria using different kind of information associated with the regions or the cells represented by topological maps. A method of computation of topological invariants, the Betti numbers, based on the complete representation of cells and the relationship between them in topological maps, is presented. We introduce new topological criteria using Betti numbers that control the number of cavities and tunnels in regions of an image partition. This process is an example of topological control during image segmentation. We provide some experiments of 2D and 3D image segmentation using different criteria to enlighten the use of topological maps for image segmentation.

There are several open problems and possible extensions for image segmentation using topological maps. From a theoretical point of view, the main issue to tackle involves the definition of topological maps in any dimension. The problem is to construct a minimal combinatorial map that describes an nD image partition into regions. In 2D and 3D, we propose a constructive definition that is proved in [8, 9]. Starting from a combinatorial map describing all the elements of the image, the model is refined by successive simplifications using removal operations. The process produces the minimal map representing the partition of the image. The principal difficulty is to guarantee the preservation of all topological information during the cell removals while obtaining the minimal number of cells in the combinatorial map. One can directly extend the constructive definition in higher dimensions since combinatorial maps and removal operations are defined in any dimension. The constraints required to preserve all the topological information may also be stated for any dimension, but guaranteeing that the obtained combinatorial map is minimal, remains an open problem. This is not an issue for the particular topic of image segmentation since the generic algorithm does also work in non-minimal combinatorial maps. However, this is not satisfying since two different combinatorial maps may describe the same partition.

Another theoretical problem is the computation of Betti numbers and their use as criteria for image segmentation in any dimension. Currently, we only have computation algorithms in case of 2D and 3D topological maps. This issue is related to the problem of combinatorial balls' characterization. Actually, Betti numbers are defined for a cellular complex where each cell is homeomorphic to a topological ball. As there is no combinatorial way to recognize if a cell is a topological ball,

there is no easy way to compute Betti numbers. This is also an open problem in computational topology. In 2D and 3D, we solve this problem by using the Euler characteristic of each border of regions and the implicit cells, but this solution is not available in higher dimensions. An extension of this problem is the computation of other topological invariants such as the homology group generators for regions in a topological map.

Future works that do not raise theoretical questions might include the development of alternative segmentation methods using topological maps. In this work, we presented a bottom-up segmentation approach that uses the removal operation to group regions and obtain the final segmentation. Using the split operation, which divides a region in smaller regions, one can develop a top-down approach. Starting with the whole image in a unique region, the final partition is obtained by successive splits of regions. New criteria based on region properties should be developed to support the top-down approach as the method raises new issues. One aspect of such criteria is to define how regions are split from the geometrical and topological point of views. An example of topological criterion we envision is the split of a region based on the number of tunnels or cavities. An initial idea is to split a region to remove a tunnel or a cavity from that region: this criteria might gain from the computation of the homology group generators to guide the split operation. Another idea is to split a region with multiple cavities such as each of the resulting regions has exactly one cavity.

To broaden the use of topological maps in computer vision related works, we also want to tackle real world image segmentation issues. In preliminary experiments, we obtained interesting results regarding the use of Betti numbers as criteria for image segmentation. Using image segmentation with topological maps, in conjunction with topological criteria like constraints on Betti numbers, could ease the development of image segmentation tools. This should be fully demonstrated by applying image segmentation with topological maps to real use cases and compare results to existing approaches. Experts from the field should be brought along to define the goals of the segmentation tool and evaluate the results. New criteria, based on different parameters should also be introduced and mixed with the existing ones to produce better results.

References

1. Bertrand, Y., Damiand, G., Fiorio, C.: Topological map: minimal encoding of 3d segmented images. In: Proc. of 3rd Workshop on Graph-Based Representation in Pattern Recognition, Ischia, Italy, pp. 64–73 (2001)
2. Braquelaire, J.P., Brun, L.: Image segmentation with topological maps and inter-pixel representation. J. Vis. Commun. Image Represent. **9**(1), 62–79 (1998)
3. Braquelaire, J.P., Desbarats, P., Domenger, J.P.: 3D split and merge with 3-maps. In: Proc. of International Workshop on Graph Based Representations, IAPR-TC15, Ischia, Italy, pp. 32–43 (2001)
4. Braquelaire, J.P., Desbarats, P., Domenger, J.P., Wüthrich, C.A.: A topological structuring for aggregates of 3d discrete objects. In: Proc. of International Workshop on Graph Based Representations, IAPR-TC15, Austria, pp. 193–202 (1999)

5. Braquelaire, J.P., Domenger, J.P.: Representation of segmented images with discrete geometric maps. Image Vis. Comput. **17**(10), 715–735 (1999)
6. Braquelaire, J.P., Guitton, P.: A model for image structuration. In: Proceedings of Computer Graphics International, pp. 426–435 (1988)
7. Brun, L.: Segmentation d'images couleur à base topologique. Thèse de doctorat, Université Bordeaux 1 (1996)
8. Damiand, G.: Topological model for 3d image representation: definition and incremental extraction algorithm. Comput. Vis. Image Underst. **109**(3), 260–289 (2008)
9. Damiand, G., Bertrand, Y., Fiorio, C.: Topological model for two-dimensional image representation: definition and optimal extraction algorithm. Comput. Vis. Image Underst. **93**(2), 111–154 (2004)
10. Damiand, G., Dexet-Guiard, M., Lienhardt, P., Andres, E.: Removal and contraction operations to define combinatorial pyramids: application to the design of a spatial modeler. Image Vis. Comput. **23**(2), 259–269 (2005)
11. Damiand, G., Lienhardt, P.: Removal and contraction for n-dimensional generalized maps. In: Proc. of International Conference on Discrete Geometry for Computer Imagery, Naples, Italy. Lecture Notes in Computer Science, vol. 2886, pp. 408–419. Springer, Berlin (2003)
12. Damiand, G., Peltier, S., Fuchs, L., Lienhardt, P.: Topological map: an efficient tool to compute incrementally topological features on 3d images. In: Reulke, R., Eckardt, U., Flach, B., Knauer, U., Polthier, K. (eds.) IWCIA. Lecture Notes in Computer Science, vol. 4040, pp. 1–15. Springer, Berlin (2006)
13. Damiand, G., Resch, P.: Topological map based algorithms for 3d image segmentation. In: Proc. of 10th International Conference on Discrete Geometry for Computer Imagery, Bordeaux, France. Lecture Notes in Computer Science, vol. 2301, pp. 220–231. Springer, Berlin (2002). http://dx.doi.org/10.1007/3-540-45986-3_20
14. Domenger, J.P.: Conception et implémentation du noyau graphique d'un environnement 2D1/2 d'édition d'images discrètes. Thèse de doctorat, Université Bordeaux 1 (1992)
15. Dupas, A., Damiand, G.: First results for 3d image segmentation with topological map. In: Proc. of International Conference on Discrete Geometry for Computer Imagery, Lyon, France. Lecture Notes in Computer Science, vol. 4992, pp. 507–518. Springer, Berlin (2008)
16. Dupas, A., Damiand, G.: Region merging with topological control. Discrete Appl. Math. **157**(16), 3435–3446 (2009)
17. Felzenszwalb, P.F., Huttenlocher, D.P.: Image segmentation using local variation. In: Proc. of Computer Vision and Pattern Recognition, pp. 98–104 (1998)
18. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient graph-based image segmentation. Int. J. Comput. Vis. **59**(2), 167–181 (2004)
19. Fiorio, C.: A topologically consistent representation for image analysis: the frontiers topological graph. In: Proc. of International Conference on Discrete Geometry for Computer Imagery, Lyon, France. Lecture Notes in Computer Science, vol. 1176, pp. 151–162 (1996)
20. Fourey, S., Brun, L.: A first step toward combinatorial pyramids in n-D spaces. In: Proc. of Graph-Based Representations in Pattern Recognition, Venice, Italy. Lecture Notes in Computer Sciences, vol. 5534, pp. 304–313. Springer, Berlin (2009)
21. Fourey, S., Brun, L.: Efficient encoding of n-d combinatorial pyramids. In: Proc. of International Conference on Pattern Recognition, ICPR '10, pp. 1036–1039. IEEE Comput. Soc., Los Alamitos (2010)
22. Freixenet, J., Muoz, X., Raba, D., Mart, J., Cuf, X.: Yet another survey on image segmentation: region and boundary information integration. In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds.) Computer Vision ECCV 2002. Lecture Notes in Computer Science, vol. 2352, pp. 21–25. Springer, Berlin (2002)
23. Fu, K.S., Mui, J.K.: A survey on image segmentation. Pattern Recognit. **13**(1), 3–16 (1981)
24. Itwm, F.: Survey of 3d image segmentation methods. Fraunhofer Institut Techno- und Wirtschaftsmathematik ITWM 123, Kaiserslautern, Germany (2007)
25. Khalimsky, E., Kopperman, R., Meyer, P.R.: Computer graphics and connected topologies on finite ordered sets. Topol. Appl. **36**, 1–17 (1990)

26. Köthe, U.: Xpmaps and topological segmentation—a unified approach to finite topologies in the plane. In: Proc. of International Conference Discrete Geometry for Computer Imagery, Bordeaux, France. Lecture Notes in Computer Science, vol. 2301, pp. 22–33 (2002)
27. Kovalevsky, V.A.: Finite topology as applied to image analysis. Comput. Vis. Graph. Image Process. **46**, 141–161 (1989)
28. Kropatsch, W.G., Macho, H.: Finding the structure of connected components using dual irregular pyramids. In: Proc. of International Conference on Discrete Geometry for Computer Imagery, pp. 147–158 (1995)
29. Lienhardt, P.: Topological models for boundary representation: a comparison with n-dimensional generalized maps. Comput. Aided Des. **23**(1), 59–82 (1991)
30. Lienhardt, P.: N-dimensional generalized combinatorial maps and cellular quasi-manifolds. Int. J. Comput. Geom. Appl. **4**(3), 275–324 (1994)
31. Munkres, J.R.: Elements of Algebraic Topology. Addison-Wesley, Reading (1984)
32. Pal, N.R., Pal, S.K.: A review on image segmentation techniques. Pattern Recognit. **26**(9), 1277–1294 (1993)
33. Rosenfeld, A.: Adjacency in digital pictures. Inf. Control **26**(1), 24–33 (1974)
34. Tarjan, R.: Efficiency of a good but not linear set union algorithm. J. ACM **22**(2), 215–225 (1975).
35. Willersinn, D., Kropatsch, W.: Dual graph contraction for irregular pyramids. In: Proceedings of 12th IAPR International Conference on Signal Processing, Jerusalem, Israel, vol. 3, pp. 251–256 (1994)
36. Zhang, Y.J.: Evaluation and comparison of different segmentation algorithms. Pattern Recognit. Lett. **18**(10), 963–974 (1997)

Chapter 13
Multigrid Convergence of Discrete Geometric Estimators

David Coeurjolly, Jacques-Olivier Lachaud, and Tristan Roussillon

Abstract The analysis of digital shapes requires tools to determine accurately their geometric characteristics. Their boundary is by essence discrete and is seen by continuous geometry as a jagged continuous curve, either straight or not derivable. *Discrete geometric estimators* are specific tools designed to determine geometric information on such curves. We present here global geometric estimators of area, length, moments, as well as local geometric estimators of tangent and curvature. We further study their *multigrid convergence*, a fundamental property which guarantees that the estimation tends toward the exact one as the sampling resolution gets finer and finer. Known theorems on multigrid convergence are summarized. A representative subsets of estimators have been implemented in a common framework (the library `DGtal`), and have been experimentally evaluated for several classes of shapes. Thus, the interested users have all the information for choosing the best adapted estimators to their applications, and readily dispose of an efficient implementation.

13.1 Introduction

Since early developments in image processing and image understanding, many tools have been developed in order to quantify the geometry of a digital shape. Such digital shapes can be defined for instance either from a segmentation process as a subset of image pixels sharing the same colorimetric information, or as the result of the digitization of a continuous object.

D. Coeurjolly (✉) · T. Roussillon
LIRIS, UMR CNRS 5205, Université de Lyon, 69676 Villeurbanne, France
e-mail: david.coeurjolly@liris.cnrs.fr

T. Roussillon
e-mail: tristan.roussillon@liris.cnrs.fr

J.-O. Lachaud
LAMA, UMR CNRS 5127, University of Savoie, 73376 Le Bourget du Lac, France
e-mail: jacques-olivier.lachaud@univ-savoie.fr

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4_13,

In many applications, it is important to have a geometrical quantification or description from measurements which are invariant under a specific class of transforms (rotation, translation, and scaling) or which preserve important geometrical features (characteristic points, local convexity, etc.). In this context, we usually consider differential or integral quantities evaluated either on the digital shape or its boundary. Beside such type of quantification, we can distinguish two classes of geometrical descriptors. The first class corresponds to global descriptors which associate a global numerical quantity with each shape. In this class, we have arc length or perimeter estimators of digital shape boundaries, but also integral quantities such as geometrical moments approximated on the digital shape. The second class contains local estimators which usually associate a numerical quantity with each point of the shape. For example, curvature or normal vector estimators at boundary points belong to this class.

