The Design and Analysis of Spatial Data Structures

The Design and Analysis of Spatial Data Structures

Hanan Samet
UNIVERSITY OF MARYLAND

ADDISON - WESLEY PUBLISHING COMPANY, INC.
Reading, Massachusetts • Menlo Park, California • New York
Don Mills, Ontario • Wokingham, England • Amsterdam
Bonn • Sydney • Singapore • Tokyo • Madrid • San Juan

This book is in the Addison-Wesley Series in Computer Science
Michael A. Harrison: Consulting Editor

Library of Congress Cataloging-in-Publication Data

Samet, Hanan.
The Design and analysis of spatial data structures/by Hanan Samet.
p. cm.
Bibliography: p.
Includes index.
ISBN 0-201-50255-0
1. Data structures (Computer science) 2. Computer graphics.
I. Title.
QA76.9.D35S26 1989 89-30382
005.7'3--dc19 CIP

Reprinted with corrections April, 1990

CDEFGHIJ-MA-9 10

Credits:
Thor Bestul created the cover art.

Gyuri Fekete generated Figure 1.16; Daniel DeMenthon, Figures 1.20, 1.21, and 1.23; Jiang-Hsing Chu, Figures 2.48 and 2.52; and Walid Aref, Figures 4.38 through 4.40.

Figures 1.1, 4.9, and 4.10 are from H. Samet and R. E. Webber, On encoding boundaries with quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence 6*, 3 (May 1984), 365–369. © 1984 IEEE. Reprinted by permission of IEEE.

Figures 1.2, 1.3, 1.5 through 1.10, 1.12, 1.14, 1.25, 1.26, 2.3, 2.4, 2.18, 2.20, 2.30, 2 32, 2.53, 2.54, 2.57, 2.58, 3.20, 3.21, 4.1 through 4.5, 4.7, 4.8, 4.11, and 5.2 are from H. Samet, The quadtree and related hierarchical data structures, *ACM Computing Surveys 16*, 2 (June 1984), 187–260. Reprinted by permission of ACM.

Figures 1.4 and 5.6 are from H. Samet and R. E. Webber, Hierarchical data structures and algorithms for computer graphics. Part I. Fundamentals, *IEEE Computer Graphics and Applications 8*, 3 (May 1988), 48–68. © 1988 IEEE. Reprinted by permission of IEEE.

Figure 1.30 is from M. Li, W. I. Grosky, and R. Jain, Normalized quadtrees with respect to translations. *Computer Graphics and Image Processing 20*, 1 (September 1982), 72–81. Reprinted by permission of Academic Press.

Figures 2.7 and 2.10 through 2.15 are from H. Samet, Deletion in two-dimensional quad trees, *Communications of the ACM 23*, 12 (December 1980), 703–710. Reprinted by permission of ACM.

Figures 2.26 and 2.27 are from D. T. Lee and C. K. Wong, Worst-case analysis for region and partial region searches in multidimensional binary search trees and quad trees, *Acta Informatica 9*, 1 (1977), 23–29. Reprinted by permission of Springer-Verlag.

Continued on p. 493

PREFACE

Spatial data consist of points, lines, rectangles, regions, surfaces, and volumes. The representation of such data is becoming increasingly important in applications in computer graphics, computer vision, database management systems, computer-aided design, solid modeling, robotics, geographic information systems (GIS), image processing, computational geometry, pattern recognition, and other areas. Once an application has been specified, it is common for the spatial data types to be more precise. For example, consider a geographic information system (GIS). In such a case, line data are differentiated on the basis of whether the lines are isolated (e.g., earthquake faults), elements of tree-like structures (e.g., rivers and their tributaries), or elements of networks (e.g., rail and highway systems). Similarly region data are often in the form of polygons that are isolated (e.g., lakes), adjacent (e.g., nations), or nested (e.g., contours). Clearly the variations are large.

