

**This is an authorized facsimile
printed by microfilm/xerography on acid-free paper
in 1985 by
UNIVERSITY MICROFILMS INTERNATIONAL
Ann Arbor, Michigan, U.S.A.**

3058

8421278

Webber, Robert Edward

ANALYSIS OF QUADTREE ALGORITHMS

University of Maryland

PH.D. 1983

**University
Microfilms
International** 300 N. Zeeb Road, Ann Arbor, MI 48106

INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.
2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.
3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.
4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.
5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

**University
Microfilms
International**

300 N. Zeeb Road
Ann Arbor, MI 48106

ANALYSIS OF QUADTREE ALGORITHMS

by

Robert Edward Webber

**Dissertation submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1983**

copy 1

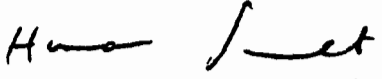
© Copyright Robert Edward Webber 1983

APPROVAL SHEET

Title of Thesis: Analysis of Quadtree Algorithms

Name of Candidate: Robert Edward Webber
Doctor of Philosophy, 1983

Thesis and Abstract Approved:



Hanan Samet
Associate Professor
Computer Science

Date Approved: Dec. 2, 1983

ABSTRACT

Title of Dissertation: Analysis of Quadtree Algorithms

Robert Edward Webber, Doctor of Philosophy, 1983

Dissertation directed by: Hanan Samet
Associate Professor
Computer Science

In this thesis, several aspects of quadtree representations are analyzed. The quadtree is a hierarchical variable-resolution data structure suitable for representing the geometric objects of computer graphics, the polygonal maps of computer cartography, and the digitized images of computer vision. The analysis of quadtrees is presented in three parts: (1) a formal semantics for quadtree algorithms, (2) improved algorithms for manipulating the standard region quadtree, and (3) adaptations of the quadtree methodology to the task of representing polygonal maps.

The first portion of this thesis provides a formal semantics for quadtree algorithms, simultaneously overviewing and unifying previous research in this field. Such a semantics has many usages. It forms a basis for correctness proofs of quadtree algorithms. It provides a foundation for automatic derivation of variations of quadtree algorithms. It guides comparison of quadtree algorithms with algorithms based on other hierarchical data structures.

In the next portion of this thesis, the worst-case behavior of standard region (and line) quadtree algorithms is investigated. An improvement in the worst-case performance results from usage of a quadtree normalization algorithm that calculates a good placement of an image on the digitization grid. This normalization algorithm is first developed in order to show that a quadtree can be constructed from a chain code

in time proportional to the length of the chain code. The algorithm is then modified to allow the building of a normalized quadtree from an arbitrary quadtree in time proportional to the size of the two quadtrees. Algorithms for connected-component analysis and quadtree-to-chain-code conversion on normalized quadtrees are developed that execute in time proportional to the number of nodes in the normalized quadtree.

In the last portion of this thesis, we consider the problem of how to adapt the quadtree methodology to the task of representing polygonal maps. Three approaches for representing polygonal maps are presented. First, we analyze the storage reduction that results from performing common subtree elimination on a quadtree that represents a polygonal map. Next, we consider how to use point quadtrees to store a Voronoi tessellation of a polygonal map. Finally, we develop a quadtree variant, called a PM quadtree, that provides a compact representation that is easy to both access and update.

TABLE OF CONTENTS

1. INTRODUCTION	1
2. A FORMAL BASIS FOR A THEORY OF QUADTREE ALGORITHMS	21
2.1. PRELIMINARY DEFINITIONS AND THEOREMS	21
2.1.1. DEFINING NEIGHBOR RELATIONS	22
2.1.2. THEOREMS REGARDING NEIGHBOR RELATIONS	25
2.2. THE DEFINITION OF TWO TYPES OF QUADTREES	36
2.2.1. DEFINITION OF PICTURES AND THEIR PROPERTIES	37
2.2.2. DEFINITION OF REGION PYRAMIDS AND QUADTREES	40
2.2.3. DEFINITIONS RELATING TO AND INCLUDING LINE QUADTREES	42
2.3. RESULTS ON THE SIZE OF QUADTREES	46
2.4. ON FINDING NEIGHBORING QUADTREE NODES	55
3. NORMALIZING QUADTREES WITH RESPECT TO AGGREGATE NEIGHBOR FINDING	92
3.1. DEVELOPMENT OF ANF ALGORITHMS FOR PYRAMIDS	92
3.1.1. THE OPTIMAL POSITIONING OF CHAIN CODES	92
3.1.2. THE CHAIN CODE TO QUADTREE ALGORITHM	96
3.2. GENERALIZATION OF ANF ALGORITHM TO DIRECTLY TRANSFORM QUADTREES	98
3.2.1. OPTIMAL POSITIONING OF QUADTREES	98
3.2.2. USING ANF NORMALIZATION IN OTHER ALGORITHMS	99
3.2.3. COMPARISON OF ANF TRANSFORM WITH NODE MINIMIZATION	101
4. USING QUADTREES TO STORE POLYGONAL MAPS	110
4.1. A GEOMETRIC MODEL FOR THE NEIGHBOR AXIOMS	111
4.2. USING COMMON SUBTREE ELIMINATION TO COMPACT QUADTREES	114
4.3. USING VORONOI DIAGRAMS AND PR QUADTREES TO STORE POLYGONAL MAPS	118
4.4. USING MAP VERTICES AND PR QUADTREES TO STORE POLYGONAL MAPS	121
4.4.1. BACKGROUND FOR PM QUADTREE DEVELOPMENT	121
4.4.2. THE PM_1 QUADTREE	123
4.4.3. THE PM_2 QUADTREE	125
4.4.4. THE PM_3 QUADTREE	126
4.4.5. ALGORITHMS FOR PM QUADTREES	127
4.4.6. POINT-IN-POLYGON DETERMINATION	128
4.4.7. LINE SEGMENT INSERTION IN PM QUADTREES	130
4.4.8. OVERLAY ALGORITHM FOR PM QUADTREES	132
4.4.9. CONCLUSIONS	137

5. CONCLUSIONS**159**

LIST OF FIGURES

Figure 1-1:	An unbiased decomposition of the viewing window	8
Figure 1-2:	A biased decomposition of the viewing window	9
Figure 1-3:	A sample image	10
Figure 1-4:	4x4 decomposition of image in Figure 1-3	11
Figure 1-5:	A quadtree decomposition of Figure 1-3	12
Figure 1-6:	Pointer representation of Figure 1-5	13
Figure 1-7:	Dewey Decimal representation of Figure 1-5	14
Figure 1-8:	DF-expression representation of Figure 1-5	15
Figure 1-9:	Four/Nine/Four decomposition of a 12x12 image	16
Figure 1-10:	Dewey Decimal representation corresponding to Figure 1-9	17
Figure 1-11:	Line quadtree equivalent of Figure 1-5	18
Figure 1-12:	Example of a point quadtree decomposition	19
Figure 1-13:	PR quadtree equivalent of Figure 1-12	20
Figure 2-1:	Layout of indices for F^2	61
Figure 2-2:	Layout of subframes for F^2	62
Figure 2-3:	Example of a pair of subframes, x and z , where $x \neq z$	63
Figure 2-4:	Representation of a binary picture denoted $\langle F^1, \{BLACK, WHITE\}, f \rangle$, with an area of 2 and a perimeter of length 6	64
Figure 2-5:	Representation of $embed(NW, P)$, where P is the binary picture of Figure 2-4	65
Figure 2-6:	Representation of $surround(P)$, where P is the binary picture of Figure 2-4	66
Figure 2-7:	Two pictures related via a frame isomorphism	67
Figure 2-8:	Two pictures related via a frame isomorphism preserving neighbor relations among non-white pixels	68
Figure 2-9:	Illustration of a shift from $\langle F^1, C, f \rangle$ to $\langle F^2, C, f'' \rangle$	69
Figure 2-10:	Example of a shift of picture in Figure 2-4 with distance 1 in the western direction	70
Figure 2-11:	The overlay of two binary pictures	71
Figure 2-12:	Representation of the region pyramid for picture in Figure 2-6	72
Figure 2-13:	Representation of the region quadtree of the pyramid in Figure 2-12	73
Figure 2-14:	Representation of the partial quadtree Q^1 of the pyramid in Figure 2-12	74
Figure 2-15:	Representation of the difference picture of Figure 2-5	75
Figure 2-16:	Representation of the line pyramid for Figure 2-15	76
Figure 2-17:	Representation of the line quadtree for Figure 2-16	77
Figure 2-18:	Representation of the partial quadtree Q^1 of the line quadtree equivalent of the region quadtree in Figure 2-13	78