When defining an algorithm that evaluates such descriptors on digital shape, so called estimator, the evaluation of such estimator accuracy may be challenging. In the literature, several approaches have been proposed. The first one is application driven and consists in validating the estimators within a complete shape description pipeline. For instance, one can evaluate a curvature estimator in a global characteristic points estimation framework of contours.

One can also evaluate the accuracy of the estimator in terms of expected properties. For instance, we can evaluate the stability of a curvature estimator when rotations of input shapes are given.

A more formal evaluation process consists in comparing the estimated quantities with exact Euclidean values on a family of continuous shapes in a multigrid asymptotic framework. More precisely, let $\mathbb{X}$ be a family of compact simply connected subsets of $\mathbb{R}^2$ with continuous curvature fields. We denote by $D(X,h)$ the Gauss digitization of $X \in \mathbb{X}$ with grid step h, seen as a union of pixels of side h in $\mathbb{R}^2$. For sake of clarity, we shorten in the sequel $D(X,h)$ into D and denote its complementary by $\bar{D}$. Moreover, let us assume that D contains at least one pixel, i.e. $|D| \geq 1$.

In this multigrid framework, comparing the estimated quantity to the expected Euclidean one when h tends to zero is called the *multigrid convergence* analysis of an estimator [35]. Indeed, at a given resolution, infinitely many shapes have the same digitization, which hampers the objective comparison of estimators. For estimators of local geometric quantities like tangent or curvature, few results exist. We may quote some convergence results for tangent estimators [21, 42, 51]. And there is no correct convergence results for curvature as far as we know.

In this chapter, we use this multigrid comparison framework in order to review and evaluate existing local and global estimators on digital shapes. A important contribution is to have considered a large set of estimators in a unique technical framework: the DGtal open-source library [18]. DGtal is a generic open source library for Digital Geometry programming for which the main objective is to structure different developments from the digital geometry and topology community. For the purpose of this chapter, we use DGtal to represent multigrid digital objects and shapes, to define the geometric estimators and we provide ways to compare estimated values to expected Euclidean ones.

The chapter is organized as follows: Sect. 13.2 focuses on global estimators (area, moments and arc length) and Sects. 13.3, 13.4, and 13.5 are devoted to local estimators, tangent, and curvature, respectively. In all cases, each section starts with a formal definition of the multigrid convergence of an estimator. In Sect. 13.6, we discuss on implementation details of both the estimator and the comparative evaluation framework.

13.2 Global Estimators

13.2.1 Multigrid Convergence for Global Estimators

Multigrid convergence is an interesting way of relating digital and Euclidean geometries. The idea is to ask for discrete geometric estimations to converge toward the corresponding Euclidean quantity when considering finer and finer shape digitizations (here, Gauss digitization). The following definition is taken from Definition 2.10 of [35].

Definition 1 (*Multigrid convergence for global geometric quantities*) A discrete geometric estimator $\hat{E}$ of some geometric quantity E is *multigrid convergent* for a family of shapes $\mathbb{X}$ and a digitization process D iff for all shape $X \in \mathbb{X}$, there exists a grid step $h_X > 0$ such that the estimate $\hat{E}(D(X,h),h)$ is defined for all $0 < h < h_X$ and

$$\left|\hat{E}\big(D(X,h),h\big) - E(X)\right| \leq \tau_X(h),$$

where $\tau_X : \mathbb{R}^+ \to \mathbb{R}^+$ is with null limit at 0. This function is the *speed of convergence* of the estimator.

The convergence of most estimators depends on the family of shapes in the Euclidean plane that is considered. We therefore introduce several standard families that will be used to define the range of validity of multigrid convergence theorems. A curve is said to be C^n if it has continuous n-th order derivatives.

- The family of all finite convex shapes in the Euclidean plane is denoted with $\mathbb{X}^C$.
- The family of convex sets whose boundary is a C^n arc with positive curvature everywhere is denoted by $\mathbb{X}^{n\text{-}SC}$.
- The family of all planar piecewise n-smooth convex set is denoted with $\mathbb{X}^{n\text{-}PW\text{-}SC}$. These sets are convex sets whose boundary consists of a finite number of C^n arcs with positive curvature everywhere except at arc endpoints. Clearly $\mathbb{X}^{n\text{-}SC} \subsetneq \mathbb{X}^{n\text{-}PW\text{-}SC}$.

For experiments, we will use shapes that are representative of these families. Several representative shapes digitized at different scales are illustrated in Fig. 13.1. They will be used for the upcoming experiments on global and local geometric estimators.

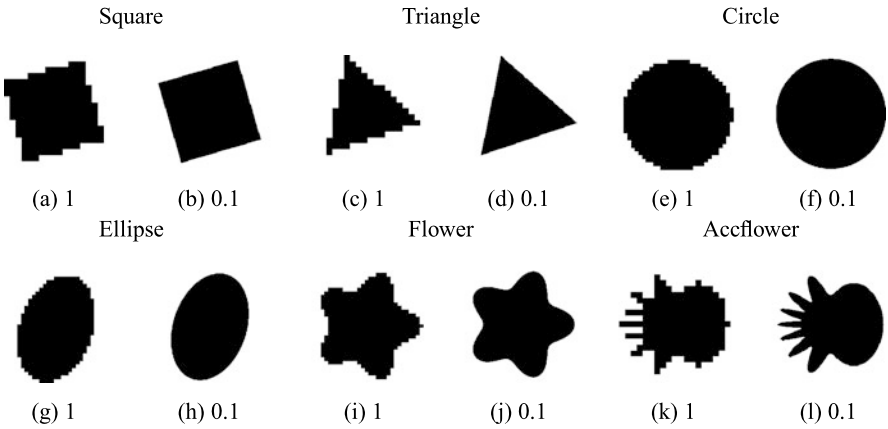


Fig. 13.1 Digitization at two different grid steps ($h = 1$ or $h = 0.1$) of tests shapes: (**a**–**d**) the *square* and *triangle* are in $\mathbb{X}^C$; the *circle* (**e**, **f**) and the *ellipse* (**g**, **h**) belongs to $\mathbb{X}^{3\text{-SC}}$; the *flower* (**i**, **j**), and the "*accflower*" (**k**, **l**) are in $\mathbb{X}^{3\text{-PW-SC}}$. All shapes have diameter 20

13.2.2 Area and Moments

Designing a multigrid convergent estimator of the area is fairly simple. We define the *area estimator by counting* $\hat{A}$ as

$$\hat{A}(Y,h) = h^2 \sum_{(i,j)\in Y} 1, \tag{13.1}$$

where Y is an arbitrary digital shape and h the grid step. This estimator just counts the number of h-grid square in Y and normalizes the result with the area of each grid square.

As reported in [36], Gauss and Dirichlet knew already that this area estimator was multigrid convergent for finite convex shape ($\mathbb{X}^C$) with a speed $O(l \cdot h)$, where l is the shape perimeter. Huxley [28] improves the bound to $O(h^{\frac{15}{11}}(\log \frac{1}{h})^{\frac{47}{22}})$ for the family $\mathbb{X}^{3\text{-}PW\text{-}SC}$. This simple estimator has thus super-linear convergence for a rather wide class of shapes.

Klette and Žunić [36] follow the idea of (13.1) to design the *discrete* (p,q)-*moment estimator* $\widehat{m_{p,q}}$, for integers $p, q \geq 0$, as follows:

$$\widehat{m_{p,q}}(Y,h) = h^{2+p+q} \sum_{(i,j)\in Y} i^p \cdot j^q. \tag{13.2}$$

These estimators approximate the (p,q)-moments of a shape X, which are defined as $m_{p,q}(X) = \iint_X x^p y^q dxdy$. Their speed of convergence is summed up in Table 13.1. In a similar way, central moments may be approximated. We refer the reader to [36] for further details. Note that moments can be used to determine for instance the center of gravity or the orientation of a shape. Furthermore, several rotational invariant quantities can be obtained as combination of (p,q)-moments. For

instance, Zernike and Legendre moments widely used in many 2D and 3D shape indexing and retrieval are linear combination of (p, q)-moments [56, 67]. Hence, convergence results on (p, q)-moments lead to convergence of Zernike and Legendre moments as well.

The previous estimators require to visit all points of the digital object, and not only its boundary. The computational complexity of these estimators is thus $O(1/h^2)$. However, a discrete variant of Green theorem allows to compute these quantities by simply visiting the shape boundary, thus reducing the computational complexity to $O(1/h)$ for convex shapes. See Lien [47] for a generic discrete Green theorem framework and Brlek et al. for a digital geometry application [3].

13.2.3 Perimeter and Length Estimators

It is more complex to estimate the perimeter of a digital shape. Indeed, enumerating the number of grid steps of the digital shape boundary does not lead to a reliable perimeter estimator. It is called the *naive perimeter estimator* $\hat{L}^{\text{naive}}$ and is defined as

$$\hat{L}^{\text{naive}}(Y, h) = h \sum_{\sigma \in \partial Y} 1. \tag{13.3}$$

This estimator overestimates the shape perimeter. Indeed, it is clear that it always returns the perimeter of the axis-aligned bounding box of the shape.

Therefore first approaches to length estimation tried to assign different weights to different local configurations so as to be more precise. The Rosen-Proffitt estimator [59] and BLUE (best linear unbiased) estimator [19] belong to this category. However it was proved in Tajine and Daurat [66] that all these approaches can never achieve multigrid convergence, whatever the (finite) number of configurations taken into account.

More complex approaches are required to achieve convergence. We list below several of them, which are also experimentally compared (see Fig. 13.2). Most of them are not only valid for perimeter estimation but also for curve length estimation.

- The *DSS length estimator* $\hat{L}^{\text{DSS}}$, proposed by Kovalevsky and Fuchs [37], relies on a greedy decomposition of the input digital contour into Digital Straight Segments (DSS). It starts from a point, then finds the longest DSS starting from that point. The end point of the DSS defines a new starting point. The process is repeated till the whole contour has been visited. The DSS end points form a polygonal line. The length or perimeter of the digital contour is then simply defined as Euclidean length of this polygonal line.
- The *MLP length estimator* $\hat{L}^{\text{MLP}}$, proposed by Sloboda et al. [65], also relies on a polygonal approximation of the digital contour. For a given simple digital shape, the Minimum Length Polygon (MLP) is indeed the shortest Euclidean curve which separates the interior pixel centers from the exterior pixel centers. The length is then defined as the perimeter of this curve. Several linear-time algorithms for computing the MLP are available [60, 63].