Many of the data structures currently used to represent spatial data are hierarchical. They are based on the principle of recursive decomposition (similar to *divide & conquer* methods [Aho74]). One such data structure is the quadtree (octree in three dimensions). As we shall see, the term *quadtree* has taken on a generic meaning. In this book, it is my goal to show how a number of hierarchical data structures used in different domains are related to each other and to quadtrees. My presentation concentrates on these different representations and illustrates how a number of basic operations that use them are performed.

Hierarchical data structures are useful because of their ability to focus on the interesting subsets of the data. This focusing results in an efficient representation and in improved execution times. Thus they are particularly convenient for performing set operations. Many of the operations described can often be performed as efficiently, or more so, with other data structures. Nevertheless hierarchical data structures are attractive because of their conceptual clarity and ease of implementation. In addition, the use of some of them provides a spatial index. This is very useful in applications involving spatial databases.

As an example of the type of problems to which the techniques described in this book are applicable, consider a cartographic database consisting of a number of maps and some typical queries. The database contains a contour map, say at 50-foot elevation intervals, and a land use map classifying areas according to crop growth. Our goal is to determine all regions between 400- and 600-foot elevation levels where wheat is grown. This will require an intersection operation on the two maps. Such an analysis could be rather costly, depending on the way the maps are represented. For example, since areas where corn is grown are of no interest, we wish to spend a minimal amount of effort searching such regions. Yet traditional region representations such as the boundary code [Free74] are very local in application, making it difficult to avoid examining a corn-growing area that meets the desired elevation criterion. In contrast, hierarchical representations such as the region quadtree are more global in nature and enable the elimination of larger areas from consideration.

Another query might be to determine whether two roads intersect within a given area. We could check them point by point; however, a more efficient method of analysis would be to represent them by a hierarchical sequence of enclosing rectangles and to discover whether in fact the rectangles do overlap. If they do not, the search is terminated. If an intersection is possible, more work may have to be done, depending on which method of representation is used.

A similar query can be constructed for point data—for example, to determine all cities within 50 miles of St. Louis that have a population in excess of 20,000. Again we could check each city individually. However, using a representation that decomposes the United States into square areas having sides of length 100 miles would mean that at most four squares need to be examined. Thus California and its adjacent states can be safely ignored.

Finally, suppose we wish to integrate our queries over a database containing many different types of data (e.g., points, lines, areas). A typical query might be, "Find all cities with a population in excess of 5,000 people in wheat-growing regions within 20 miles of the Mississippi River." In this book we will present a number of different ways of representing data so that such queries and other operations can be efficiently processed.

This book is organized as follows. There is one chapter for each spatial data type, in which I present a number of different data structures. The aim is to gain the ability to evaluate them and to determine their applicability. Two problems are treated in great detail: the rectangle intersection problem, discussed in the context of the representation of collections of small rectangles (Chapter 3), and the point location problem, discussed in the context of the representation of curvilinear data (Chapter 4). A comprehensive treatment of the use of quadtrees and octrees in other applications in computer graphics, image processing, and geographic information systems (GIS) can be found in [Same90b].

Chapter 1 gives a general introduction to the principle of recursive decomposition with a concentration on two-dimensional regions. Key properties, as well as a historical overview, are presented.

Chapter 2 discusses hierarchical representations of multidimensional point data. These data structures are particularly useful in applications in database management systems because they are designed to facilitate responses to search queries.

Chapter 3 examines the hierarchical representation of collections of small rectangles. Such data arise in applications in computational geometry, very large-scale integrations (VLSI), cartography, and database management. Examples from these fields (e.g., the rectangle intersection problem) are used to illustrate their differences. Many of the representations are closely related to those used for point data. This chapter is an expansion of [Same88a].

Chapter 4 treats the hierarchical representation of curvilinear data. The primary focus is on the representation of polygonal maps. The goal is to be able to solve the point location problem. Quadtree-like solutions are compared with those from computational geometry such as the K-structure [Kirk83] and the layered dag [Edel86a].

Chapter 5 looks at the representation of three-dimensional region data. In this case, a number of octree variants are examined, as well as constructive solid geometry (CSG) and the boundary model (BRep). Algorithms are discussed for converting between some of these representations. The representation of surfaces (i.e., 2.5-dimensional data) is also briefly discussed in this chapter.