Figure 2-19:	Record structure for difference picture of Figure 2-15	79
Figure 2-20:	Record structure for line pyramid of Figure 2-16	80
Figure 2-21:	Record structure for line quadtree of Figure 2-17	81
Figure 2-22:	Example of 4 by 4 checker-board picture and corresponding region quadtree	82
Figure 2-23:	Illustration of the path sequence in surround(P) corresponding to Figures 2-4 and 2-6	83
Figure 2-24:	Illustration of the reduction of the augmented path of Figure 2-23	84
Figure 2-25:	Example of a picture of a modified checker-board pattern	85
Figure 2-26:	Two quadtrees representing a 4x4 square in an 8x8 picture	86
Figure 2-27:	Example of quadtree whose aggregate neighbor finding cost is of the order of the number of nodes squared	87
Figure 3-1:	The region pyramid corresponding to the chain code for NNNNNEEESSWSSESSWWWN, where f_0 is SW.SW.NE	102
Figure 3-2:	Example of the centered snake path in a pyramid of depth 3	103
Figure 3-3:	Example of the path in Figure 3-2 shifted to a more efficient position	104
Figure 3-4:	Result of ANF shifting of Figure 3-1	105
Figure 3-5:	Actual chain code for perimeter following algorithm applied to image of Figure 3-1	106
Figure 3-6:	Quadtree corresponding to Figure 3-5	107
Figure 3-7:	A sketch of the construction of a sequence where ANF creates too many superfluous nodes	108
Figure 3-8:	A sketch of the construction of a sequence where NM causes neighbor finding to be too laborious	109
Figure 4-1:	Subframes of F^3 labeled by Cartesian coordinates of lower left hand corner	138
Figure 4-2:	A quadtree representing two regions separated by a 45 degree line	139
Figure 4-3:	Pointer representation of Figure 4-2	140
Figure 4-4:	Result of common subtree elimination on Figure 4-3	141
Figure 4-5:	A quadtree representing two regions separated by a line with slope approximately 4/16ths	142
Figure 4-6:	Pointer representation of Figure 4-5	143
Figure 4-7:	Result of common subtree elimination on Figure 4-6	144
Figure 4-8:	A Sample Voronoi Diagram	145
Figure 4-9:	First Step in Finding Voronoi Points for Map	146
Figure 4-10:	Second Step in Finding Voronoi Points for Map	147
Figure 4-11:	Final Step in Finding Voronoi Points for Map	148
Figure 4-12:	Sample polygonal map	149
Figure 4-13:	Line quadtree corresponding to Figure 4-12	150
Figure 4-14:	PR quadtree corresponding to Figure 4-12 when the line segments of the map are ignored. When the line segments are included, the PM_3 quadtree corresponding to Figure 4-12	151
Figure 4-15:	The PM_1 quadtree corresponding to Figure 4-12	152
Figure 4-16:	Example illustrating $D_3 > D_2'$ when C3 is used	153
Figure 4-17:	The PM_2 quadtree corresponding to Figure 4-12	154
Figure 4-18:	Result of using C3', instead of C3, in generating PM quadtree for Figure 4-16	155

- Figure 4-19:** Example demonstrating why diagonal neighbors should not be examined prematurely when attempting to perform point-in-polygon determination 156
- Figure 4-20:** Example demonstrating how the inserting of a q-edge can result in the splitting of one that had been inserted earlier 157
- Figure 4-21:** Example demonstrating the sensitivity fo the PR quadtree to shifts 158

LIST OF TABLES

Table 2-1:	TABLE OF INVERSE NEIGHBOR AXIOMS	88
Table 2-2:	TABLE OF NEIGHBOR COMPOSITION AXIOMS	88
Table 2-3:	A SAMPLE PROOF	89
Table 2-4:	CONSTRUCTION OF SIDE NEIGHBORS GIVEN $x \ r \ y$	89
Table 2-5:	PROCEDURAL ENCODING OF TABLE 2-4	90

CHAPTER 1

INTRODUCTION

Region representation is an important problem in image processing, computer graphics, computerized cartography, and related areas. Recently there has been much interest in hierarchical data structures, e.g., the quadtree (see overview in [30]). The term quadtree [7, 17] is used to describe a class of variable resolution data structures whose common property is that of organizing a space using the principle of recursive decomposition. In the two-dimensional case, a rectangular planar region is recursively subdivided into four rectangular planar regions until each region can be easily described by a simpler data structure (e.g., a fixed size array, a linked-list, a binary tree, etc.). Quadtrees can be classified on the basis of the type of data that they are used to represent and on the basis of the principle used to guide the decomposition process. The decomposition may be into equal-sized parts (termed a regular decomposition) or it may be arbitrary (e.g., one might subdivide a region with respect to an off-center point in anticipation of some bias in the input data).

Papers involving quadtree-like techniques first surfaced in the late sixties. They contained primarily empirical and intuitive arguments that indicate a general usefulness for quadtree storage strategies. It was not until the mid-seventies that theoretical results on quadtree structures began to appear. A brief introduction to these early papers is presented below. Further discussion of theoretical results appear in Chapter 2. In particular, Sections 2.3 and 2.4 integrate previously known theoretical results with the formalisms of Sections 2.1 and 2.2.

In 1969, Warnock [35] formulated a hidden-surface elimination algorithm using a recursive decomposition of the viewing window. As described in a later overview of hidden-surface algorithms [33], Warnock considered the output field to be a window that could either be directly processed if it was simple enough (e.g., the window is a

pixel or is homogeneous), or (otherwise) it could be subdivided into four subwindows, each of which would then be processed separately. Two decomposition strategies were considered. One strategy was the subdivision of the window into four congruent subwindows as shown in Figure 1-1. The other strategy was the subdivision of the window about an arbitrary polygon's vertex that happened to be inside the window as illustrated in Figure 1-2. The motivation for using a polygon's vertex as a subdivision point is that if a window contains a polygon's vertex, then it certainly couldn't be homogeneous.

Also, during 1969, Nilsson [22] described a structure that allowed a robot to store an internal map of a room that would permit large empty areas to be quickly processed. His approach involved a hierarchical 4x4 decomposition (i.e., a decomposition into sixteen parts) of a grid (the structure was called the GRID MODEL) where a cell was decomposed if it contained a mixture of object and background. Only binary images were discussed. An example of Nilsson's approach applied to the image in Figure 1-3 is shown in is shown in Figure 1-4. In the context of a 1970 discussion of space planning (taken to include such diverse fields as architectural design and robotics), Eastman [6] noted that instead of using a 4x4 decomposition, a 2x2 decomposition could be used, thereby yielding a view of quadtrees more consistent with the one in this dissertation. The 2x2 decomposition corresponding to Figure 1-3 is presented in Figure 1-5.

This view of a quadtree [6], henceforth termed *region quadtree*, stores a regular decomposition of an image into homogeneous regions. There are many ways of storing a region quadtree in a computer. Figure 1-6 illustrates the most straight-forward storage technique, i.e., that of representing each quadtree node by a structure that contains six fields: a father link, a node value (typically black, white, or grey), and a link to each of the node's four sons. Note that the father link is not explicitly drawn as it can be visually derived from the son links. The analysis of various quadtree algorithms presented in this thesis assumes that this storage scheme is being used. An interesting alternative scheme (see Figure 1-7) is one where the quadtree is stored as a list of paths from the root of the quadtree to each of its leafs. In this figure, NW, NE, SE, and SW denote quadrants - i.e., northwest, northeast, southeast, and southwest,

respectively. This scheme is an adaptation of the Dewey Decimal technique discussed by Knuth [18]. The usefulness of these Dewey Decimal quadtree representations has recently been investigated by Gargantini [9], with the modification that the paths are only stored for the black nodes or the white nodes. Less useful for general manipulation, but much more compact than the above representations, is the DF-expression [15] representation shown in Figure 1-8. A DF-expression (Depth First) corresponds to a preorder listing of the value fields of the nodes of a quadtree.

In 1971, Klinger [16] presented a quadtree variant that could represent images whose sides were not powers of 2. One way of representing a 12x12 image is to embed it within a 16x16 image and recursively decompose the 16x16 image into 2x2 blocks. However, Klinger advocated alternating a 2x2 decomposition with a 3x3 decomposition as shown in Figure 1-9. This variant was implemented using a Dewey Decimal representation as presented in Figure 1-10. In this figure, NW, NE, SE, and SW have their standard meaning when a region is split into four quadrants. When a region is split into nine quadrants, NW, NC, NE, EC, SE, SC, SW, WC, and CC stand for northwest, northcentral, northeast, eastcentral, southeast, southcentral, southwest, westcentral, and center regions, respectively. This data structure was designed for use in a pattern finding task on multicolored images. In 1975, further work on quadtrees [17] was reported, but the emphasis had shifted to strictly 2x2 decompositions.

Sometimes it is useful to specify regions in terms of their borders. The edge quadtree [32] is an example of such an approach that uses regular decomposition to represent such general curves (or in our case, region borders). A region is subdivided repeatedly until the leaf nodes represent regions containing a single boundary curve that can be approximated by a straight line. The line quadtree [27] is similar to the edge quadtree except that a region is repeatedly subdivided (using regular decomposition) until the leaf nodes represent regions that have no line segments passing through their interior. An example of the line quadtree equivalent of Figure 1-5 is given by Figure 1-11. There is also an edge quadtree variant [21] that subdivides, again employing regular decomposition, until the leaves represent regions containing a single curve that can be approximated by k straight lines (where k has been fixed a priori).