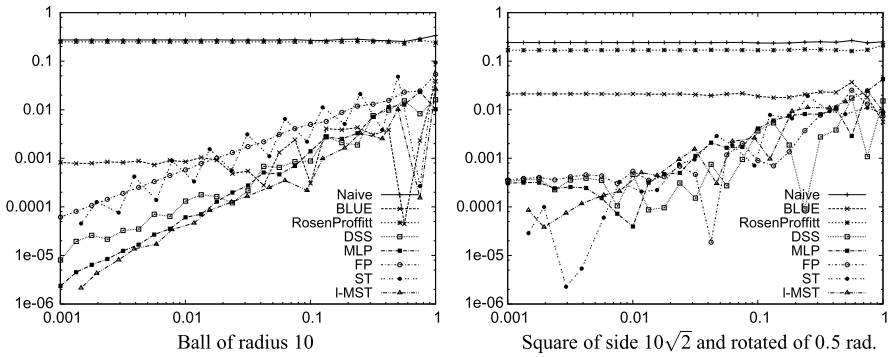


Fig. 13.2 Absolute relative error for several length and perimeter estimators. It is clear that the naive length estimator does not converge. The other estimators (DSS, MLP, FP, ST, λ-MST) present a multigrid convergence. Note that experimentally the convergence speed for DSS, FP, and ST on the ball is $O(h)$ while MLP and λ-MST achieve a better bound of $O(h^{\frac{4}{3}})$. However, on the boundary of a shape with linear parts, the convergence speed is $O(h)$ for all the estimators except the naive one

- The *FP length estimator* $\hat{L}^{\mathrm{FP}}$, proposed in [63], relies on yet another polygonal approximation of the digital contour. One can see it is a translated version of the MLP, where convex turns are translated outwards and concave turns are translated inwards by half-unit diagonal vectors. The advantage is that the polygon vertices form a subset of the grid points of the input contour.
- Another approach to local length estimation and thus perimeter estimation is to integrate the tangent estimation along the curve [8, 11] (see also the next section on tangent estimation). The *ST length estimator* $\hat{L}^{\mathrm{ST}}$ is based on the symmetric tangent while the *λ-MST length estimator* $\hat{L}^{\lambda\text{-MST}}$ is based on the λ-convex combination of maximal segments [42]. More precisely, if a grid edge σ has direction vector $\mathbf{t}(\sigma)$ and estimated unit tangent vector $\hat{\mathbf{T}}(\sigma)$, these two estimators are defined as:

$$\begin{aligned}\hat{L}^{\mathrm{ST}}(Y,h) &= h\sum_{\sigma\in\partial Y}\hat{\mathbf{T}}^{\mathrm{ST}}(\sigma)\cdot\mathbf{t}(\sigma),\\ \hat{L}^{\lambda\text{-MST}}(Y,h) &= h\sum_{\sigma\in\partial Y}\hat{\mathbf{T}}^{\lambda\text{-MST}}(\sigma)\cdot\mathbf{t}(\sigma).\end{aligned} \tag{13.4}$$

Some experimental evaluation of the multigrid convergence has been carried out for these estimators and is illustrated in Fig. 13.2. It appears that the perimeter of shapes with rectilinear boundaries is accurately estimated with any of the presented length estimators but for the naive one. However, for shapes with sufficiently smooth boundaries and positive curvature, the MLP and λ-MST have super-linear convergence and should be preferred. Note finally that only DSS, MLP, and λ-MST have proved multigrid convergence, but the found bounds are not necessarily tight. In Fig. 13.3, we present computational time for estimators implemented in DGtal release 0.4 (Naive, BLUE, RosenProffitt, DSS, MLP, FP). Convergence results for ST and λ-MST have been obtained from ImaGene library [29]. In these graphs, we can

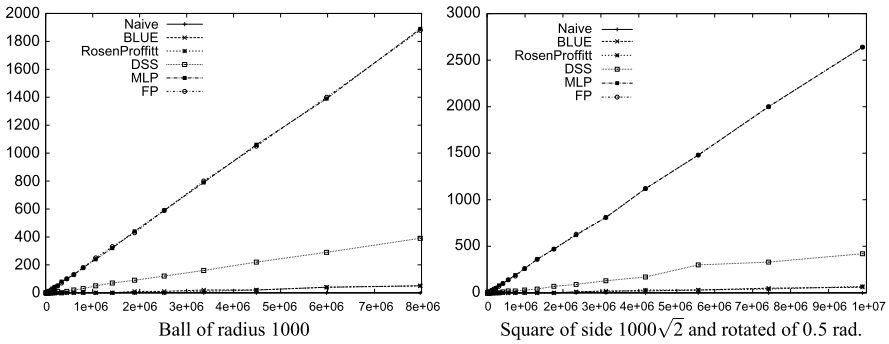


Fig. 13.3 Computation time for length estimators implemented in DGtal (Naive, BLUE, Rosen-Proffitt, DSS, MLP, FP). For the sake of clarity, *abscissa* corresponds to the size of the contour in number of grid points. *Ordinate* corresponds to timings in millisecond (Intel Xeon 2.27 GHz, DGtal 0.4)

observe the linear computational cost of all estimators with respect to the size of the contours. As expected, local estimators outperform the other ones, but a DSS based estimator is a good compromise between efficiency and theoretical multigrid convergence.

13.2.4 Summary

Table 13.1 summarizes multigrid convergence results for estimators of global geometric quantities. It appears that some theoretical bounds are not tight and that some others are yet to be proved.

13.3 Local Estimators

13.3.1 Multigrid Convergence for Local Estimators

Tangent direction, normal vector, curvature are local geometric quantities along the shape boundary. Thus, each of them is a function of the shape boundary. However, the contour of the shape digitization does not define the same domain. Therefore we cannot directly compare the true geometric function with the estimated geometric function. We provide below a definition of multigrid convergence for discrete local estimators. It is neither a parametric definition as in [21] nor a point-wise definition as the standard multigrid convergence reported in [35]. Furthermore, for the sake of simplicity, there is no direct mapping between the contour and its digitized counterpart as proposed in [39]. It is a geometric definition, stating that any digital point sufficiently close to the point of interest has its estimated geometric quantity

Table 13.1 Known multigrid convergence for several estimators of global geometric quantities

Quantity	Estimator	Shape family	Convergence speed		References
			Upper bound	Observed	
area	$\hat{A}$	$\mathbb{X}^{C}$	$O(h)$		Gauss, Dirichlet
area	$\hat{A}$	$\mathbb{X}^{3\text{-}PW\text{-}SC}$	$O(h^{\frac{15}{11}+\epsilon})$		[28]
moments	$\widehat{m_{p,q}}$	$\mathbb{X}^{3\text{-}PW\text{-}SC}$	$O(h)$		[36]
moments	$\widehat{m_{p,q}}$	$\mathbb{X}^{3\text{-}SC}$	$O(h^{\frac{15}{11}+\epsilon})$		[36]
length	$\hat{L}^{\mathrm{DSS}}$	Convex polygons	$\approx 4.5h$		[37]
length	$\hat{L}^{\mathrm{DSS}}$	$\mathbb{X}^{3\text{-}PW\text{-}SC}$	(unknown)	$O(h)$	[37]
length	ϵ-sausage	Convex polygons	$\approx 5.844h$		[2]
length	$\hat{L}^{\mathrm{ST}}$	$\mathbb{X}^{C}$	(unknown)	$O(h)$	[11]
length	$\hat{L}^{\mathrm{FP}}$	$\mathbb{X}^{C}$	(unknown)	$O(h)$	[63]
length	$\hat{L}^{\mathrm{MLP}}$	$\mathbb{X}^{C}$	$\approx 8h$		[65]
length	$\hat{L}^{\mathrm{MLP}}$	$\mathbb{X}^{3\text{-}PW\text{-}SC}$	$O(h)$	$O(h^{\frac{4}{3}})$	[65]
length	$\hat{L}^{\lambda\text{-MST}}$	$\mathbb{X}^{3\text{-}PW\text{-}SC}$	$O(h^{\frac{1}{3}})$	$O(h^{\frac{4}{3}})$	[39]

which tends toward the expected local value of the geometric function. This definition of multigrid convergence imposes shapes with continuous geometric fields. Of course, one can afterward relax this constraint by splitting the shape boundary into individual parts where the geometric function is continuous.

Let us recall that $\mathbb{X}$ is some family of shapes in the Euclidean plane. We denote by $D(X,h)$ the Gauss digitization of $X \in \mathbb{X}$ with grid step h. For any x in the topological boundary ∂X of X, let $Q(X,x)$ be some local geometric quantity of ∂X at x. A *discrete local estimator* $\hat{Q}$ is a mapping which associates with any digital contour C, a point $y \in C$ and a grid step h, some value in a vector space (e.g., $\mathbb{R}$ for the curvature). We are now in position to define the multigrid-convergence of this estimator:

Definition 2 The estimator $\hat{Q}$ is *multigrid-convergent* for the family $\mathbb{X}$ if and only if, for any $X \in \mathbb{X}$, there exists a grid step $h_X > 0$ such that the estimate $\hat{Q}(D(X,h),y,h)$ is defined for all $y \in \partial D(X,h)$ with $0 < h < h_X$, and for any $x \in \partial X$,

$$\forall y \in \partial D(X,h) \text{ with } \|y-x\|_1 \le h, \quad \left|\hat{Q}\big(D(X,h),y,h\big) - Q(X,x)\right| \le \tau_{X,x}(h),$$

where $\tau_{X,x} : \mathbb{R}^{+*} \to \mathbb{R}^{+}$ has null limit at 0. This function defines the *speed of convergence* of $\hat{Q}$ toward Q at point x of ∂X. The convergence is *uniform* for X when every $\tau_{X,x}$ is bounded from above by a function τ_X independent of $x \in \partial X$ with null limit at 0.

It is worthy to note that, for sufficiently regular shapes (par(r)-regular shapes [43]), there exists a grid step below which the boundary of the shape digitization has same topology as the shape boundary ([39], Theorem B.5). Furthermore, these two

boundaries are very close. Indeed, there exists a grid step below which for any $x \in X$ there is a $y \in \partial D(X,h)$ with $\|y - x\|_1 \leq h$ and conversely for any $y \in \partial D(X,h)$, there is a $x \in X$ with $\|y - x\|_1 \leq h$ [39, Lemma B.9].

Therefore the previous definition of multigrid convergence guarantees that the estimated local quantity converges toward the true local geometric quantity everywhere along the shape boundary.

13.3.2 Methodology for Experimental Evaluation

When multigrid convergence theorems have been established, we will reference them and indicate the known convergence rate. We nevertheless carry out an experimental evaluation of many different estimators for two reasons: (1) few convergence theorems exist for local estimators, and (2) practical error bounds at finite scale are also important for the end-user.

In the next sections, we apply the following methodology for analyzing estimators:

1. Test shapes. We use the shapes of Fig. 13.1 for the experiments. They are representative of the different shape families that we are studying. Indeed, shapes composed of linear parts, smooth parts and corners, arise naturally in image analysis. When the tangent field is not continuous (square, triangle), only the average error is significant.
2. Graphs of estimations with respect to ground truth. We display the graphs of the estimated values for different estimators (functions $\hat{Q}$) and the expected graph (function Q).
3. Error measures for decreasing h. We study the following measures:

$$\epsilon_{abs}(X,y,h) = \left|Q\big(X,x(y)\big) - \hat{Q}\big(D(X,h),y,h\big)\right| \tag{13.5}$$

or (when a vector)
$$\epsilon_{abs}(X,y,h) = \left|det\big(\mathbf{Q}\big(X,x(y)\big),\hat{\mathbf{Q}}\big(D(X,h),y,h\big)\big)\right| \tag{13.6}$$

$$\epsilon_{rel}(X,y,h) = \frac{\epsilon_{abs}(X,y,h)}{|Q(X,x(y))|} \tag{13.7}$$

$$\overline{\epsilon}_{abs}(X,h) = \frac{1}{\#D(X,h)} \sum_{y \in D(X,h)} \epsilon_{abs}(X,y,h) \tag{13.8}$$

$$\overline{\epsilon}_{rel}(X,h) = \frac{1}{\#D(X,h)} \sum_{y \in D(X,h)} \epsilon_{rel}(X,y,h) \tag{13.9}$$

   Here $x(\cdot)$ is a mapping associating to each digitized point a point on the shape boundary that is close enough ($\|y - x(y)\|_1 \leq h$). The same mapping is used for all estimators.
4. When known, computational complexities for computing estimators will be given. Otherwise, for fair comparisons, we only measure computation times for estimators implemented in the DGtal library (see Sect. 13.6).

This methodology allows us to evaluate experimentally the accuracy and multigrid convergence properties of discrete local geometric estimators. Section 13.4 studies tangent estimators and Sect. 13.5 studies curvature estimators. Their implementation in a common framework is discussed in Sect. 13.6.

13.4 Tangent

The aim of tangent estimators is to determine what is locally the shape boundary direction. For curves $\gamma(s)$ (at least C^1) defined as functions of a curvilinear abscissa s, the tangent vector is defined as $\frac{d\gamma}{ds}$, which is a unit vector. The tangent direction $\phi(s)$ is defined as the angle between this unit vector and the x-axis.

13.4.1 Tangent Estimators

Given a digital contour and a digital point, tangent estimators return a unit vector. For easier view, it is also possible to plot the angle of the tangent vector w.r.t. the x-axis. It is clear that the grid edge direction (see arrows in Fig. 13.9d for an illustration) is a very bad tangent estimator, since on any shape in $\mathbb{X}^{1\text{-SC}}$ it will have points with ϵ_{abs} or ϵ_{rel} close to $\frac{\pi}{2}$.

More complex approaches are necessary. Digital tangent estimators have been thoroughly compared in [41, 42]. They have been compared to continuous approaches in [15, 16]. We describe below some representative tangent estimators, which will be compared experimentally.