There are a number of topics for which justice requires a considerably more detailed treatment. However, due to space limitations, I have omitted a detailed discussion of them and instead refer interested readers to the appropriate literature. For example, surface representations are discussed briefly with three-dimensional data in Chapter 5 (also see Chapter 7 of [Same90b]). The notion of a pyramid is presented only at a cursory level in Chapter 1 so that it can be contrasted with the quadtree. In particular, the pyramid is a multiresolution representation, whereas the quadtree is a variable resolution representation. Readers are referred to Tanimoto and Klinger [Tani80] and the collection of papers edited by Rosenfeld [Rose83a] for a more comprehensive exposition on pyramids.

Results from computational geometry, although related to many of the topics covered in this book, are discussed only in the context of representations for collections of small rectangles (Chapter 3) and curvilinear data (Chapter 4). For more details on early work involving some of these and related topics, interested readers should consult the surveys by Bentley and Friedman [Bent79b], Overmars [Over88a], Edelsbrunner [Edel84], Nagy and Wagle [Nagy79], Peuquet [Peuq84], Requicha [Requ80], Srihari [Srih81], Samet and Rosenfeld [Same80d], Samet [Same84b, Same88a], Samet and Webber [Same88c, Same88d], and Toussaint [Tous80].

There are also a number of excellent texts containing material related to the topics that I cover. Rosenfeld and Kak [Rose82a] should be consulted for an encyclopedic treatment of image processing. Mäntylä [Mänt87] has written a comprehensive introduction to solid modeling. Burrough [Burr86] provides a survey of geographic information systems (GIS). Overmars [Over83] has produced a particularly good treatment of multidimensional point data. In a similar vein, see Mehlhorn's [Mehl84] unified treatment of multidimensional searching and computational geometry. For thorough introductions to computational geometry, see Preparata and

Shamos [Prep85] and Edelsbrunner [Edel87] (also see [Prep83, ORou88]). A broader view of the literature can be found in related bibliographies such as the ongoing collective effort coordinated by Edelsbrunner [Edel83c, Edel88], and Rosenfeld's annual collection of references in the journal *Computer Vision, Graphics, and Image Processing* (e.g., [Rose88]).

Nevertheless, given the broad and rapidly expanding nature of the field, I am bound to have omitted significant concepts and references. In addition at times I devote a disproportionate amount of attention to some concepts at the expense of others. This is principally for expository purposes; I feel that it is better to understand some structures well rather than to give readers a quick runthrough of buzzwords. For these indiscretions, I beg your pardon and hope you nevertheless bear with me.

My approach is an algorithmic one. Whenever possible, I have tried to motivate critical steps in the algorithms by a liberal use of examples. I feel that it is of paramount importance for readers to see the ease with which the representations can be implemented and used. In each chapter, except for the introduction (Chapter 1), I give at least one detailed algorithm using pseudo-code so that readers can see how the ideas can be applied. The pseudo-code is a variant of the ALGOL [Naur60] programming language that has a data structuring facility incorporating pointers and record structures. Recursion is used heavily. This language has similarities to C [Kern78], PASCAL [Jens74], SAIL [Reis76], and ALGOL W [Baue68]. Its basic features are described in the Appendix. However, the actual code is not crucial to understanding the techniques, and it may be skipped on a first reading. The index indicates the page numbers where the code for each algorithm is found.

In many cases I also give an analysis of the space and time requirements of different data structures and algorithms. The analysis is usually of an asymptotic nature and is in terms of *big O* and Ω notation [Knut76]. The *big O* notation denotes an upper bound. For example, if an algorithm takes $O(\log_2 N)$ time, then its worst-case behavior is never any worse than $\log_2 N$. The Ω notation denotes a lower bound. As an example of its use, consider the problem of sorting N numbers. When we say that sorting is $\Omega(N \cdot \log_2 N)$ we mean that given any algorithm for sorting, there is some set of N input values for which the algorithm will require at least this much time.