The data structures mentioned above are all organized about a regular decomposition of the planar image into squares. One alternative [10], closely related to the quadtree, is to use a hextree. The hextree is a data structure that is organized about a hexagonal decomposition pattern. Another alternative is to use a triangular decomposition pattern, which together with a general discussion of all possible regular tessellations of a plane appears in [2]. The appropriate tessellation for a given application is the one that best corresponds to the sampling pattern of the original raw data. If the images are purely synthetic, then the display pattern of the output device plays the deciding factor.

The relations between quadtrees and hextrees are not derived from analogies between geometric shapes, in this case squares and hexagonal configurations. Instead, these relations derive from analogies between neighbor structures. By neighbor structures, we refer to the rules used to determine which elements of a decomposition are neighbors of which other elements. A discussion of the formalization of these rules constitutes the substance of Section 2.1. The usage of these rules to define some variants of the quadtree data structure is explored in Section 2.2. It is claimed that these formalisms would also aid a comparison of quadtree and hextree algorithms, although the actual performance of such a comparison lies outside the scope of this dissertation. In Section 2.3, we analyze the space requirements of the quadtree data structures. Section 2.4 contains a presentation of various techniques for determining the neighbor of a node in a quadtree.

Although the presentation differs greatly from that in the current literature, ultimately, Chapter 2 does not yield any new results on quadtrees. Instead, it provides an alternate approach to viewing known results. In Chapter 3, we present a new quadtree transformation. This new transformation shifts the tree into a position where aggregate neighbor finding can be done in an optimal manner. The aggregate neighbor finding transformation (henceforth referred to as the *ANF transformation*) is presented (and analyzed) in Section 3.1.1 with respect to neighbor finding in a pyramid. In Section 3.1.2, these results for pyramids are extended to cover quadtrees.

Sections 3.1.1 and 3.1.2, contain an application of the ANF transformation to a

chain code and then a quadtree is constructed from the transformed chain code. In Section 3.2.1, we solve the problem of directly transforming a quadtree. Next, we demonstrate application of the ANF transformation to known quadtree algorithms. Section 3.2.2 shows that a number of these applications result in significant improvement of worst-case execution time behavior. The ANF transformation can be viewed as an image normalization procedure. Section 3.2.3 presents a comparison of the ANF normalization transformation to the node minimization transformation (henceforth referred to as the *NM transformation*) of Li, Grosky and Jain [20].

Section 4 contains an investigation of the usage of quadtrees for storing polygonal maps. Polygonal maps are often used in cartography and graphics, because straight line approximations are sufficient representations of the ideal images. Such maps are merely a collection of straight line segments. In order to enable our quadtree representations to take advantage of this extra information about the nature of the data being organized, it is necessary to establish a correspondence between the digitized space that the quadtree represents and the continuous Euclidean plane. This correspondence is presented in Section 4.1. Having made this correspondence, we can now consider three approaches to using quadtrees to store polygonal maps.

The first approach is to compress the standard region quadtree using the technique of common subtree elimination. Thus, repeated patterns in the subtrees will only be represented once. Since such repetition is common in the digitization of straight lines, much compression is expected. The analysis of this approach is presented in terms of three different types of maps:

1. randomly constructed digitized maps
2. digitized maps of a single convex object
3. digitized maps of two regions separated by a line having rational slope

Although common subtree elimination results in significant storage reduction, it seldom results in faster algorithms. Only trivial tasks, e.g., calculating the area of a map, can be speeded up to a degree comparable to the storage reduction. An alternative approach based on trying to construct a Voronoi diagram that contains all the edges of the original picture is developed in Section 4.3. The Voronoi diagram

used in this approach is stored as a collection of Voronoi points. We derive an algorithm for constructing a set of Voronoi points that determine a diagram that subsumes a given map. It quickly becomes apparent that the Voronoi diagrams are much less amenable to analysis than the digitized space that the quadtree encodes. Given a collection of Voronoi points, there remains the question of how to organize them.

The point quadtree [7] is based on an irregular decomposition of the point data being represented. It is the multidimensional analog of a binary search tree [18] where the root node of a subtree corresponds to the first node that was inserted into that subtree. Thus, the shape of the point quadtree is dependent on the order of the insertion of its constituent points. The first point inserted is stored in the root of the tree. Subsequent points are assigned subtrees according to their relation to the point stored in the root. An example of a point quadtree is shown in Figure 1-12 for points A, B, C, D, and E stored in this order.

The region quadtree can also be adapted to represent point data. We term such a tree a PR quadtree (PR denoting point region) [23]. In this case all points are associated with leaf nodes. Regular decomposition is applied until no quadrant contains more than one data point. For example, Figure 1-13 is a PR quadtree representation of the same collection of points as Figure 1-12. In order to cope with points that lie directly on one of the quadrant lines emanating from a subdivision point, we adopt the convention that each quadrant contains its upper left hand corner as well as the two half-open line segments (closed on the end touching the upper left hand corner) extending from the upper left hand corner.

We shall use the PR quadtree for storing point data, because the shape of point quadtrees is too sensitive to the order in which the data points are inserted into the tree. Since there does not yet exist a dynamic balancing algorithm for point quadtrees that is analogous to the AVL algorithm [1] for binary trees, the depth of a point quadtree is bounded from above by the number of nodes in it. On the other hand, we can view PR quadtrees as yielding an average-case balancing under the assumption that future data is equally likely in any of the four quarters (i.e., quadrants) of the square.

The final approach to using quadrees for representing polygonal maps is presented in Section 4.4 and is termed a PM quadtree. This approach derives from allowing a leaf node in the quadtree to contain a complicated structure, instead of just a numeric value. The precise complexity of the structure stored at a leaf varies among the three PM quadrees introduced in this final section. This approach can be viewed as producing an augmented PR quadtree of the vertices in the polygonal map. Its analysis is presented in terms of the relative distances between vertices in the map.