- A first natural approach is to use a local least-square fit of a polynomial [5, 46]. These techniques define a fixed window-size q. Around the point of interest they use $2q+1$ samples which are used to find the polynomial of given degree that best fit these data in the least-square sense. We focus here on low-degree polynomials. The *LR tangent estimator* $\hat{\mathbf{T}}^{LR\text{-}q}$ is the linear-regression with the window size q. The *ICIPF tangent estimator* $\hat{\mathbf{T}}^{ICIPF\text{-}q}$ is the implicit parabola fitting of window size q, made independently on each coordinate. They were found to be representative of that kind of methods [15, 16].
- A second approach is to see the digital contour as a discrete signal $(x[t], y[t])$ and to convolve it with a Gaussian derivative of a given kernel σ. This is very similar to the binomial convolution approach of [21, 23, 51]. Therefore, we choose the $H1\text{-}0GD$ *tangent estimator* $\hat{\mathbf{T}}^{H1\text{-}0GD}$ [16], which defines locally the window size as the longest maximal digital straight containing the point. A slight variant is proved to be multigrid convergent in $O(h^{\frac{1}{3}})$ for smooth shapes in $\mathbb{X}^{3\text{-}SC}$, while its experimental convergent rate is excellent [16] for smooth shapes.
- The *MCMS tangent estimator* $\hat{\mathbf{T}}^{\text{MCMS}}$ defines the tangent as the direction of the most-centered maximal digital straight segment containing the point of interest [22]. It is proved to be uniformly multigrid convergent in $O(h^{\frac{1}{3}})$ in [39, 42].

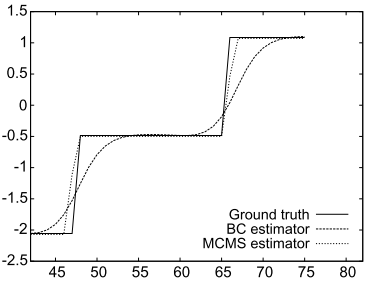
- The λ-*MST tangent estimator* $\hat{\mathbf{T}}^{\lambda\text{-MST}}$ is based on the λ-convex combination of the direction of the maximal digital straight segments containing the point of interest [42]. The function λ is a mapping governing the way these directions are combined. We use here a simple triangle function f, such that $f(0) = 0$, $f(1) = 0$, $f(0.5) = 1$. It is proved to be uniformly multigrid convergent in $O(h^{\frac{1}{3}})$ in [39, 42].
- The *BC tangent estimator* $\hat{\mathbf{T}}^{BC}$ considers the digital contour as a discrete signal $(x[t], y[t])$ and convolves it with discrete binomials and a discrete difference operator, so as to mimic the convolution by a Gaussian derivative [21, 51]. We use the suggested mask size of $d.h^{-\frac{4}{3}}$, where d is the continuous shape diameter. It is proved to be uniformly multigrid convergent in $O(h^{\frac{2}{3}})$ in [51].
- The *MATAS tangent estimator* $\hat{\mathbf{T}}^{MATAS}$ is an adaptation of the median filter commonly used in image processing [53]. If (P_i) are the vertices of the grid contour, this method consists in choosing the median orientation among the following $2q$ vectors centered on P_i: $(\mathbf{P}_{i-q}\mathbf{P}_i, \ldots, \mathbf{P}_{i-1}\mathbf{P}_i, \mathbf{P}_i\mathbf{P}_{i+1}, \ldots, \mathbf{P}_i\mathbf{P}_{i+q})$.

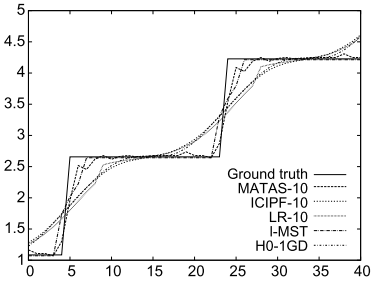
13.4.2 Experimental Evaluation

We have run these estimators on two representative shapes (the square is representative of $\mathbb{X}^C$, the ellipse is representative of $\mathbb{X}^{3\text{-SC}}$) at different steps (coarse $h = 1$, medium $h = 0.1$). Results are displayed in Fig. 13.4. Only MCMS and λ-MST detect perfectly straight parts and corners. Others tend to smooth around corners, the amount of smoothing being dependent on the (chosen) size of the window. Furthermore, LR and ICIPF oscillate around the correct value on straight parts. It is more difficult to tell which estimator is the best along the boundary of smooth curved parts. MCMS produces a staircase-like function but keeps the convexity of the shape. MATAS, LR and ICIPF may also oscillate and create false concavities. BC and λ-MST follow nicely the ground truth function. Overall, λ-MST seems to be the most versatile and accurate at these resolutions. Experiments on other shapes confirm the presented behaviors of these estimators.

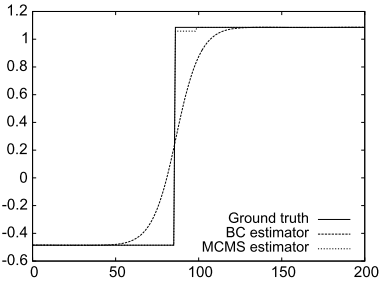
We now turn ourselves to the asymptotic behavior of these estimators, namely their possible multigrid convergence. We focus on the average absolute error of the tangent vector, i.e. $\overline{\epsilon}_{abs}(X, h)$ (see (13.8)). The error plots displayed in Fig. 13.5 show that tangent estimators with fixed window size are not multigrid convergent. This is the case of LR, ICIPF and MATAS estimators. Interestingly, but not surprisingly, small window sizes bring better precision at low scale while greater window sizes bring better precision at fine scale. This is clearly the problem of such estimators: they require a user to choose the best possible scale according to the input data.

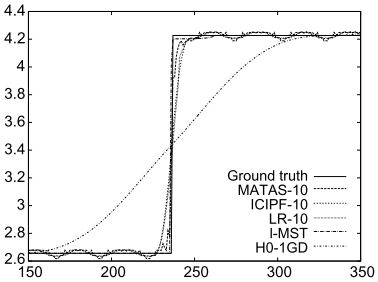
If we look at the other estimators (H1-0GD, MCMS, λ-MST, BC), the window size is automatically determined, either globally by $d.h^{-\frac{4}{3}}$ for BC estimator, or locally by maximal digital straight segments for the remaining three. All these four estimators are experimentally multigrid convergent for most of the considered shapes.



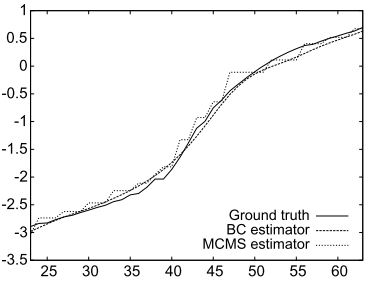


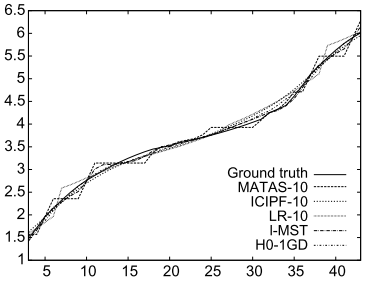
(a) The shape is the square digitized at $h = 1$.



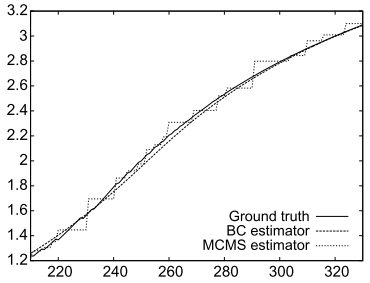


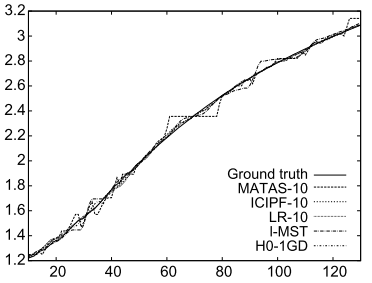
(b) The shape is the square digitized at $h = 0.1$.





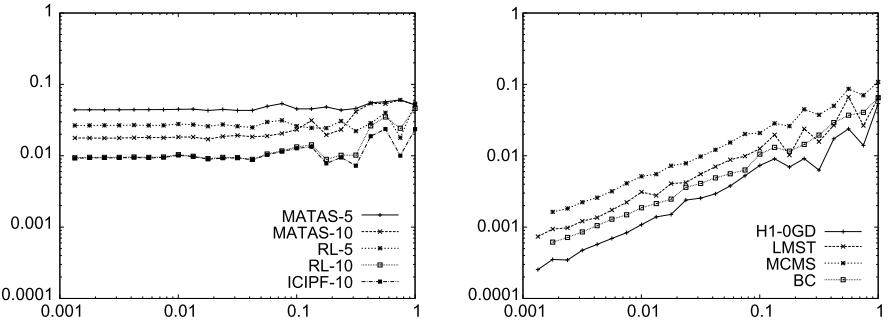
(c) The shape is the ellipse digitized at $h = 1$.



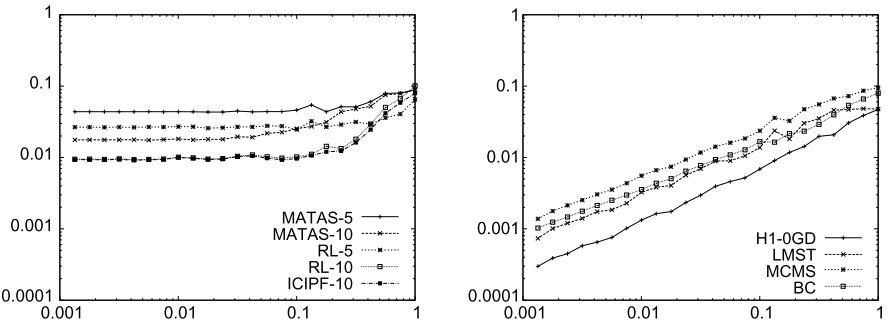


(d) The shape is the ellipse digitized at $h = 0.1$.

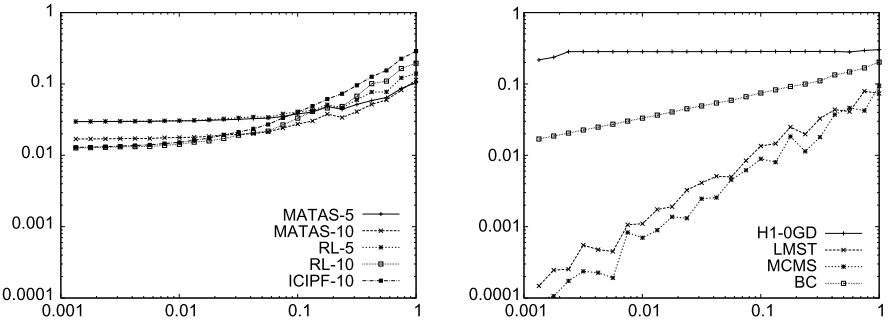
Fig. 13.4 Plots of tangent directions as a function of the grid edge index for several shapes and several tangent estimators. For each row, the shape and the digitization step is given. *Left column*: BC and MCMS estimators. *Right column*: MATAS estimator with window 10, ICIPF estimator with window 0, LR estimator with window 10, λ-MST estimator, H1-0GD estimator. Note that for a clearer view, only a representative part of the plot is displayed, and that due to implementation, grid edges indices of the *first column* are different from the ones of the *second column*



(a) The shape is the *circle* (in $\mathbb{X}^{\infty-SC}$) digitized for steps $h \in [0.001, 1]$.



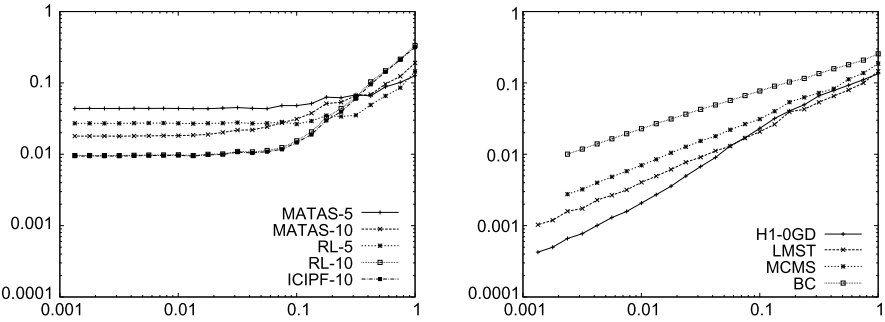
(b) The shape is the *ellipse* (in $\mathbb{X}^{\infty-SC}$) digitized for steps $h \in [0.001, 1]$.