At times I also describe implementations of some of the data structures for the purpose of comparison. In such cases counts, such as the number of fields in a record, are often given. These numbers are meant only to amplify the discussion. They are not to be taken literally, as improvements are always possible once a specific application is analyzed more carefully.

Each chapter contains a substantial number of exercises. Many of the exercises develop further the material in the text as a means of testing the reader's understanding, as well as suggesting future directions. When the exercise or its solution is not my own, I have preceded it with the name of its originator. The exercises have not been graded by difficulty. They rarely require any mathematical skills beyond the undergraduate level for their solution. However, while some of the exercises are quite straightforward, others require some ingenuity. Solutions, or references to papers that

contain the solution, are provided for a substantial number of the exercises that do not require programming. Readers are cautioned to try to solve the exercises before turning to the solutions. It is my belief that much can be learned this way (for the student and, even more so, for the author). The motivation for undertaking this task was my wonderful experience on my first encounter with the rich work on data structures by Knuth [Knut73a, Knut73b].

An extensive bibliography is provided. It contains entries for both this book and the companion text [Same90b]. Not all of the references that appear in the bibliography are cited in the two texts. They are retained for the purpose of giving readers the ability to access the entire body of literature relevant to the topics discussed in them. Each reference is annotated with a key word(s) and a list of the numbers of the sections in which it is cited in either of the texts (including exercises and solutions). In addition, a name and credit index is provided that indicates the page numbers in this book on which each author's work is cited or a credit is made.

ACKNOWLEDGMENTS

Over the years I have received help from many people, and I am extremely grateful to them. In particular Robert E. Webber, Markku Tamminen, and Michael B. Dillencourt have generously given me much of their time and have gone over critical parts of the book. I have drawn heavily on their knowledge of some of the topics covered here. I have also been extremely fortunate to work with Azriel Rosenfeld over the past ten years. His dedication and scholarship have been a true inspiration to me. I deeply cherish our association.

I was introduced to the field of spatial data structures by Gary D. Knott who asked "how to delete in point quadtrees." Azriel Rosenfeld and Charles R. Dyer provided much interaction in the initial phase of my research. Those discussions led to the discovery of the neighbor-finding principle. It is during that time that many of the basic conversion algorithms between quadtrees and other image representations were developed as well. I learned much about image processing and computer vision from them. Robert E. Webber taught me computer graphics, Markku Tamminen taught me solid modeling and representations for multiattribute data, and Michael B. Dillencourt taught me about computational geometry.

During the time that this book was written, my research was supported, in part, by the National Science Foundation, the Defense Mapping Agency, the Harry Diamond Laboratory, and the Bureau of the Census. In particular I would like to thank Richard Antony, Y. T. Chien, Su-shing Chen, Hank Cook, Phil Emmerman, Joe Rastatter, Alan Saalfeld, and Larry Tokarcik. I am appreciative of their support.

Many people helped me in the process of preparing the book for publication. Acknowledgments are due to Rene McDonald for coordinating the day-to-day matters

of getting the book out and copyediting; to Scott Carson, Emery Jou, and Jim Purtilo for TROFF assistance beyond the call of duty; to Marisa Antoy and Sergio Antoy for designing and implementing the algorithm formatter used to typeset the algorithms; to Barbara Burnett, Michael B. Dillencourt, and Sandra German for help with the index; to Jay Weber for setting up the TROFF macro files so that I can keep track of symbolic names and thus be able to move text around without worrying about the numbering of exercises, sections, and chapters; to Liz Allen for early TROFF help; to Nono Kusuma, Mark Stanley, and Joan Wright Hamilton for drawing the figures; to Richard Muntz and Gerald Estrin for providing temporary office space and computer access at UCLA; to Sandy German, Gwen Nelson, and Janet Salzman for help in initial typing of the manuscript; to S. S. Iyengar, Duane Marble, George Nagy, and Terry Smith who reviewed the book; and to Peter Gordon, John Remington, and Keith Wollman at Addison-Wesley Publishing Company for their encouragement and confidence in this project.