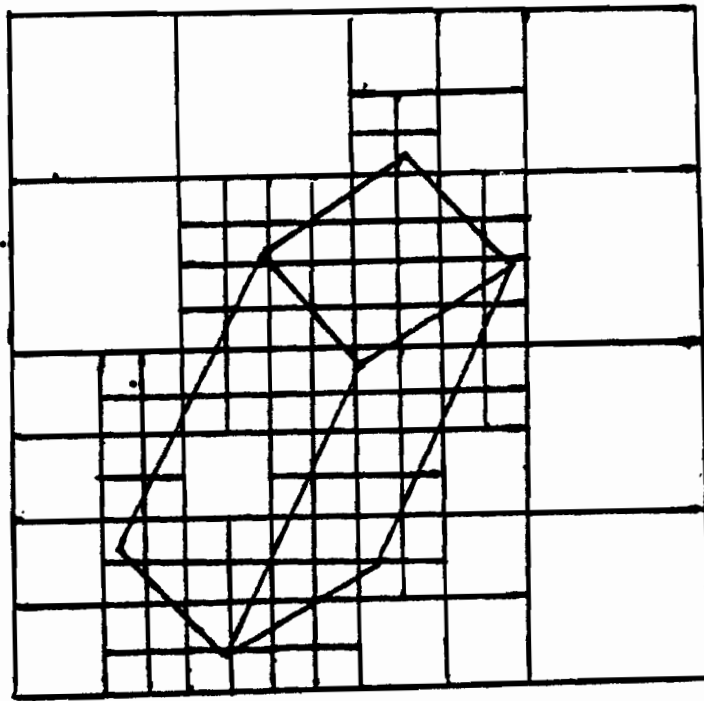


Figure 1-1: An unbiased decomposition of the viewing window

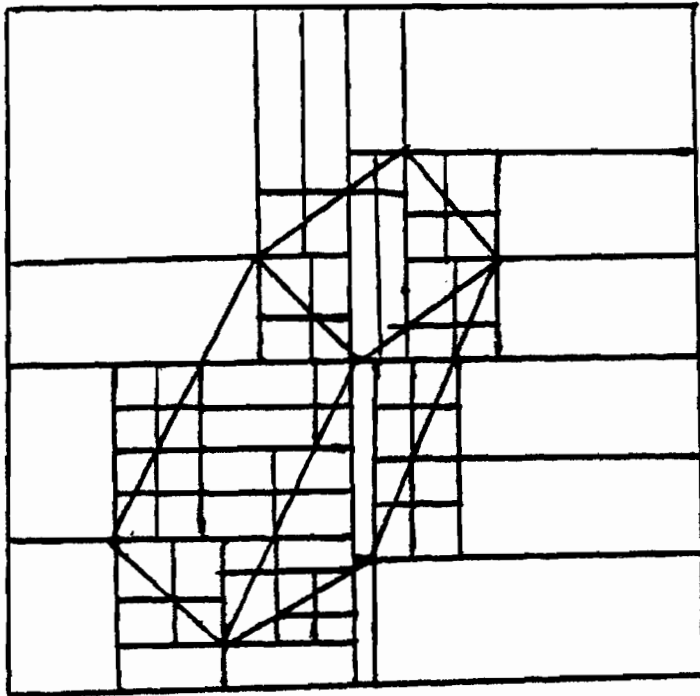


Figure 1-2: A biased decomposition of the viewing window

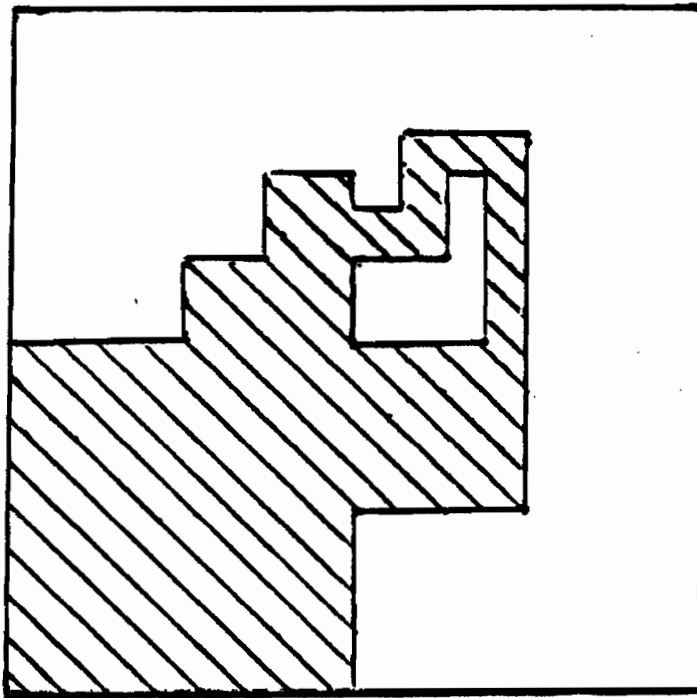


Figure 1-3: A sample image

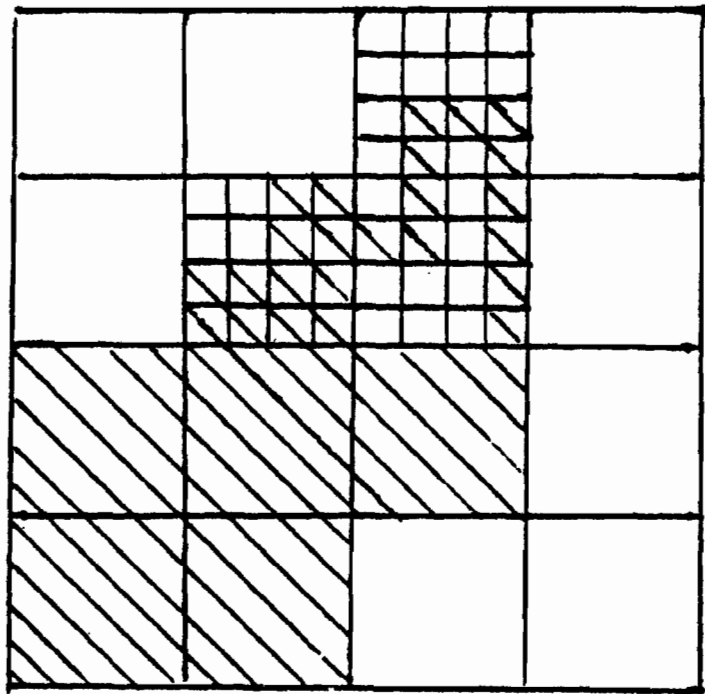


Figure 1-4: 4x4 decomposition of image in Figure 1-3

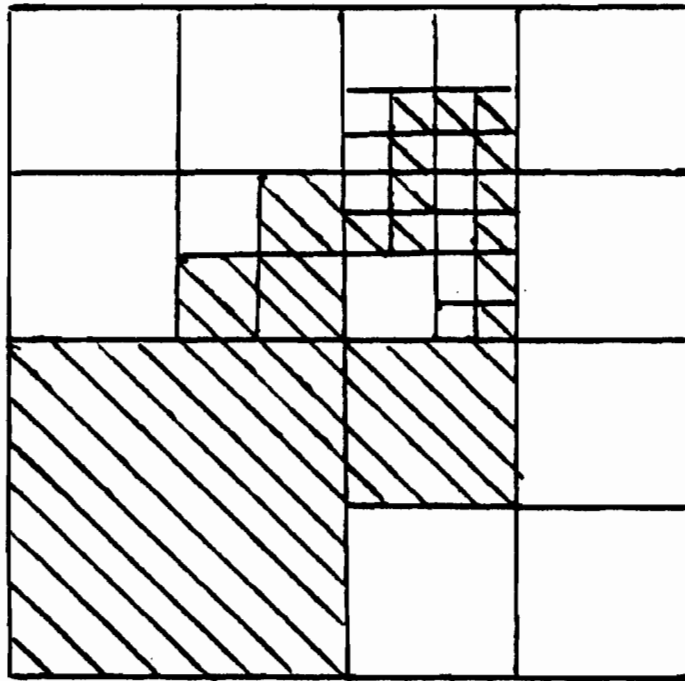


Figure 1-5: A quadtree decomposition of Figure 1-3

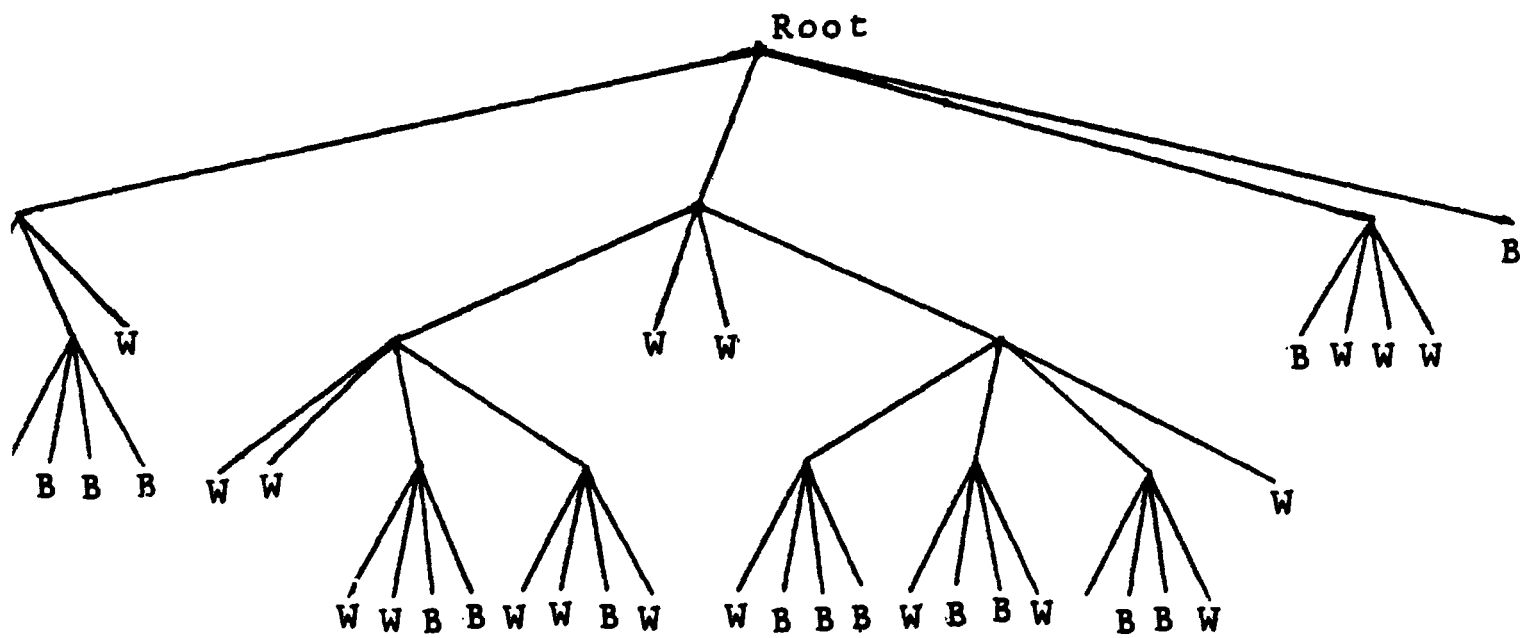


Figure 1-6: Pointer representation of Figure 1-5

NW.NW (W)
 NW.NE (W)
 NW.SE.NW (W)
 NW.SE.NE (B)
 NW.SE.SE (B)
 NW.SE.SW (B)
 NW.SW (W)
 NE.NW.NW (W)
 NE.NW.NE (W)
 NE.NW.SE.NW (W)
 NE.NW.SE.NE (W)
 NE.NW.SE.SE (B)
 NE.NW.SE.SW (B)
 NE.NW.SW.NW (W)
 NE.NW.SW.NE (W)
 NE.NW.SW.SE (B)
 NE.NW.SW.SW (W)
 NE.NE (W)
 NE.SE (W)
 NE.SW.NW.NW (W)
 NE.SW.NW.NE (B)
 NE.SW.NW.SE (B)
 NE.SW.NW.SW (W)
 NE.SW.NE.NW (W)
 NE.SW.NE.NE (B)
 NE.SW.NE.SE (B)
 NE.SW.NE.SW (W)
 NE.SW.SE.NW (W)
 NE.SW.SE.NE (B)
 NE.SW.SE.SE (B)
 NE.SW.SE.SW (W)
 NE.SW.SW (W)
 SE.NW (B)
 SE.NE (W)
 SE.SE (W)
 SE.SW (W)
 SW (B)