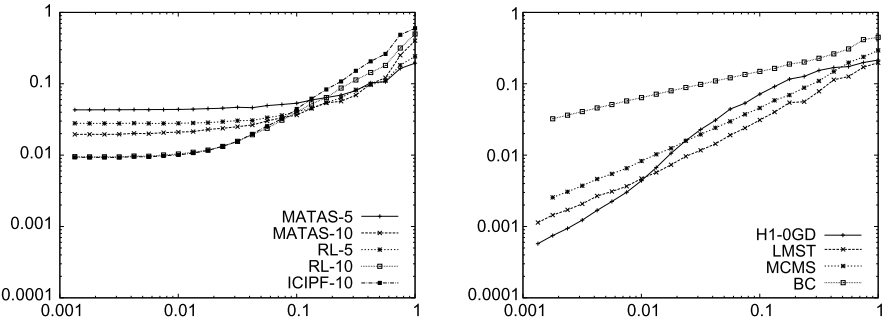
(c) The shape is the *triangle* (in $\mathbb{X}^{C}$) digitized for steps $h \in [0.001, 1]$.

Fig. 13.5 Plots in log-scale of the average absolute errors of tangent vectors as a function of the grid step for several shapes and several tangent estimators. For each row, the shape and the digitization step is given. *Left column*: MATAS estimator with windows 5 and 10, LR estimator with windows 5 and 10, ICIPF estimator with window 10. *Right column*: BC estimator, MCMS estimator, λ-MST estimator, H1-0GD estimator

However, their convergence speed may vary greatly. BC is good for smooth convex shapes, but has low convergence speed on shapes with inflexion points or linear parts. H1-0GD is excellent on smooth shapes for a fine enough sampling, but is not



(d) The shape is the *flower* (in $\mathbb{X}^{\infty-PW-SC}$) digitized for steps $h \in [0.001, 1]$.



(e) The shape is the *accflower* (in $\mathbb{X}^{\infty-PW-SC}$) digitized for steps $h \in [0.001, 1]$.

Fig. 13.5 (Continued)

Table 13.2 Known multigrid convergence for several tangent estimators. LR, MATAS, ICIPF are not multigrid convergent

Estimator	Shape family	Convergence speed		References
		Upper bound	Observed	
$\hat{\mathbf{T}}^{\text{BC}}$	Polygons	?	$O(h^{\frac{1}{3}})$	(here)
$\hat{\mathbf{T}}^{\text{BC}}$	$\mathbb{X}^{1\text{-}SC}$	$O(h^{\frac{2}{3}})$	$O(h^{\frac{2}{3}})$	[51]
$\hat{\mathbf{T}}^{\text{BC}}$	$\mathbb{X}^{1\text{-}PW\text{-}SC}$	?	$O(h^{\frac{1}{3}})$	(here)
$\hat{\mathbf{T}}^{\lambda\text{-MST}}$ and $\hat{\mathbf{T}}^{\text{MCMS}}$	Polygons	$O(h)$	$O(h)$	[39]
$\hat{\mathbf{T}}^{\lambda\text{-MST}}$ and $\hat{\mathbf{T}}^{\text{MCMS}}$	$\mathbb{X}^{1\text{-}PW\text{-}SC}$	?	$O(h^{\frac{2}{3}})$	[16]
$\hat{\mathbf{T}}^{\lambda\text{-MST}}$ and $\hat{\mathbf{T}}^{\text{MCMS}}$	$\mathbb{X}^{3\text{-}SC}$	$O(h^{\frac{1}{3}})$	$O(h^{\frac{2}{3}})$	[39]
$\hat{\mathbf{T}}^{\text{H1-0GD}}$	Polygons	?	not convergent	(here)
$\hat{\mathbf{T}}^{\text{H1-0GD}}$	$\mathbb{X}^{1\text{-}PW\text{-}SC}$	?	$\approx O(h^{\frac{2.5}{3}})$	[16]
$\hat{\mathbf{T}}^{\text{H1-0GD}}$	$\mathbb{X}^{3\text{-}SC}$	$O(h^{\frac{1}{3}})$	$O(h^{\frac{2.5}{3}})$	[16]

good on shapes with linear parts. MCMS and λ-MST are the most versatile. λ-MST is preferable to get a continuous tangent. Convergence results are summed up in Table 13.2.

13.5 Curvature

For curves $\gamma(s)$ (at least C^2) defined as functions of a curvilinear abscissa s, the curvature κ can be defined in three different, but essentially equivalent ways:

(i) norm of the second derivative: $\kappa(s) = |\frac{d^2\gamma}{ds^2}|$,
(ii) derivative of the tangent orientation: if $\phi(s)$ is the angle between the tangent and a given line, $\kappa(s) = \frac{d\phi}{ds}$,
(iii) inverse of the osculating circle radius: if $\rho(s)$ is the radius of the osculating circle, $\kappa(s) = 1/\rho(s)$.

Given a grid point of a digital contour, curvature estimators are expected to return a value in $\mathbb{R}$ close to the curvature of the underlying shape. Estimating the curvature by finite differences over the two neighbors of a given grid point returns either a positive (resp. negative) high value in convex (resp. concave) corners ($\pm 1/\sqrt{2}$) or a null one in runs and is thus a very bad solution.

Many curvature estimators have been proposed in the literature to cope with this problem. They are roughly based on one of the three above-mentioned definitions as it has been noticed in [27, 70].

In methods (i) and (ii), derivatives are often approximated from the convolution of either the tangent orientation [22, 68, 70] (i), or the digital contour viewed as a discrete signal $(x[t], y[t])$ [21, 23, 51] (ii). They can also be computed from some polynomials of a given degree locally fitted to the digital contour [27, 52].

Tangents and osculating circles used in methods (ii) and (iii) often rely on fitting techniques, either in a continuous setting (least square line or arc fitting [70]), or in a discrete setting to limit the arithmetic effects: digital straight segments [22, 68], digital level layers (extension to polynomials of higher degrees) [25, 61], approximation of the osculating circle with digital straight segments [12, 13, 27], digital circular arcs [62].

In most approaches, a user-given window or smoothing parameter is used so as to remove the jaggedness of digital contours and to make it continuous [21–24, 49, 51, 68, 70]. Few curvature estimators do not require an external parameter and we chose to focus on these methods.

13.5.1 Curvature Estimators

The curvature estimators that do not require any parameter either rely on discrete primitives such as digital straight segment (DSS), digital circular arc (DCA), or on global optimization such as the Global Minimum Curvature estimator [32].

In this section, we compare the following curvature estimators:

- The *MS estimator* $\hat{\kappa}^{\mathrm{MS}}$ [12] used only the length of maximal DSS to estimate the radius of the osculating circle.

  The method relies on the assumption that maximal DSS of the digitization of a Euclidean circle behave like chords of height h and length $\Theta(h^{\frac{1}{2}})$. Maximal

DSS are however almost always tighter and the length of maximal DSS has been proved to be in $O(h^{\frac{1}{2}})$ but in $\Theta(h^{\frac{2}{3}})$ on average [17].

- The *CC estimator* $\hat{\boldsymbol{\kappa}}^{\mathrm{CC}}$ [13] (HK2005 in [27]), associates with any grid point of a digital contour, the curvature of the circumscribed circle of a triangle defined by the extremities of its two digital half-tangents.

  It has been proved to be convergent if the length of maximal DSS is in $\Omega(h^{\frac{1}{2}})$ [8]. This condition is however not fulfilled because the length of maximal DSS has been proved later to be in $\Theta(h^{\frac{2}{3}})$ on average [17].
- Another estimation of the osculating circles can be obtained from the maximal DCA along the digital contour [62]. The *MDCA estimator* $\hat{\boldsymbol{\kappa}}^{\mathrm{MDCA}}$ is the piecewise constant function that associates with any grid point of a digital contour the curvature value of the most centered maximal DCA.

  Although this approach seems quite natural, it has been proposed only recently due to the lack of available implementation of on-line DCA recognition algorithms [10, 38, 64]. It is a natural extension of the tangent estimator based on the most centered maximal DSS (*MCMS tangent estimator* in Sect. 13.4) to the osculating circle estimation problem. As a result, the λ-*MST tangent estimator* used to improve this tangent estimator may probably improve this curvature estimator.

  The *MDCA estimator* has been proved to be convergent [62] if the length of the maximal DCA along the digital contour of the digitization of strictly convex shapes with continuous curvature field is in $\Omega(h^{\frac{1}{2}})$, which is observed on average.
- The *GMC curvature estimator* $\hat{\boldsymbol{\kappa}}^{\mathrm{GMC}}$ [32] computes the curvature of the shape that minimizes its squared curvature among all the Euclidean shapes that may be digitized as a digital set close to the input.

  The first step consists in computing the whole set of maximal DSS. This processing provides tangent and arc length estimations (Sect. 13.2.3 and Sect. 13.4) used to bound the set of valid shapes in the tangential space $(\phi(s), s)$. In this tangential space, the polygonal line that minimizes its length is then computed to approximate the shape of piecewise constant curvature that minimizes its squared curvature.

  The minimization is performed by an iterative numerical technique that stops when the difference between the squared curvature of the last two solution shapes is less than a small quantity, set to 1.10^{-8} in what follows.
- Finally, for comparisons, we also introduce the *BC curvature estimator* $\hat{\boldsymbol{\kappa}}^{\mathrm{BC}}$ [21, 51], which is computed from derivative estimations, obtained through a discrete difference operator applied on the digital contour viewed as a discrete signal $(x[t], y[t])$ convolved by a binomial kernel of a given size. The mask size is an input parameter that is not easy to determine, but following [51], it has been set to $d.h^{-\frac{4}{3}}$ where d is the diameter of the continuous shape.

  The multigrid convergence of the estimation of the first (resp. second) derivative at rate $O(h^{\frac{2}{3}})$ (resp. $O(h^{\frac{4}{9}})$) has been proved in [21, 51].

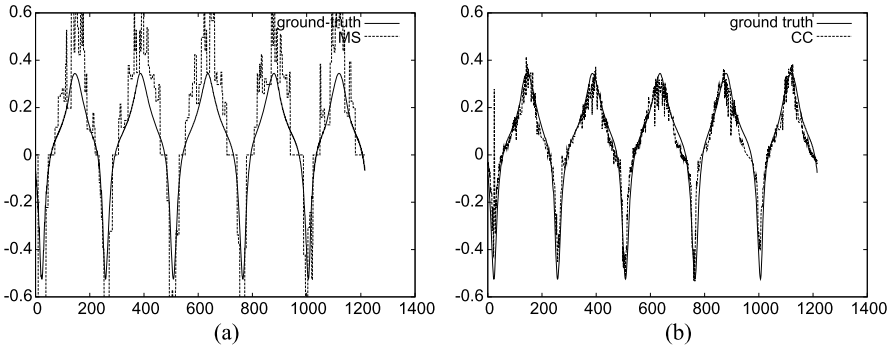


Fig. 13.6 Curvature plots for the flower, digitized with a grid step equal to 0.1 (Fig. 13.1j). *MS estimator* in (**a**) and *CC estimator* in (**b**)

13.5.2 Experimental Evaluation

We first plot the curvature values provided by the *MS estimator* (resp. *CC estimator*) in Fig. 13.6a (resp. Fig. 13.6b) when applied to the digital contour of Fig. 13.1j.

Because a maximal DSS is a good neighborhood to check the local convexity and concavity of a digital curve [63], the *MS estimator* provides positive curvature values in convex parts, negative curvature values in concave parts and null curvature values around inflection points (Fig. 13.6a). However, the *MS estimator* systematically over-estimates the true curvature values in the convex parts and underestimates the true curvature values in the concave parts. The deviation is important at low resolution and increases as the grid step h decreases. This is clearly a bad (and not convergent) estimator.

The *CC estimator* does not respect the convex and concave parts of the digital contour (see the peak of positive curvature value near the starting point in Fig. 13.6b), it oscillates a lot but gives correct results on average at low resolution.