Aside from the individuals named above, I have also benefited from discussions with many other people over the past years. They have commented on various parts of the book and include Chuan-Heng Ang, Walid Aref, James Arvo, Thor Bestul, Sharat Chandran, Chiun-Hong Chien, Jiang-Hsing Chu, Leila De Floriani, Daniel DeMenthon, Roger Eastman, Herbert Edelsbrunner, Christos Faloutsos, George (Gyuri) Fekete, Kikuo Fujimura, John Gannon, John Goldak, Erik Hoel, Liuqing Huang, Frederik W. Jansen, Ajay Kela, David Kirk, Per Åke Larson, Dani Lischinski, Don Meagher, David Mount, Randal C. Nelson, Glenn Pearson, Ron Sacks-Davis, Timos Sellis, Clifford A. Shaffer, Deepak Sherlekar, Li Tong, Brian Von Herzen, Peter Widmayer, and David Wise. I deeply appreciate their help.

A GUIDE TO THE INSTRUCTOR

This book can be used in a second data structures course, one with emphasis on the representation of spatial data. The focus is on the use of the principle of divide-and-conquer for which hierarchical data structures provide a good demonstration. Throughout the book both worst-case optimal methods and methods that work well in practice are emphasized in conformance with my view that the well-rounded computer scientist should be conversant with both types of algorithms. This material is more than can be covered in one semester; but the instructor can reduce it as necessary. For example, the detailed examples can be skipped or used as a basis of a term project or programming assignments.

The book can also be used to organize a course to be prerequisite to courses in computer graphics and solid modeling, computational geometry, database management systems, multidimensional searching, image processing, and VLSI design. The discussions of the representations of two-dimensional regions in Chapter 1, polygonal representations in Chapter 4, and most of Chapter 5 are relevant to computer graphics and solid modeling. The discussions of plane-sweep methods and their associated data structures such as segment trees, interval trees, and priority search trees in Sections 3.2 and 3.3 and point location and associated data structures such as the

K-structure and the layered dag in Section 4.3 are relevant to computational geometry. Bucket methods such as linear hashing, spiral hashing, grid file, and EXCELL, in Section 2.8, and R-trees in Section 3.5.3 are important in the study of database management systems. Methods for multidimensional searching that are discussed include k–d trees in Section 2.4, range trees and priority search trees in Section 2.5, and point-based rectangle representations in Section 3.4. The discussions of the representation of two-dimensional regions in Chapter 1, polygonal representations in Chapter 4, and use of point methods for focussing the Hough Transform are relevant to image processing. Finally the rectangle-representation methods and plane-sweep methods of Chapter 3 are important in the field of VLSI design.

The natural home for courses that use this book is in a computer science department, but the book could also be used in a curriculum in geographic information systems (GIS). Such a course is offered in geography departments. The emphasis for a course in this area would be on the use of quadtree-like methods for representing spatial data.

CONTENTS

1 INTRODUCTION

There are numerous hierarchical data structuring techniques in use for representing spatial data. One commonly used technique is the quadtree, which has evolved from work in different fields. Thus it is natural that a number of adaptations of it exist for each spatial data type. Its development has been motivated to a large extent by a desire to save storage by aggregating data having identical or similar values. We will see, however, that this is not always the case. In fact, the savings in execution time that arise from this aggregation are often of equal or greater importance.

In this chapter we start with a historical overview of quadtrees, including definitions. Since the primary focus in this book is on the representation of regions, what follows is a discussion of region representation in the context of different space decomposition methods. This is done by examining polygonal and nonpolygonal tilings of the plane. The emphasis is on justifying the use of a decomposition into squares. We conclude with a detailed analysis of the space requirements of the quadtree representation.

Most of the presentation in this chapter is in the context of two-dimensional regions. The extension of the topics in this chapter, and remaining chapters, to three-dimensional region data, and higher, is straightforward and, aside from definitions, is often left to the exercises. Nevertheless, the concept of an octree, a quadtree-like representation of three-dimensional regions, is defined and a brief explanation is given of how some of the results described here are applicable to higher-dimensional data.