Figure 1-7: Dewey Decimal representation of Figure 1-5

G G W W G W B B B W G G W W G
W W B B G W W B W W W G G W B
B B G W B B W G W B B W W G B
W W W B

Figure 1-8: DF-expression representation of Figure 1-5

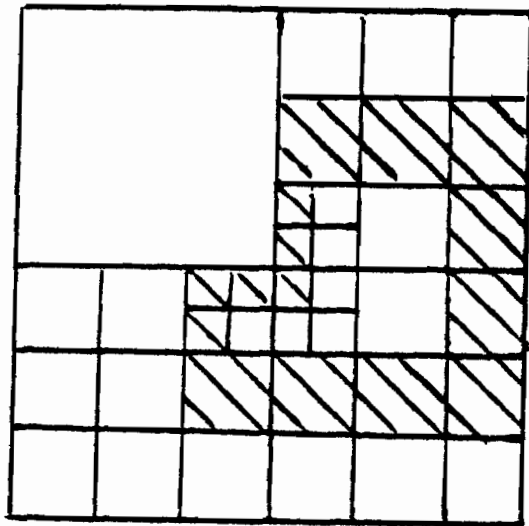


Figure 1-9: Four/Nine/Four decomposition of a 12x12 image

NW (W)
 NE.NW (W)
 NE.NC (W)
 NE.NE (W)
 NE.EC (B)
 NE.SE (B)
 NE.SC (W)
 NE.SW.NW (B)
 NE.SW.NE (W)
 NE.SW.SE (W)
 NE.SW.SW (B)
 NE.WC (B)
 NE.CC (B)
 SE.NW.NW (B)
 SE.NW.NE (W)
 SE.NW.SE (W)
 SE.NW.SW (W)
 SE.NC (W)
 SE.NE (B)
 SE.EC (B)
 SE.SE (W)
 SE.SC (W)
 SE.SW (W)
 SE.WC (B)
 SE.CC (B)
 SW.NW (W)
 SW.NC (W)
 SW.NE.NW (B)
 SW.NE.NE (B)
 SW.NE.SE (W)
 SW.NE.SW (B)
 SW.EC (B)
 SW.SE (W)
 SW.SC (W)
 SW.SW (W)
 SW.WC (W)
 SW.CC (W)

Figure 1-10: Dewey Decimal representation corresponding to Figure 1-9

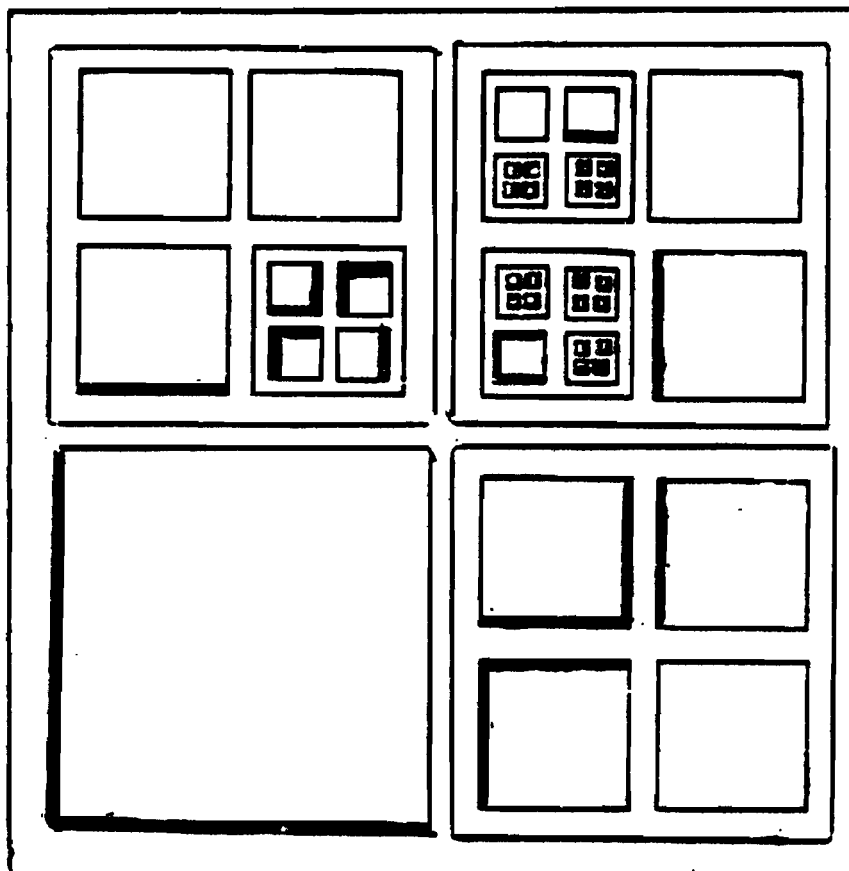


Figure 1-11: Line quadtree equivalent of Figure 1-5

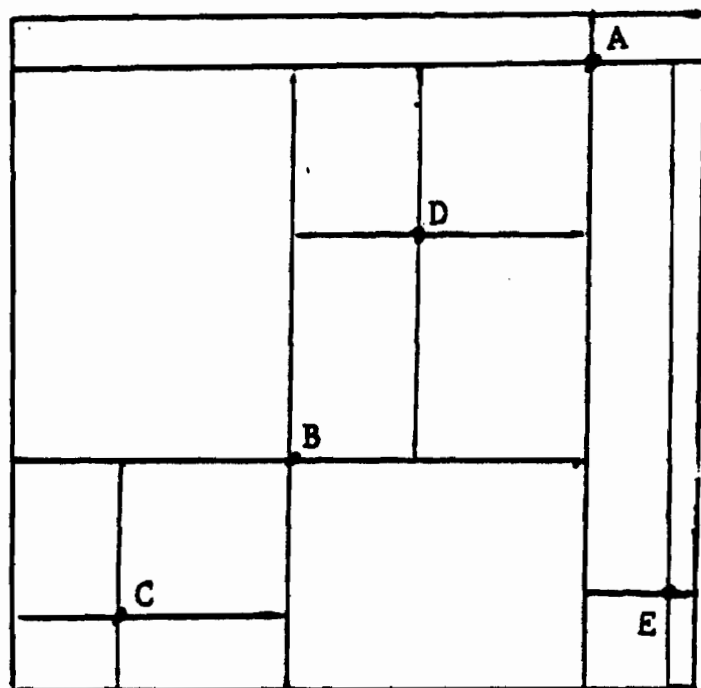


Figure 1-12: Example of a point quadtree decomposition

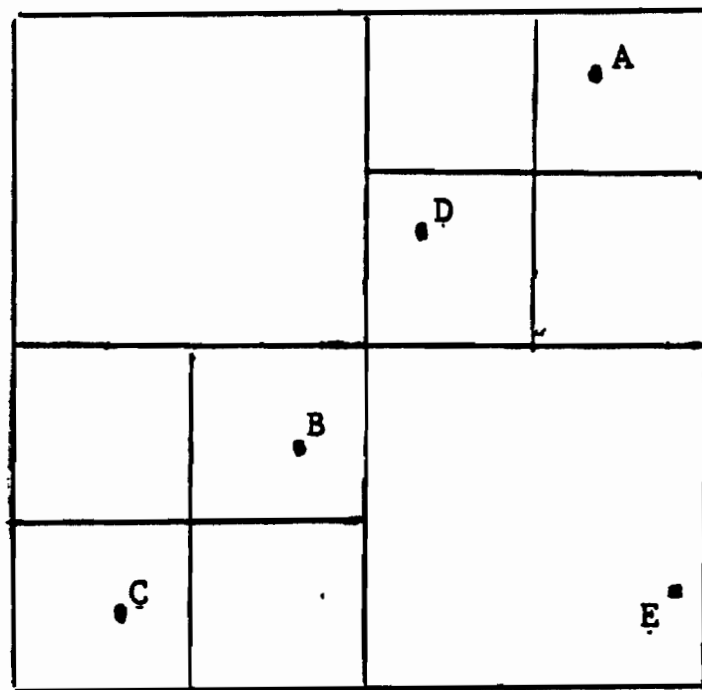


Figure 1-13: PR quadtree equivalent of Figure 1-12

CHAPTER 2

A FORMAL BASIS FOR A THEORY OF QUADTREE ALGORITHMS

2.1. PRELIMINARY DEFINITIONS AND THEOREMS

In this section, definitions are given for the basic concepts that are implicit in any discussion of quadtree algorithms. The most fundamental of these concepts is the concept of the neighbor structure of the nodes of a quadtree. This concept is presented in terms of a set of axioms that implicitly define the various neighbor relations over a domain of elements collectively referred to as a frame. Many properties of this neighbor structure are demonstrated.