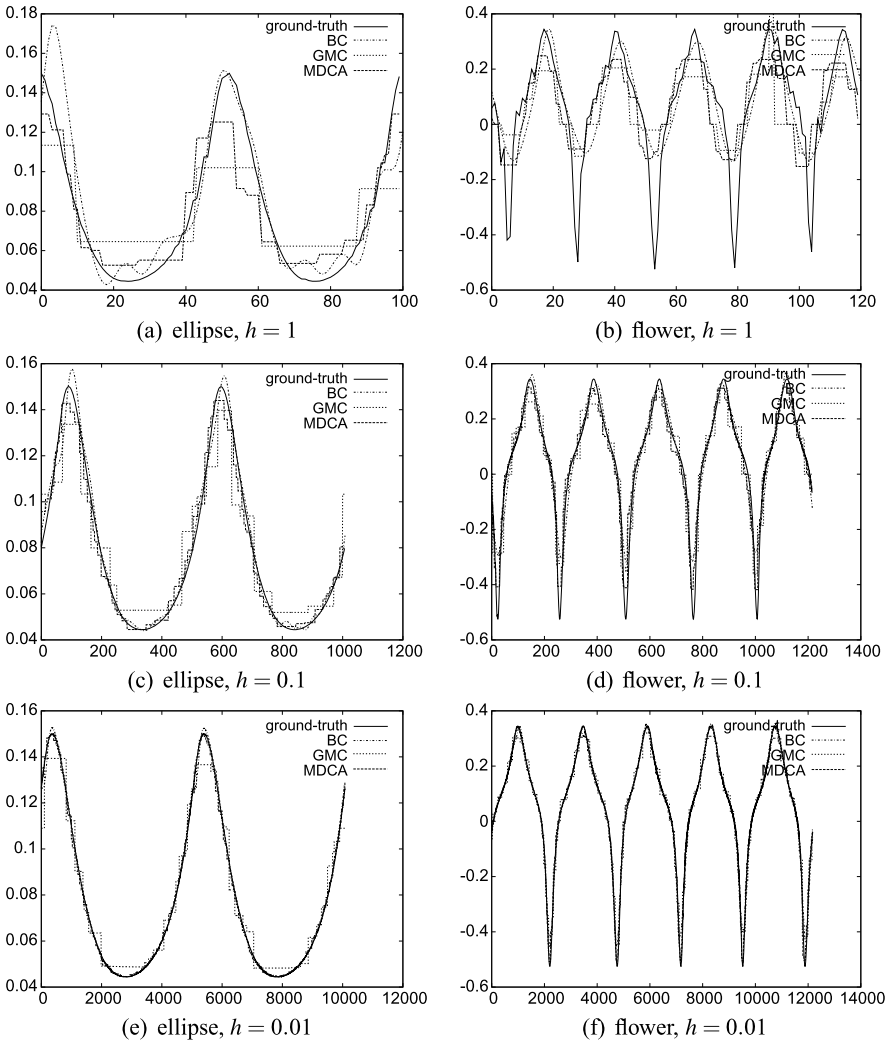
In Fig. 13.7, we compare the curvature plots derived from the *MDCA, GMC, BC estimators* to the ground-truth.

The visual deviation between the estimated graphs and the ground-truth graph reflects the average absolute error. For either estimator, the curvature estimations are more accurate for the ellipse than for the flower. For either shape, the curvature values obtained from any estimator get closer to the ground-truth (Fig. 13.7) and the absolute error decreases as the grid step h decreases.

For the ellipse and the flower, at grid step $h = 0.01$, the *MDCA estimator* and the *BC estimator* are better than the *GMC estimator*. In Fig. 13.7, their graphs are hardly confounded with the ground-truth graph.

In Fig. 13.8, the average absolute error has been plotted against the grid step h. The *CC estimator* is not convergent and has the highest errors. However, the other estimators ($\hat{\boldsymbol{\kappa}}^{\mathrm{MDCA}}$, $\hat{\boldsymbol{\kappa}}^{\mathrm{GMC}}$, $\hat{\boldsymbol{\kappa}}^{\mathrm{BC}}$) appear to be multigrid convergent.

We experimentally observed that the *MDCA estimator* has low absolute errors that decrease as the grid step h decreases. The convergence speed on average of the



(a) ellipse, $h = 1$ (b) flower, $h = 1$

(c) ellipse, $h = 0.1$ (d) flower, $h = 0.1$

(e) ellipse, $h = 0.01$ (f) flower, $h = 0.01$

Fig. 13.7 Curvature plots for two shapes digitized at three different resolutions, computed from *MDCA, GMC, BC estimators*

MDCA estimator is $O(h^{0.5})$ (even maybe $O(h^{\alpha})$ with $\alpha > 0.5$) for the ellipse and the flower (Fig. 13.8b and c) but $O(h^2)$ for the circle.

The *GMC estimator* and the *BC estimator* have usually higher errors. The *BC estimator* has lower errors than the *MDCA estimator* for the ellipse and for the flower at low resolution (when the grid step is decreasing from 1 to 0.3). The *GMC estimator* has however always higher errors than the *MDCA estimator*.

The *GMC estimator* and the *BC estimator* have usually a slower convergence speed:

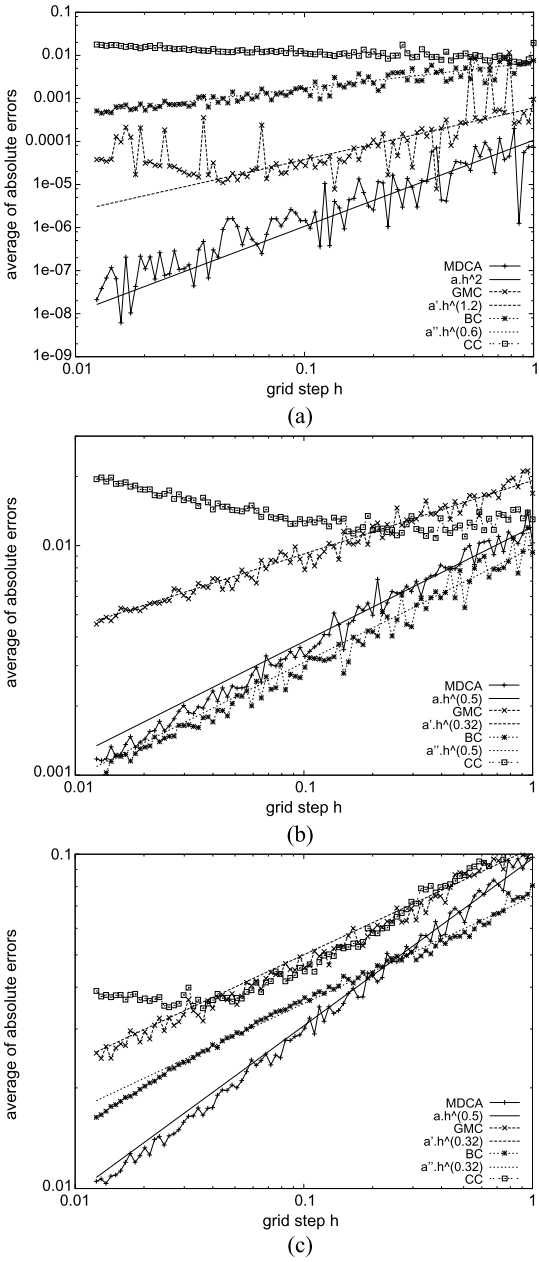


Fig. 13.8 The average absolute error has been plotted against the grid step h for the digitization of a circle in (**a**), an ellipse in (**b**) and a flower in (**c**)

- respectively $O(h^{1.2})$ and $O(h^{0.6})$ for the circle (note that the *GMC estimator* is sensitive to the stop criterion of its optimization process when errors are small),
- respectively $O(h^{0.32})$ and $O(h^{0.5})$ for the ellipse,

- $O(h^{0.32})$ for the flower (but note that the error graph of the *BC estimator* is not straight and further experiments should be done at smaller grid steps to get the convergence speed).

Eventually the *MDCA, GMC, BC estimators* appear to be experimentally multigrid-convergent, but there is no correct theoretical convergence results for curvature estimation as far as we know, contrary to the case of tangent estimation (Sect. 13.4).

13.6 Implementation

In this section, we discuss about implementation details of both the geometrical estimators presented in the previous sections, and the experimental evaluation framework. All the estimators described in this chapter have been implemented in DGtal [18]. DGtal is an open-source C++ library focusing on the implementation of digital geometry objects and concepts. For short, it allows to represent images and objects in n-dimensional digital spaces equipped with both geometrical and topological tools.

In the context of this chapter, we will only consider the representation and the analysis of shape in dimension 2. As discussed in the introduction, the input digital object can be obtained either from an explicit description, from a segmentation process of an image (iso-level, ...), or as the digitization $D(X, h)$ of a continuous shape $X \in \mathbb{X}$. For the first two cases, DGtal provides mechanisms to construct such digital sets either explicitly or from a contour tracking process. For the last case, DGtal implements various implicit and parametric continuous shapes for which some global and local geometrical quantities are known. All such shape implementations are model of a concept of `CEuclideanShapes`[1] (see Fig. 13.1 for an illustration of DGtal Euclidean shapes). A digital object is thus obtained from a `GaussDigitizer` parametrized by a model of `CEuclideanShapes` and a grid step h. `CEuclideanShapes` models will be crucial for multigrid convergence analysis.

As discussed above and whatever the way the input digital object is specified, we need to access to its geometrical information in various ways:

- As a sequence of grid points, subset of $\mathbb{Z}^2$, e.g. for area and moment descriptors;
- As a representation of its boundary, e.g. for tangent or curvature estimators.

In the latter case, several options exist to define and represent a shape contour. Most of the options depend on the underlying topological model (Kong's like digital topology or cellular topology). Furthermore, depending on the algorithm used to perform the analysis, one may prefer a sequence of chain codes, a sequence

[1] DGtal uses a generic programming paradigm based on concepts and models of concepts. If a structure name starts from a capital "C", we describe a concept.

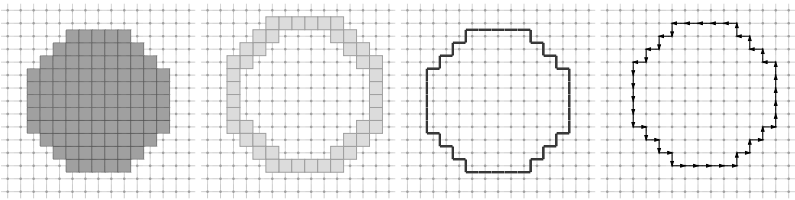


Fig. 13.9 Different representations of a Euclidean shape digitization: as a set of pixels (**a**), as a sequence of 4-connected pixels (**b**), as a sequence of 1-cell or *linels* (**c**), as a sequence of grid point displacements (**d**)

of linels or a sequence of 4-connected grid points to describe the contour (cf. Fig. 13.9).

To obtain a generic and extensible implementation of contour based estimators, we have defined a `GridCurve` structure constructed upon a topological cellular model which aims to provide several facets of a shape contour. More precisely, given the result of the contour tracking process, it provides mechanism (`Ranges` and `Iterators` on `Ranges`) to process the boundary sequence either as a set of grid points or a set of linels. Hence, a local geometrical estimator on contour, or more precisely a model of `CLocalGeometricalEstimator`, have an interface containing at least the two following methods:

- `void init(double h, ConstIterator & begin, ConstIterator & end, ...)`: initialize the geometrical estimator with grid step h on a contour defined between iterators `begin` and `end`;
- `Quantity eval( ConstIterator & it)`: evaluate the estimator at the position `it` of the contour and return a `Quantity`.

In our framework, the type `ConstIterator` is a template parameter chosen in the contour iterator types provided in `GridCurve`.

Similarly, we have a concept of `CGlobalGeometricalEstimator` and models of this concept have an `eval()` method which returns a unique quantity for a shape (or subset of it).

Based on models of `CEuclideanShapes`, we can obtain expected continuous values using `TrueLocalEstimatorOnPoints` and `TrueGlobalEstimatorOnPoints`. Since both expected and estimated values are given by estimators with a consistent interface, it makes the multigrid comparison very simple. Indeed, it allows to design a generic `CompareLocalEstimators` which return a statistic on the difference of two estimator values.