1.1 BASIC DEFINITIONS

First, we define a few terms with respect to two-dimensional data. Assume the existence of an array of picture elements (termed *pixels*) in two dimensions. We use the term *image* to refer to the original array of pixels. If its elements are black or

white, then it is said to be *binary*. If shades of gray are possible (i.e., gray levels), the image is said to be a *gray-scale* image. In the discussion, we are primarily concerned with binary images. Assume that the image is on an infinite background of white pixels. The *border* of the image is the outer boundary of the square corresponding to the array.

Two pixels are said to be 4-*adjacent* if they are adjacent to each other in the horizontal or vertical direction. If the concept of adjacency also includes adjacency at a corner (i.e., diagonal adjacencies), then the pixels are said to be 8-*adjacent*. A set S is said to be *four-connected* (*eight-connected*) if for any pixels p, q in S there exists a sequence of pixels $p = p_0, p_1, \cdots, p_n = q$ in S such that p_{i+1} is 4-adjacent (8-adjacent) to p_i; $0 \le i < n$.

A black *region*, or black four-connected *component*, is a maximal four-connected set of black pixels. The process of assigning the same label to all 4-adjacent black pixels is called *connected component labeling* (see Chapter 5 of [Same90b]). A white *region* is a maximal *eight-connected* set of white pixels defined analogously. The complement of a black region consists of a union of eight-connected white regions. Exactly one of these white regions contains the infinite background of white pixels. All the other white regions, if any, are called *holes* in the black region. The black region, say R, is surrounded by the infinite white region and R surrounds the other white regions, if any.

A pixel is said to have four edges, each of which is of unit length. The *boundary* of a black region consists of the set of edges of its constituent pixels that also serve as edges of white pixels. Similar definitions can be formulated in terms of rectangular blocks, all of whose pixels are identically colored. For example, two disjoint blocks, P and Q, are said to be 4-*adjacent* if there exists a pixel p in P and a pixel q in Q such that p and q are 4-adjacent. Eight-adjacency for blocks (as well as connected component labeling) is defined analogously.

1.2 OVERVIEW OF QUADTREES AND OCTREES

The term *quadtree* is used to describe a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space. They can be differentiated on the following bases:

1. The type of data they are used to represent
2. The principle guiding the decomposition process
3. The resolution (variable or not)

Currently they are used for point data, areas, curves, surfaces, and volumes. The decomposition may be into equal parts on each level (i.e., regular polygons and termed a *regular decomposition*), or it may be governed by the input. In computer graphics this distinction is often phrased in terms of image-space hierarchies versus object-space hierarchies, respectively [Suth74]. The resolution of the decomposition

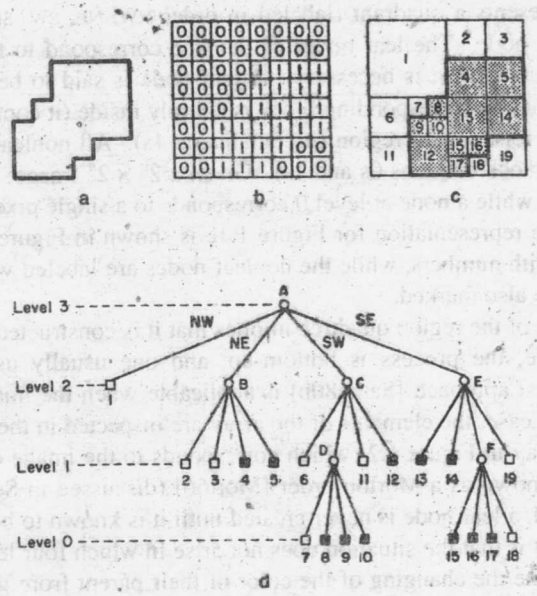

Figure 1.1 An example of (a) a region, (b) its binary array, (c) its maximal blocks (blocks in the region are shaded), and (d) the corresponding quadtree

(i.e., the number of times that the decomposition process is applied) may be fixed beforehand, or it may be governed by properties of the input data. For some applications we can also differentiate the data structures on the basis of whether they specify the boundaries of regions (e.g., curves and surfaces) or organize their interiors (e.g., areas and volumes).