In Section 2.2, the notion of a frame is used as a basis for presenting other objects that will concern us in later parts of this thesis. The following definitions are given to suggest the form that the corresponding full definition will take in subsequent sections. A picture is defined as a mapping between the elements of a frame and the elements of a set of colors. A pyramid is then viewed as an extension of the concept of a picture where colors can be associated with subframes, as well as frame elements. The properties of the pyramid follow from restrictions placed on the mapping of subframes to colors. A quadtree can then be defined as a subset of the ordered pairs that form the pyramid color mapping, such that the mapping can be reconstructed from the subset and the general properties of pyramids.

From such definitions, the formal properties of quadtrees can be deduced. These properties are subsequently used, both implicitly and explicitly, in the demonstration of the correctness of various quadtree algorithms. The remainder of Section 2 overviews previous investigations of quadtrees and shows how they can be presented in terms of the definitions of this section. Sometimes this means creating proofs for theorems that

had always been implicitly assumed, e.g., Theorem 10 (The Quadtree Representation Theorem). At other times, this means finding entirely different proofs for known theorems, e.g., Theorem 19 (The Quadtree Storage Requirements Theorem).

It is possible to give a formal definition of pictures, pyramids, and quadtrees by referring to systems of 2^i by 2^i matrices. Using such a system, we would define neighbor relationships in terms of arithmetic on the ordered pairs that serve as indices of the elements of the matrices. This would be analogous to defining propositional logic in terms of TRUE and FALSE being 1 and 0, respectively, where AND is multiplication and NOT is subtraction from one. Although feasible, it is inappropriate for two reasons.

First, by presenting quadtrees in terms of matrices, we find the resulting theory is at the wrong level. Generally, the correctness of a quadtree algorithm hinges on such properties as: if $a_{0,0}$, $a_{0,1}$, and $a_{1,1}$ are elements of a matrix, then so is $a_{1,0}$. At the level of naive matrix theory, we simply do not have the machinery to consider such problems. But if we fall back to some set-theoretic definition of matrices, then we are swamped with extraneous details. Such problems may be mere aesthetics to human theorem provers, but have quite drastic consequences for mechanical theorem provers.

The second reason is that the matrix approach does not indicate appropriate ways to generalize the definitions. For example, it would be cumbersome to attempt to present hextrees in terms of matrices; however, the approach followed below would work with minimal adjustments. Also, it would be difficult, using a matrix approach, to indicate similarities between hextrees and quadtrees.

2.1.1. DEFINING NEIGHBOR RELATIONS

In what follows, we present a large collection of axioms that implicitly define the eight neighbor relations on elements of a frame. When dealing with any collection of axioms larger than one, the problem of consistency must be considered. We could take the standard approach of showing that these axioms can be modelled by a subset of arithmetic. However, arithmetic is so complex that this would hardly be enlightening. For the present, we will not bother with a consistency proof. However, in Section 4.1,

a consistency proof will be presented as a by-product of an investigation of how quadrees can be used to represent polygonal maps on the Euclidean plane.

A frame F^i is a set of indices for atomic picture elements (pixels) that form a 2^i by 2^i picture. These indices are taken from the set of strings over the alphabet {NW, NE, SE, SW}. All references to equality of indices are considered to be references to the standard equality of strings. The expression $x.y$ denotes an application of the operation of string concatenation to the strings represented by x and y . The empty string is denoted by λ . The frame F^i is defined recursively as follows:

$$F^0 = \{ \lambda \}$$

$$F^n = \text{prefix}(\text{NW}, F^{n-1}) \cup \text{prefix}(\text{NE}, F^{n-1}) \cup \text{prefix}(\text{SE}, F^{n-1}) \cup \text{prefix}(\text{SW}, F^{n-1})$$

where $\text{prefix}(x, Y) = \{x.y \mid y \in Y\}$. Figure 2-1 shows the layout of indices for the standard representation of the frame F^2 .

The principal operator for extracting subsets of frames is the subframe function, which is defined as follows:

$$\text{subframe}(F^i, x) = \text{prefix}(x, F^{i-\text{length}(x)})$$

This definition is only meaningful when i is greater than or equal to $\text{length}(x)$. Note that $\text{length}(\text{NW.NE})$ is equal to 2, since NW and NE are single letters in alphabet over which the indices are defined. From the above definitions, it follows that the $\text{subframe}(F^i, x)$ is the set of indices in F^i that begin with x .

The maximum depth of the $\text{subframe}(F^i, x)$ refers to the value of i . The root of the $\text{subframe}(F^i, x)$ refers to the value of x . The depth of a subframe refers to the length of the root of that subframe. Figure 2-2 presents a representation of all the subframes of F^2 , where each subframe is labeled by its root.

From our point of view, the most important aspect of the elements of a frame is the structure imposed on them by the notion that some of the elements are neighbors of other elements. This structure is captured by the interactions between four major (side) relations, n, e, s, and w, and four minor (diagonal) relations, nw, ne, se, and sw. These relations are defined with respect to the subframes of a given frame. The question of the relation of frame elements x and y is handled by considering the relation of the subframes $\{x\}$ and $\{y\}$.

The axioms that define the interactions between these relations and subframes are presented in three groups. The relevant axioms for equality are included for the sake of completeness. The first group is shown in Table 2-1, which shows which relations are inverses of each other. The table entries for e and w are viewed as shorthand for the axioms:

$$\begin{aligned} x \ e \ y &\text{ implies } y \ w \ x \\ &\text{and} \\ x \ w \ y &\text{ implies } y \ e \ x \end{aligned}$$

The second group of axioms is shown in Table 2-2. These axioms indicate the manner in which the neighbor relations can be composed. Thus, the entry of $=$ in column w of row e represents the axiom:

$$x \ e \ y \text{ and } y \ w \ z \text{ implies that } x = z$$

which is different from our interpretation of entries in Table 2-1 (i.e., e and w are inverse relations). An asterisk in Table 2-2 indicates that no axiom is being given for that situation.

The third group of axioms shows the manner in which the neighbor relations can be inferred from the subframe naming convention. Let us introduce the notion that $\text{subframe}(F^i, x)$ is a son of $\text{subframe}(F^i, y)$ if and only if the former is a subset of the latter and the length of x is one more than the length of y . Then the axioms R1, R2, and R3 (when taken together with the previous two groups of axioms) will establish the appropriate neighbor relations among siblings (i.e., among sons of the same father). Axioms R4 and R5 will extend these neighbor relations to the other subframes of the same generation.

- R1) $\text{subframe}(F^i, x.NW) \ w \ \text{subframe}(F^i, x.NE)$
- R2) $\text{subframe}(F^i, x.SW) \ s \ \text{subframe}(F^i, x.NW)$
- R3) $\text{subframe}(F^i, x.SE) \ s \ \text{subframe}(F^i, x.NE)$
- R4) $\text{if } \text{subframe}(F^i, x) \ w \ \text{subframe}(F^i, y)$
 $\text{then } \text{subframe}(F^i, x.NE) \ w \ \text{subframe}(F^i, y.NW)$
- R5) $\text{if } \text{subframe}(F^i, x) \ s \ \text{subframe}(F^i, y)$
 $\text{then } \text{subframe}(F^i, x.NE) \ s \ \text{subframe}(F^i, y.SE)$

To see how the above axiom system for neighbor relations can be used, consider Figure 2-1. There, we observe that the region labeled NE.SW is northeast of the

region labeled SW.NE. This corresponds to the formal statement that there is a proof of the relationship:

$$\text{subframe}(F^2, \text{NE.SW}) \text{ ne } \text{subframe}(F^2, \text{SW.NE})$$

One such proof is shown in Table 2-3. Incidentally, the proof shown is the shortest proof for the above relationship, although it is beyond the scope of this thesis to prove that it is. Each row of this table indicates a step of the proof followed by the justification of that step. The notation for the justification is highly abbreviated. For example, the reference to Table 2-1 in the entry of the reason for line number 4 of the proof refers to one of the axioms in the group associated with Table 2-1, in this case:

$$x \text{ s } y \text{ implies } y \text{ n } x$$

It should be noted that such proofs take place in the context of a propositional logic augmented by the axiom schemes presented above. Hence a given instance of a neighbor relationship is true if and only if it is provable.

2.1.2. THEOREMS REGARDING NEIGHBOR RELATIONS

The reason for presenting a formal theory of neighbor relations is that this theory provides a context for later discussion of algorithm correctness. Actually, we will have little use for proofs like the one in Table 2-3, but will instead want to consider theorems from their metatheory. Four such theorems are proved below. These are the Neighbor Uniqueness Theorem, the Neighbor Decidability Theorem, the Subframe Invariance Theorem, and the No-Corner-Cutting Theorem.

The Neighbor Uniqueness Theorem states that two different subframes can not bear the same relation to the same subframe. Among other things, this means that in order to show that $x \text{ s } y$ is false, it is sufficient to show that $x \text{ s } z$ and that y is not equal to z . For example, since we have already proved that

$$\text{subframe}(F^2, \text{NE.SW}) \text{ ne } \text{subframe}(F^2, \text{SW.NE})$$

we can deduce from the Neighbor Uniqueness Theorem that

$$\text{subframe}(F^2, \text{NE.SW}) \text{ ne } \text{subframe}(F^2, \text{SE.NW})$$

is false.