In the following example, we illustrate the multigrid Euclidean shape construction and the comparison of three length estimators (RosenProffitt, DSS and MLP). In this example, we have detailed the overall process: shape construction and digitization, domain and Khalimsky space construction, contour tracking and finally, evaluation of estimators.

```
//....
// h and radius are parameters here
//....
// Types
typedef Ball2D<Space> Shape;
typedef Space::Point Point;
typedef Space::RealPoint RealPoint;
typedef Space::Integer Integer;
typedef HyperRectDomain<Space> Domain;
typedef KhalimskySpaceND<Space::dimension,Integer> KSpace;
typedef KSpace::SCell SCell;
typedef GridCurve<KSpace>::PointsRange PointsRange;
typedef GridCurve<KSpace>::ArrowsRange ArrowsRange;
typedef PointsRange::ConstIterator ConstIteratorOnPoints;

//Euclidean ball
Shape aShape(Point(0,0), radius);

//Gauss Digitization
GaussDigitizer<Space,Shape> dig;
dig.attach( aShape ); // attaches the shape.
dig.init( aShape.getLowerBound(), aShape.getUpperbound(),h);

// The domain size is given by the digitizer according to
// the window and the step.
Domain domain = dig.getDomain();

// Create cellular space
KSpace K;

bool ok = K.init( dig.getLowerBound(), dig.getUpperBound(), true );
if ( ! ok ) {
  std::cerr << "␣"
            << "␣error␣in␣creating␣KSpace." << std::endl;
  return false;
}
try {
  // Extracts shape boundary
  SurfelAdjacency<KSpace::dimension> SAdj( true );
  SCell bel = Surfaces<KSpace>::findABel( K, dig, 10000 );

  // Getting the consecutive surfels of the 2D boundary
  std::vector<Point> points;
  Surfaces<KSpace>::track2DBoundaryPoints( points,
                                           K, SAdj,
                                           dig, bel );
  trace.info() << "#␣tracking..." << endl;

  // Create GridCurve
  GridCurve<KSpace> gridcurve;
  gridcurve.initFromVector( points );
  trace.info() << "#grid␣curve␣created,␣h=" << h << endl;

  //ranges
  ArrowsRange ra = gridcurve.getArrowsRange();
  PointsRange rp = gridcurve.getPointsRange();

  //Three length estimators working on different contour
  //representations

  RosenProffittLocalLengthEstimator< ArrowsRange::ConstIterator >
       RosenProffittlength;
  RosenProffittlength.init(h, ra.begin(), ra.end(), gridcurve.isClosed());

  DSSLengthEstimator< PointsRange::ConstIterator > DSSlength;
  DSSlength.init(h, rp.begin(), rp.end(), gridcurve.isClosed());
```

```
    MLPLengthEstimator< PointsRange::ConstIterator > MLPlength;
    MLPlength.init(h, rp.begin(), rp.end(), gridcurve.isClosed());

    trace.info() << "#Estimations" <<std::endl;
    trace.info() << "#h␣true␣RosenProffitt␣DSS␣MLP␣" <<std::endl;
    trace.info() << h << "␣" << M_PI*2.0
                 << "␣" << RosenProffittlength.eval()
                 << "␣" << DSSlength.eval()
                 << "␣" << MLPlength.eval()
                 << std::endl;
  } catch ( InputException e ) {
    std::cerr << "␣"
              << "␣error␣in␣finding␣a␣bel." << std::endl;
    return false;
  }
```

13.7 Related Problems and Perspectives

13.7.1 Geometric Estimators Along Damaged or Noisy Contours

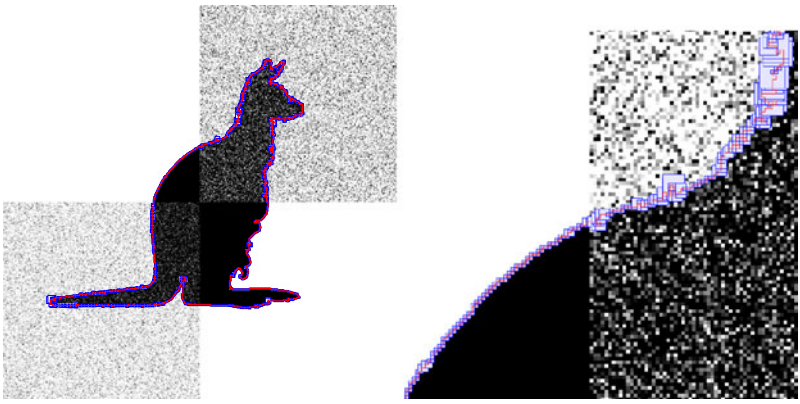
In real applications, images may have been damaged or acquisition devices may induce noise in the image data. Furthermore, binarization algorithms and segmentation algorithms may also damage the boundary of the regions or shapes. These contours are thus not anymore the perfect digitization of "nice" Euclidean shapes (e.g. shapes in $\mathbb{X}^{n\text{-}PW\text{-}SC}$), and have parts that are winding where they should be straight. We will call them hereafter *noisy contours* (an example is given in Fig. 13.10 where a kangaroo shape has been damaged by Gaussian noise).

In the pattern recognition community, a lot of tools have been developed to analyze the geometry of noisy contours, especially to detect corner or dominant points (see for instance [52]). These points are related to curvature information. However, these tools are not designed for estimating quantitatively the geometric characteristics of the contours but rather qualitatively. We only quote here works that give quantitative geometric information on perfect or noisy contours.

A common way to tackle noise along contours is to filter the contour with a smoothing kernel. The size of the kernel given by the user is more or less proportional to the amount of damage along the contour. The BC tangent estimator and the BC curvature estimator are members of this family of techniques [21, 23, 51]. These techniques are efficient when the contour is rather uniformly damaged, but they smooth indifferently noise and high-curvature parts of the contour (like corners).

Approaches based on fitting like the LR or ICIPF estimators [5, 46, 70] are also able to tackle noise along contours, since they tend to find the median or average polynomial that best approaches locally the data. Again, the window size parameter is used to suppress at the same time arithmetic effects and noise artifacts. This parameter is generally set by the user.

In the digital geometry community, a common technique is to use the so-called *blurred segments* [14] instead of digital straight segment. Compared to digital straight segment whose thickness is always less than 1, blurred segments have a



(a) Noise detection (b) Zoom on kangaroo's back

Fig. 13.10 Noisy contour and noise detection along it by the method of [30]. The input image has been damaged by two different Gaussian noises in different regions. The thresholded shape has a boundary which is damaged in these regions (noisy contour in *solid dark gray*). The detector indicates for each contour point what is the local scale at which this part of the contour should be analyzed: the scale or noise level at a point is indicated by the size of the *light gray box* around it. It is worthy to note that the noise level is almost everywhere proportional to the amount of contour degradation

user-given maximal thickness. The noisy parts of contours are thus ignored when using a larger thickness. Several estimators just replace standard segments with blurred segments so as to take into account noisy contours. For instance, the curvature estimator presented in [55] is the noisy variant of the CC estimator. The GMC estimator also uses blurred segments to handle noisy contours. The thickness is generally set by the user.

Note that digital estimators based on digital straight segments (like the H0-1GD, MCMS and λ-MST tangent estimators, or the MDCA curvature estimator) can also be adapted to noisy contours by sub-sampling the input contour. For instance, we can use a 3×3 tile over the input contour so as to remove perturbation no greater than 1 pixel along the contour. However we have yet not run a full set of experiments so as to know if this approach leads to better estimators than the ones quoted in the preceding paragraphs.

Finally, all these techniques require the determination of one or several parameters in order to process at best noisy contours. This scale or smoothing parameter must not be too low otherwise damaged parts are considered high-curvature places or corners, but it must not be too high also in order to preserve features and to have accurate estimates of geometric information.

If a gray-level input noisy image is available, scale space analysis may provide information on the amount of noise [7, 20, 26, 34]. They cannot handle directly noisy contours. For noisy contours, Kerautret and Lachaud [30] have recently proposed a method to automatically detect the meaningful scales of digital contours. It can give locally along the contour what is the amount of noise and the first scale at which the

contour should be analyzed (see Fig. 13.10, and on-line demonstration [33]). Their technique relies on the asymptotic properties of maximal digital straight segments. They have proposed a variant of λ-MST estimator for noisy contours, which uses the noise information given by the meaningful scales [31].

13.7.2 Geometric Estimators in 3D and nD

In higher dimensions, several 2D estimators or frameworks can be extended. However, many open problems exist and beside the fact that a few proofs of multigrid convergence exist, a complete experimental multigrid evaluation of curvature estimators for instance has not been done yet on digital surfaces in $\mathbb{Z}^3$. In this section, we just give a brief overview of existing techniques:

- *Surface area*: to compute the area of a surface in $\mathbb{Z}^3$, a first solution is based on weighted local configurations [48, 69]. The idea is to associate weights with local configurations of surface voxels or surfels. Then, given an object, the surface area is approximated by summing all weights associated with all configurations defined on the object surface. Similarly to the BLUE estimator, weights are given by a statistical analysis to minimize surface area error for a given class of shapes. By deriving results from the length case in dimension 2 [66], surface weighted configuration estimators can never achieve multigrid convergence. In [50], the authors use statistical analysis and integral geometry to design a fast estimation of the surface area. Again, the quality of the estimation is controlled by a parameter (number of line probes). Another option is to generalize the discrete normal vector integration scheme as described in [8, 11]. As detailed in [9], we can prove that if the normal vector estimation is multigrid convergent, then the integration of the vector field leading to the surface area estimation is multigrid convergent as well.
- *Normal vector field computation*: at a point x on a smooth surface, the normal vector at x can be defined as the cross product of first order derivatives at x (tangent) of two curves lying on the surface crossing at x. In a digital context, given a surface element of a cellular representation of a digital surface, two natural digital 4-connected curves can be defined by the intersection of the surface with the two coordinate planes containing the surfel elementary normal vector. Hence, Lenoir et al. suggested to compute the normal vector at a surfel as the cross product of tangent computed on the two 2D digital curves [45]. Following this framework, multigrid convergence can be achieved if the tangent estimator used on the 2D curves is multigrid convergent [9, 39]. The normal vector field of a digital surface in $\mathbb{Z}^n$, for arbitrary n, can be computed with a similar approach [40].
- *Curvature*: For curvature computation on digital surfaces, only few estimators have been proposed in the digital geometry framework. We can cite Lenoir's slice based approach for the mean curvature estimation [44], Gauss map area evaluation for the Gaussian curvature [8], techniques based on integral invariants for both mean and Gaussian curvatures [4, 57, 58]. Integral invariant techniques are definitely relevant in the digital geometry context, even in case of a noisy surface.

However, they require a window parameter which could be difficult to set for a large class of shapes.

In computational geometry, several techniques have been proposed to construct accurate curvature estimators with bounded errors. Usually, bounds are parametrized by a sampling parameter for a given sampling hypothesis. An example of a sampling hypothesis for a smooth surface would be that the sampling density should be proportional to the curvature. In this context, convergence or stability of geometric estimators have been proposed as a function of the sampling parameter. In many situations the sampling theorems used in computational geometry do not match with the specific isotropic behavior of digital surfaces. In [1, 6, 54], estimators are defined on point sets based on Voronoi structures and the error is given in terms of Hausdorff distance (which is consistent with digital surfaces). Investigating the links between computational geometry and digital geometry on this subject is a challenging problem.

13.7.3 Current Bottlenecks and Open Problems

As detailed in the previous sections, we can overview current theoretical bottlenecks in the design of discrete geometric estimators:

- *Stability w.r.t. noise*: Prior detection of the contour local noise level to be used as an estimator parameter, or with estimators which are theoretically robust to a given noise model.
- *Estimators of differential quantity of order 2*: From our point of view, existing curvature estimators are not yet satisfactory since either no proof of multigrid convergence exists, or the convergence is controlled by an external parameter (e.g., window size or Gaussian kernel width). It would be interesting, for instance, to focus on the multigrid behavior of circular arc segment on digital 2D contours. Indeed, many proofs related to the length or the tangent estimation are based on the multigrid behavior of DSS. On digital surfaces and in higher dimension, we think that a better understanding of links between computational and digital geometry results would lead to new results in this domain.

Beside these theoretical bottlenecks, complete experimental multigrid evaluations are now mandatory when designing a new discrete estimator. With the help of both a theoretical methodology (multigrid shape database and error measures) and open-source libraries (`ImaGene` [29] or `DGtal` [18]), we expect to have a complete and stable experimental framework. An important future work would be to continue the implementation of existing estimators with comparative studies. In dimension 3, main bottlenecks are related to efficiency and computational costs. Indeed, in many Material sciences or Medical imaging applications, we may have to analyze digital shapes whose size achieves up to 2048^3. In the implementation of 3D estimators, several theoretical and technical problems have to be addressed, such as out-of-core techniques, hierarchical data representation and adaptive algorithms, and others.

Acknowledgements This work was partially funded by project KIDICO (ANR-2010-BLAN-0205) of the French Research Agency (ANR).