The first example of a quadtree representation of data is concerned with the representation of two-dimensional binary region data. The most studied quadtree approach to region representation, called a *region quadtree* (but often termed a *quadtree* in the rest of this chapter), is based on the successive subdivision of a bounded image array into four equal-sized quadrants. If the array does not consist entirely of 1s or entirely of 0s (i.e., the region does not cover the entire array), then it is subdivided into quadrants, subquadrants, and so on, until blocks are obtained that consist entirely of 1s or entirely of 0s; that is, each block is entirely contained in the region or entirely disjoint from it. The region quadtree can be characterized as a variable resolution data structure.

As an example of the region quadtree, consider the region shown in Figure 1.1a represented by the $2^3 \times 2^3$ binary array in Figure 1.1b. Observe that the 1s correspond to picture elements (i.e., pixels) in the region, and the 0s correspond to picture elements outside the region. The resulting blocks for the array of Figure 1.1b are shown in Figure 1.1c. This process is represented by a tree of degree 4 (i.e., each nonleaf node has four sons).

In the tree representation, the root node corresponds to the entire array. Each son of a node represents a quadrant (labeled in order NW, NE, SW, SE) of the region represented by that node. The leaf nodes of the tree correspond to those blocks for which no further subdivision is necessary. A leaf node is said to be black or white depending on whether its corresponding block is entirely inside (it contains only 1s) or entirely outside the represented region (it contains no 1s). All nonleaf nodes are said to be gray (i.e., its block contains 0s and 1s). Given a $2^n \times 2^n$ image, the root node is said to be at level n while a node at level 0 corresponds to a single pixel in the image.[1] The region quadtree representation for Figure 1.1c is shown in Figure 1.1d. The leaf nodes are labeled with numbers, while the nonleaf nodes are labeled with letters. The levels of the tree are also marked.

Our definition of the region quadtree implies that it is constructed by a top-down process. In practice, the process is bottom-up, and one usually uses one of two approaches. The first approach [Same80b] is applicable when the image array is not too large. In such a case, the elements of the array are inspected in the order given by the labels on the array in Figure 1.2 (which corresponds to the image of Figure 1.1a). This order is also known as a Morton order [Mort66] (discussed in Section 1.3). By using such a method, a leaf node is never created until it is known to be maximal. An equivalent statement is that the situation does not arise in which four leaf nodes of the same color necessitate the changing of the color of their parent from gray to black or white as is appropriate. (For more details, see Section 4.1 of [Same90b].)

The second approach [Same81a] is applicable to large images. In this case, the elements of the image are processed one row at a time—for example, in the order given by the labels on the array in Figure 1.3 (which corresponds to the image of Figure 1.1a). This order is also known as a row or raster-scan order (discussed in Section 1.3). A quadtree is built by adding pixel-sized nodes one by one in the order in which they appear in the file. (For more details, see Section 4.2.1 of [Same90b].) This process can be time-consuming due to the many merging and node insertion operations that need to take place.

The above method has been improved by using a predictive method [Shaf86a, Shaf87a], which only makes a single insertion for each node in the final quadtree and performs no merge operations. It is based on processing the image in row order (top to bottom, left to right), always inserting the largest node (i.e., block) for which the current pixel is the first (upper leftmost) pixel. Such a policy avoids the necessity of merging since the upper leftmost pixel of any block is inserted before any other pixel of that block. Therefore it is impossible for four sibling nodes to be of the same color. This method makes use of an auxiliary array of size $O(2^n)$ for a $2^n \times 2^n$ image. (For more details, see Section 4.2.3 of [Same90b].)

The region quadtree is easily extended to represent three-dimensional binary region data and the resulting data structure is called a *region octree* (termed an *octree*

[1] Alternatively we can say that the root node is at depth 0 while a node at depth n corresponds to a single pixel in the image. In this book both concepts of level and depth are used to describe the relative position of nodes. The one that is chosen is context dependent.