Theorem 1: (The Neighbor Uniqueness Theorem): For every neighbor relation r :

$x \text{ r } y$ and $x \text{ r } z$ implies $y = z$

Proof: Assume that $x \text{ r } y$ and $x \text{ r } z$. Table 2-1 shows that for each neighbor relation r , there exists an inverse relation r^{-1} such that

$x \text{ r } y$ implies $y \text{ r}^{-1} x$

Inspection of Table 2-2 shows that it is always the case that

$y \text{ r}^{-1} x$ and $x \text{ r } z$ implies $y = z$

Thus, the theorem is proved.

We note that the proof of the Neighbor Uniqueness Theorem was not constructive. Thus, we know that a subframe can have at most one neighbor on a given side, but we do not know how to find that one neighbor. The Neighbor Decidability Theorem yields a procedure for identifying that neighbor. However, before we can prove the Neighbor Decidability Theorem, we need the following lemma, which limits the types of inferences that need be used in a proof relating two subframes of F^i . For example, from the proof of the following lemma, it will be clear that the shortest proof of a relationship among subframes of F^i will not contain references to subframes of frames other than F^i .

The proof of the following lemma uses the technique of ordering all the possible proofs in our Neighbor Theory by the number of deductions in each proof. For example, the proof in Table 2-3 is formed from eleven deductions. Within this ordering, it is necessary that proofs involving the same number of deductions also be ordered, but it is not important how they are ordered.

Lemma 2: If for some neighbor relation r , it is provable that

$\text{subframe}(F^i, x) \text{ r } \text{subframe}(F^j, y)$

then $i = j$ and $\text{length}(x) = \text{length}(y)$.

Proof: Consider the shortest proof for a statement S of the form:

$\text{subframe}(F^i, x) \text{ r } \text{subframe}(F^j, y)$

for some neighbor relation r , where either the maximum depths of the subframes differ or the actual depths of the subframes differ. In the following discussion, such subframes are called incomparable. The statement S must be the consequence of the application of one of the axioms of the Neighbor Theory. We will now show that the axiom applied could not have been in any of the three groups that partition the axioms of Neighbor Theory.

The axiom used could not be one of the axioms of the first group,

listed in Table 2-1, because their consequents only relate subframes that were related in their antecedents (and if these subframes were related in the antecedents then this isn't the shortest proof relating two incomparable subframes).

The axiom used could not be one of the axioms of the second group, listed in Table 2-2, because their consequents only relate subframes that were related to a third intermediate subframe in their antecedent. Clearly, if the two subframes of the consequent are incomparable, then one of the subframes must be incomparable to the intermediate subframe of the antecedent (and hence there must exist a shorter proof relating incomparable subframes).

Finally, we consider the third group of axioms, i.e., R1 through R5. The axiom used could not be one of R1 through R3, because each of these axioms asserts a relation between subframes of the same maximum depth and actual depth. Neither could the axiom used be either R4 or R5, because their consequents assert a relation between subframes of equal maximum depth and if the subframes of the antecedent are of equal actual depth then the subframes of the consequent will also be of equal actual depth.

Thus, there is no axiom that will allow the deduction of a neighbor relation between incomparable subframes, unless there is a proof relating incomparable subframes that is shorter than the one for S , which was assumed to be the shortest such proof. Therefore, the lemma is true.

If we consider the axioms R1 through R3, we find that there are an infinite number of subframe relations that are being asserted by these axioms. However, for any fixed maximum depth, there are only a finite number of assertions about relations among subframes of that maximum depth. The proof of the above lemma shows that in deciding whether or not two subframes are related, only the assertions about subframes of the same maximum depth as those in question are relevant to the decision (proof). Now, we are ready to prove the Neighbor Decidability Theorem. The Neighbor Decidability Theorem states that for any instance of a neighbor relationship, we can decide whether or not that relationship holds. Thus, instead of having humans grind out proofs like the one in Table 2-3, it is possible to have computers do it.

Theorem 3: (The Neighbor Decidability Theorem): There is an effective algorithm (i.e., an algorithm that always halts) such that given any two subframes, x and y , and a neighbor relation r , the algorithm returns true iff the conjecture $x \ r \ y$ is provable.

Proof: By induction on i , it can be shown that the size of F^i and the number of subframes of F^i are both finite numbers. Let S denote the set of all statements that assert neighbor or equality relations among subframes of F^i (for the value of i relevant to the conjecture to be decided). Clearly, S is finite.

Lemma 2 stated that the only neighbor relations relevant to a decision procedure are those that use the same frames as those that are in the subframes that are to be proved related. If x and y are of different maximum depths, then by Lemma 2, we know immediately to return false.

Now, let us construct the decision process. Let the current state of the decision process be that subset of S which has been proved so far. The initial state of the decision process is the empty set. The successor state of the current state is the union of the current state with the set of relations that can be deduced from the current state by exactly one application of an axiom. The calculation of the successor state from the current state is clearly effective. This is because only axioms $R1$ through $R3$ can generate an unbounded number of consequences from a finite number of antecedents. However, of this unbounded number of consequences, only a finite number of them need be considered as relevant to the decision procedure.

An algorithm that starts with the initial state as its current state and then continually replaces the current state with its successor state until the successor state equals the previous current state constitutes the core of the decision process. It would eventually halt, because the successor state can never be smaller than the current state nor larger than S . When the algorithm halts, the current state will be the set of all statements that are provable and assert neighbor relations or equality relations among the subframes of F^i .

The remaining task for the decision procedure is to check whether the conjecture is a member of the current state when the above algorithm halted. This is also effective since the current state must be of finite size. Thus, there is an effective decision algorithm for Neighbor Theory.

To prove the above theorem, a decision procedure that runs in time exponential with respect to i , on subframes of F^i , was developed. In Section 2.4, the correctness of an algorithm [25], which could be used as a basis for a decision procedure that runs in time linear with respect to i (on subframes of F^i), will be shown.

Although it is true, as noted in Lemma 2, that there can not exist a neighbor relationship between two subframes with different maximum depths, there is a certain parallelism between the properties of subframes that have the same roots. For example, since we have already proven that

$$\text{subframe}(F^2, \text{NE.SW}) \text{ ne } \text{subframe}(F^2, \text{SW.NE})$$

we would like to be able to immediately infer

$$\text{subframe}(F^3, \text{NE.SW}) \text{ ne } \text{subframe}(F^3, \text{SW.NE})$$

as indicated in Figure 2-3. This ambition is justified by the following theorem.

Theorem 4: (The Subframe Invariance Theorem): For all neighbor relations r , if i is not less than $\text{length}(x)$ then

$$\text{subframe}(F^i, x) \text{ r } \text{subframe}(F^i, y) \\ \text{iff}$$

$$\text{subframe}(F^{i+1}, x) \text{ r } \text{subframe}(F^{i+1}, y)$$

Proof: By Lemma 2, we know that if $\text{length}(x)$ is not equal to $\text{length}(y)$, then both sides of the above "iff" construct are false, which makes the theorem true. Also, Lemma 2 shows that the shortest proof of each of the statements (should they be provable) can be constructed with all references to subframes restricted to those that are the same depth as that are in the statement. In the following, we show that if one side of the "iff" is provable, then so is the other side. We always assume that we are working with the shortest possible proof of the given relationship. Now, let us consider the two parts of this theorem.

In the "if" direction, we can take the proof of the antecedent and substitute " F^{i+1} " for every occurrence of " F^i ". We note that all the axioms hold true under this substitution, because if x is a valid root for a subframe of F^i , then it is also a valid root for a subframe of F^{i+1} . Thus, the result of this substitution is a proof of the consequent.

In the "only if" direction, we take the proof of the antecedent (which was the consequent in the "only if" direction) and substitutes " F^i " for every occurrence of " F^{i+1} ". The question then arises of whether this substitution has invalidated any of the steps of the proof. The answer is no, because our working with the shortest proof of

$$\text{subframe}(F^{i+1}, x) \text{ r } \text{subframe}(F^{i+1}, y)$$

implies that there were no references to subframes with roots longer than the length of x . This is because relations on such subframes could not be part of the deduction chain leading to the statement, by reasoning analogous to that of the proof of Lemma 2. Since we assumed we had the shortest proof, we know that such superfluous references will not be in it.

Thus, both cases of the Subframe Invariance Theorem hold.

In order to show the correctness of the neighbor-finding algorithms of Section 2.4, it will be necessary to use some results that are stronger than the ones shown above. The problem is that the axioms of Neighbor Theory force deductions to flow in a top-down manner, whereas some of the algorithms work in a bottom-up fashion. Thus, it is necessary to show not only that if the father has a northern neighbor, then so will all his sons (as is shown by the Positive Dominant Lemma) but to also show that if the father does not have a northern neighbor, then neither will two of his sons (as we will eventually show in the Negative Recessive Lemma). These lemmas form part of the reasoning justifying the No-Corner-Cutting Theorem, which states (among other things)

that if a subframe has a northwest neighbor, then it also must have a northern neighbor.