References

1. Amenta, N., Kil, Y.: Defining point-set surfaces. In: ACM SIGGRAPH, vol. 23, p. 270. ACM, New York (2004)
2. Asano, T., Kawamura, Y., Klette, R., Obokata, K.: Minimum-length polygons in approximation sausages. In: 4th International Workshop on Visual Form. Lecture Notes in Computer Science, vol. 2059, pp. 103–112. Springer, Berlin (2001)
3. Brlek, S., Labelle, G., Lacasse, A.: The discrete green theorem and some applications in discrete geometry. Theor. Comput. Sci. **346**(2–3), 200–225 (2005)
4. Bullard, J.W., Garboczi, E.J., Carter, W.C., Fullet, E.R.: Numerical methods for computing interfacial mean curvature. Comput. Mater. Sci. **4**, 103–116 (1995)
5. Cazals, F., Pouget, M.: Estimating differential quantities using polynomial fitting of osculating jets. Comput. Aided Geom. Des. **22**, 121–146 (2005)
6. Chazal, F., Cohen-Steiner, D., Lieutier, A., Thibert, B.: Stability of Curvature Measures (2008)
7. Chen, K.: Adaptive smoothing via contextual and local discontinuities. IEEE Trans. Pattern Anal. Mach. Intell. **27**(10), 1552–1566 (2005)
8. Coeurjolly, D.: Algorithmique et géométrie pour la caractérisation des courbes et des surfaces. Ph.D. thesis, Université Lyon 2 (2002)
9. Coeurjolly, D., Flin, F., Teytaud, O., Tougne, L.: Multigrid convergence and surface area estimation. In: Theoretical Foundations of Computer Vision "Geometry, Morphology, and Computational Imaging". Lecture Notes in Computer Science, vol. 2616, pp. 101–119. Springer, Berlin (2003)
10. Coeurjolly, D., Gérard, Y., Reveillès, J.P., Tougne, L.: An elementary algorithm for digital arc segmentation. Discrete Appl. Math. **139**(1–3), 31–50 (2004)
11. Coeurjolly, D., Klette, R.: A comparative evaluation of length estimators of digital curves. IEEE Trans. Pattern Anal. Mach. Intell. **26**(2), 252–258 (2004)
12. Coeurjolly, D., Miguet, S., Tougne, L.: Discrete curvature based on osculating circle estimation. In: 4th International Workshop on Visual Form. Lecture Notes in Computer Science, vol. 2059, pp. 303–312 (2001)
13. Coeurjolly, D., Svensson, S.: Estimation of curvature along curves with application to fibres in 3d images of paper. In: 13th Scandinavian Conference on Image Analysis. Lecture Notes in Computer Science, vol. 2749, pp. 247–254 (2003)
14. Debled-Rennesson, I., Feschet, F., Rouyer-Degli, J.: Optimal blurred segments decomposition of noisy shapes in linear times. Comput. Graph. **30**, 30–36 (2006)
15. de Vieilleville, F., Lachaud, J.O.: Experimental comparison of continuous and discrete tangent estimators along digital curves. In: 12th International Workshop on Combinatorial Image Analysis. Lecture Notes in Computer Science, vol. 4958, pp. 26–37. Springer, Berlin (2008)
16. de Vieilleville, F., Lachaud, J.O.: Comparison and improvement of tangent estimators on digital curves. Pattern Recognit. **42**(8), 1693–1707 (2009)
17. de Vieilleville, F., Lachaud, J.O., Feschet, F.: Convex digital polygons, maximal digital straight segments and convergence of discrete geometric estimators. J. Math. Imaging Vis. **27**(2), 139–156 (2007)
18. DGtal: digital geometry tools and algorithms library. http://liris.cnrs.fr/dgtal
19. Dorst, L., Smeulders, A.W.M.: Length estimators for digitized contours. Comput. Vis. Graph. Image Process. **40**(3), 311–333 (1987)
20. Elder, J., Zucker, S.W.: Local scale control for edge detection and blur estimation. IEEE Trans. Pattern Anal. Mach. Intell. **20**(7), 669–716 (1998)

21. Esbelin, H.A., Malgouyres, R.: Convergence of binomial-based derivative estimation for c2-noisy discretized curves. In: 15th Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 5810, pp. 57–66 (2009)
22. Feschet, F., Tougne, L.: Optimal time computation of the tangent of a discrete curve: application to the curvature. In: Proc. 8th Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 1568, pp. 31–40 (1999)
23. Fiorio, C., Mercat, C., Rieux, F.: Curvature estimation for discrete curves based on auto-adaptive masks of convolution. In: Computational Modeling of Objects Presented in Images. Lecture Notes in Computer Science, vol. 6026, pp. 47–59 (2010)
24. Fleischmann, O., Wietzke, L., Sommer, G.: A novel curvature estimator for digital curves and images. In: 32th Annual Symposium of the German Association for Pattern Recognition. Lecture Notes in Computer Science, vol. 6376, pp. 442–451 (2010)
25. Gérard, Y., Provot, L., Feschet, F.: Introduction to digital level layers. In: 16th IAPR International Conference Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 6607, pp. 83–94. Springer, Berlin (2011)
26. Goshtasby, A., Satter, M.: An adaptive window mechanism for image smoothing. Comput. Vis. Image Underst. **111**, 155–169 (2008)
27. Hermann, S., Klette, R.: A comparative study on 2d curvature estimators. In: 17th International Conference on Computer Theory and Applications, pp. 584–589 (2007)
28. Huxley, M.N.: Exponential sums and lattice points. Proc. Lond. Math. Soc. **60**, 471–502 (1990)
29. ImaGene: generic digital image library. https://gforge.liris.cnrs.fr/projects/imagene
30. Kerautret, B., Lachaud, J.: Multiscale analysis of discrete contours for unsupervised noise detection. In: 13th International Workshop on Combinatorial Image Analysis. Lecture Notes in Computer Science, vol. 5852, pp. 187–200. Springer, Mexico (2009)
31. Kerautret, B., Lachaud, J.: Meaningful scales detection along digital contours for unsupervised local noise estimation. IEEE Trans. Pattern Anal. Mach. Intell. (submitted)
32. Kerautret, B., Lachaud, J.O.: Curvature estimation along noisy digital contours by approximate global optimization. Pattern Recognit. **42**(10), 2265–2278 (2009)
33. Kerautret, B., Lachaud, J.O.: Noise level and meaningful scale detection online demonstration. http://kerrecherche.iutsd.uhp-nancy.fr/MeaningfulBoxes (2010)
34. Kervrann, C.: An adaptive window approach for image smoothing and structures preserving. In: 8th European Conference on Computer Vision. Lecture Notes in Computer Science, vol. 3023, pp. 132–144. Springer, Berlin (2004)
35. Klette, R., Rosenfeld, A.: Digital Geometry—Geometric Methods for Digital Picture Analysis (2004)
36. Klette, R., Žunić, J.: Multigrid convergence of calculated features in image analysis. J. Math. Imaging Vis. **13**, 173–191 (2000)
37. Kovalevsky, V., Fuchs, S.: Theoretical and experimental analysis of the accuracy of perimeter estimates. In: Proc. Robust Computer Vision, pp. 218–242 (1992)
38. Kovalevsky, V.A.: New definition and fast recognition of digital straight segments and arcs. In: 10th International Conference on Pattern Analysis and Pattern Recognition, pp. 31–34 (1990)
39. Lachaud, J.O.: Espaces non-euclidiens et analyse d'image : modèles déformables riemanniens et discrets, topologie et géométrie discrète. Habilitation diriger des recherches, Université Bordeaux 1, Talence (2006) (in French)
40. Lachaud, J.O., Vialard, A.: Geometric measures on arbitrary dimensional digital surfaces. In: 11th International Conference Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 2886, pp. 434–443. Springer, Berlin (2003)
41. Lachaud, J.O., Vialard, A., de Vieilleville, F.: Analysis and comparative evaluation of discrete tangent estimators. In: 12th International Conference on Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 3429, pp. 140–251. Springer, Berlin (2005)
42. Lachaud, J.O., Vialard, A., de Vieilleville, F.: Fast, accurate and convergent tangent estimation on digital contours. Image Vis. Comput. **25**(10), 1572–1587 (2007)

43. Latecki, L.J., Conrad, C., Gross, A.: Preserving topology by a digitization process. J. Math. Imaging Vis. **8**(2), 131–159 (1998)
44. Lenoir, A.: Fast estimation of mean curvature on the surface of a 3d discrete object. In: 7th International Workshop on Discrete Geometry for Computer Imagery, pp. 175–186. Springer, Berlin (1997)
45. Lenoir, A., Malgouyres, R., Revenu, M.: Fast computation of the normal vector field of the surface of a 3d discrete object. In: 6th International Workshop on Discrete Geometry for Computer Imagery, vol. 1176, pp. 101–109 (1996)
46. Lewiner, T., Gomes, J.D., Jr., Lopes, H., Craizer, M.: Curvature and torsion estimators based on parametric curve fitting. Comput. Graph. **29**, 641–655 (2005)
47. Lien, S.L.C.: Combining computation with geometry. Ph.D. thesis, California Institute of Technology (1984)
48. Lindblad, J.: Surface area estimation of digitized 3D objects using weighted local configurations. Image Vis. Comput. **23**(2), 111–122 (2005)
49. Liu, H., Latecki, L., Liu, W.: A unified curvature definition for regular, polygonal, and digital planar curves. Int. J. Comput. Vis. **80**(1), 104–124 (2008)
50. Liu, Y.S., Yi, J., Zhang, H., Zheng, G.Q., Paul, J.C.: Surface area estimation of digitized 3d objects using quasi-Monte Carlo methods. Pattern Recognit. **43**(11), 3900–3909 (2010)
51. Malgouyres, R., Brunet, F., Fourey, S.: Binomial convolutions and derivatives estimation from noisy discretizations. In: 14th Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 4992, pp. 370–379 (2008)
52. Marji, M.: On the detection of dominant points on digital planar curves. Ph.D. thesis, Wayne State University, Detroit (2003)
53. Matas, J., Shao, Z., Kittler, J.: Estimation of curvature and tangent direction by median filtered differencing. In: 8th International Conference on Image Analysis and Processing. Lecture Notes in Computer Science, vol. 974, pp. 83–88 (1995)
54. Merigot, Q., Ovsjanikov, M., Guibas, L.: Robust Voronoi-based curvature and feature estimation. In: SIAM/ACM Joint Conference on Geometric and Physical Modeling (2009)
55. Nguyen, T., Debled-Rennesson, I.: Curvature estimation in noisy curves. In: 12th International Conference on Computer Analysis of Images and Patterns. Lecture Notes in Computer Science, vol. 4673, pp. 474–481 (2007)
56. Novotni, M., Klein, R.: Shape retrieval using 3D Zernike descriptors. Comput. Aided Des. **36**(11), 1047–1062 (2004)
57. Pottmann, H., Wallner, J., Huang, Q., Yang, Y.: Integral invariants for robust geometry processing. Comput. Aided Geom. Des. **26**(1), 37–60 (2009)
58. Pottmann, H., Wallner, J., Yang, Y., Lai, Y., Hu, S.: Principal curvatures from the integral invariant viewpoint. Comput. Aided Geom. Des. **24**(8–9), 428–442 (2007)
59. Proffitt, D., Rosen, D.: Metrication errors and coding efficiency of chain-encoding schemes for the representation of lines and edges. Comput. Graph. Image Process. **10**(4), 318–332 (1979)
60. Provençal, X., Lachaud, J.O.: Two linear-time algorithms for computing the minimum length polygon of a digital contour. In: 15th International Conference on Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 5810, pp. 104–117 (2009)
61. Provot, L., Gérard, Y.: Estimation of the derivatives of a digital function with a convergent bounded error. In: 16th IAPR International Conference in Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 6607, pp. 284–295. Springer, Berlin (2011)
62. Roussillon, T., Lachaud, J.O.: Accurate curvature estimation along digital contours with maximal digital circular arcs. In: 14th International Workshop in Combinatorial Image Analysis. Lecture Notes in Computer Science, vol. 6636, pp. 43–55. Springer, Berlin (2011)
63. Roussillon, T., Sivignon, I.: Faithful polygonal representation of the convex and concave parts of a digital curve. Pattern Recognit. **44**(10–11), 2693–2700 (2011)
64. Roussillon, T., Sivignon, I., Tougne, L.: On three constrained versions of the digital circular arc recognition problem. In: 15th IAPR International Conference on Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 5810, pp. 34–45 (2009)

65. Sloboda, F., Zaťko, B., Stoer, J.: On approximation of planar one-dimensional continua. In: Advances in Digital and Computational Geometry, pp. 113–160 (1998)
66. Tajine, M., Daurat, A.: On local definitions of length of digital curves. In: 11th International Conference on Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 2886, pp. 114–123. Springer, Berlin (2003)
67. Teague, M.R.: Image analysis via the general theory of moments. J. Opt. Soc. Am. **70**(8), 920–930 (1980)
68. Vialard, A.: Geometrical parameters extraction from discrete paths. In: 7th Discrete Geometry for Computer Imagery. Lecture Notes in Computer Science, vol. 1176, pp. 24–35 (1996)
69. Windreich, G., Kiryati, N., Lohmann, G.: Voxel-based surface area estimation: from theory to practice. Pattern Recognit. **36**(11), 2531–2541 (2003)
70. Worring, M., Smeulders, A.W.M.: Digital curvature estimation. CVGIP, Image Underst. **58**(3), 366–382 (1993)

Index

V.E. Brimkov, R.P. Barneva (eds.), *Digital Geometry Algorithms*,
Lecture Notes in Computational Vision and Biomechanics 2,
DOI 10.1007/978-94-007-4174-4, © Springer Science+Business Media Dordrecht 2012