The first lemma that we will consider is the Positive Dominant Lemma. This lemma states that a subframe inherits the property of having a neighbor on a given side from its father. Thus, if a subframe has a northern neighbor, all of its sons will have northern neighbors. The proof is constructive in that it not only shows that the northern neighbor exists, but it also shows how to compute the northern neighbor's root given the root of the northern neighbor of the father. This construction is summarized in Table 2-4.

For Table 2-4, and for the following proofs, we introduce a new notation for the sons of a subframe. If x is a subframe whose depth is less than its maximum depth, then if

$$x = \text{subframe}(F^i, a)$$

then the four sons of x are denoted

$$x^{NW} = \text{subframe}(F^i, a.NW)$$

$$x^{NE} = \text{subframe}(F^i, a.NE)$$

$$x^{SE} = \text{subframe}(F^i, a.SE)$$

$$x^{SW} = \text{subframe}(F^i, a.SW)$$

If x has a father, then it is denoted x' .

Lemma 5: (The Positive Dominant Lemma): If, for some side relation r , it is provable that

$$x \ r \ y$$

and x' is a son of x , then there must exist z such that

$$x' \ r \ z$$

Proof: The son of x' , according to our definition of son, must be one of the following: x^{NW} , x^{NE} , x^{SE} , and x^{SW} . The proof for this lemma can be constructed by considering each of the sixteen cases resulting from the possible choices of the side relation r and the son x' . Table 2-4 shows the appropriate value for z for each of these cases.

For example, to take the most difficult case, Table 2-4 indicates that for r equal to e and x' equal to x^{SW} , we can use y^{SE} for z . This is equivalent to the statement

$$x \ e \ y \text{ implies } x^{SW} \ e \ y^{SE}$$

To justify this, we first demonstrate that

$$x \text{ e } y \text{ implies } x^{NW} \text{ e } y^{NE}$$

using axiom R4. Next, we show that

$$x^{SW} \text{ s } x^{NW}$$

and

$$y^{NE} \text{ n } y^{SE}$$

using axioms R2 and R3, respectively. Then, using the axioms of Table 2-2, we chain the above together to form the following sequence of deductions:

$$\begin{aligned} & x \text{ e } y \text{ implies } x^{NW} \text{ e } y^{NE} \\ & x^{SW} \text{ s } x^{NW} \text{ and } x^{NW} \text{ e } y^{NE} \text{ implies } x^{SW} \text{ se } y^{NE} \\ & x^{SW} \text{ se } y^{NE} \text{ and } y^{NE} \text{ n } y^{SE} \text{ implies } x^{SW} \text{ e } y^{SE} \end{aligned}$$

which yield:

$$x \text{ e } y \text{ implies } x^{SW} \text{ e } y^{SE}$$

The proofs of the other cases are trivial modifications of the one shown above. Thus, the lemma is demonstrated.

The above lemma shows that a son will always have a neighbor on a given side, if his father does. The situation when that father does not have a neighbor on a given side is analyzed by the Negative Recessive Lemma. That lemma claims that if the father does not have a neighbor on a given side, then two of his sons will not have a neighbor on that side either. This is proved by using the technique of embedding a collection of subframes in a larger collection where the appropriate subframe now has a neighbor on the given side. Next, it is shown that for two of its sons, if they have neighbors on the given side in the original collection, then they would now have two distinct neighbors on the same side. Since this would contradict the Neighbor Uniqueness Theorem, the two sons must not have had neighbors in the original collection. To set all this up, we need to prove the Embedding Lemma, which describes how the above embedding works.

Later, when we get around to proving properties of pyramids and quadrees, this type of embedding will again prove useful.

Lemma 6: (The Embedding Lemma): For each x in $\{NW, NE, SE, SW\}$ and for each neighbor relation r , if

$$\text{subframe}(F^i, y) \text{ r } \text{subframe}(F^i, z)$$

then

$$\text{subframe}(F^{i+1}, x, y) \text{ r } \text{subframe}(F^{i+1}, x, z)$$

Proof: Consider the shortest proof of

$$\text{subframe}(F^i, y) \text{ } r \text{ } \text{subframe}(F^i, z)$$

Without loss of generality, let us assume that x (of the above lemma) is NW. Now, replace each subframe's root, appearing in the proof, with the concatenation of NW and that root. Also, change every subframe's depth from i to $i+1$. Recall that all subframes in the shortest proof of a neighbor relationship will have the same depth. It is claimed that this substitution has produced a valid proof of

$$\text{subframe}(F^{i+1}, \text{NW}.y) \text{ } r \text{ } \text{subframe}(F^{i+1}, \text{NW}.z)$$

The above claim results from consideration of the effect that the same substitution has when applied to each of the axioms. First, we note, from the definition of a subframe, that y is a valid root for a subframe of depth i iff $\text{NW}.y$ is a valid root of a subframe of depth $i+1$; since

$$\text{length}(\text{NW}.y) = 1 + \text{length}(y)$$

Next, we note that the result of each substitution is a theorem that is a special case of the axiom on which the substitution was made. For example, the result of performing the substitution on axiom R1 yields

$$\text{subframe}(F^{i+1}, \text{NW}.x.\text{NW}) \text{ } w \text{ } \text{subframe}(F^{i+1}, \text{NW}.x.\text{NE})$$

which is a special case of

$$\text{subframe}(F^i, x.\text{NW}) \text{ } w \text{ } \text{subframe}(F^i, x.\text{NE})$$

Thus, the substitution yields a valid proof of the consequent of the lemma.

Now, we can combine the Neighbor Uniqueness Theorem with the above lemma, in the manner described above, to prove that certain subframes do not have neighbors on certain sides. This lemma, the Negative Recessive Lemma, together with the Positive Dominant Lemma, shows how the property of having a neighbor on a given side is inherited by the sons from the father. For example, the Negative Recessive Lemma is used in Section 2.4 to show that in the frame F^i , there are 2^i indices that correspond to subframes without northern neighbors (see Lemma 23). Like the proof of the Positive Dominant Lemma, the following proof is constructive, i.e., we show that two sons will not have neighbors on a particular side by showing how to determine which two sons will not have neighbors.

Lemma 7: (The Negative Recessive Lemma): For each side relation r , if $\text{subframe}(F^i, x)$ does not have a neighbor in the direction r , then two of the sons of $\text{subframe}(F^i, x)$ will not have neighbors in the direction r .

Proof: Without loss of generality, let us assume that the side relation under consideration is n . First, we note that

$$\text{subframe}(F^i, x.\text{NW}) \text{ } n \text{ } \text{subframe}(F^i, x.\text{SW})$$

and

$$\text{subframe}(F^i, x, \text{NE}) \text{ n } \text{subframe}(F^i, x, \text{SE})$$

can be proved without reference to any nodes other than $\text{subframe}(F^i, x)$ and his sons. Thus, we are left with showing that neither $\text{subframe}(F^i, x, \text{NW})$ nor $\text{subframe}(F^i, x, \text{NE})$ have northern neighbors. Recalling Table 2-4, we note that if x had a northern neighbor y , then

$$\text{subframe}(F^i, y, \text{SW}) \text{ n } \text{subframe}(F^i, x, \text{NW})$$

and

$$\text{subframe}(F^i, y, \text{SE}) \text{ n } \text{subframe}(F^i, x, \text{NE})$$

Let us now work only with $\text{subframe}(F^i, x, \text{NW})$ as the reasoning for the other subframe follows in much the same manner.

Since the father of x does not have a northern neighbor, it follows from the Positive Dominant Lemma that none of the ancestors of x have northern neighbors. Now, embed all the subframes of F^i in SW (i.e., perform the substitutions of the Embedding Lemma). Note that from the Positive Dominant Lemma and Table 2-4, we can deduce that there exists a z such that

$$\text{subframe}(F^{i+1}, \text{NW}, z, \text{SW}) \text{ n } \text{subframe}(F^{i+1}, \text{SW}, x, \text{NW})$$

Now, if $\text{subframe}(F^i, x, \text{NW})$ had had a northern neighbor y' , then according to the Embedding Lemma

$$\text{subframe}(F^{i+1}, \text{SW}, y') \text{ n } \text{subframe}(F^{i+1}, \text{SW}, x, \text{NW})$$

However, this contradicts the Unique Neighbor Theorem. Therefore, $\text{subframe}(F^i, x, \text{NW})$ has no northern neighbor. Similarly, $\text{subframe}(F^i, x, \text{NE})$ does not have a northern neighbor. Since each row of Table 2-4 has two entries that refer to sons of y , it follows that the above demonstration will work for any side relation.

Now, we are in a position to prove the No-Corner-Cutting Theorem. The proof of the No-Corner-Cutting Theorem is presented in two parts. The first part, Lemma 8, considers what can be deduced from the fact that there exists a proof of a particular diagonal relationship. Specifically, it shows that the last deduction in the shortest proof of such a relationship must have one of two forms. The second part of the proof, which is presented as the proof of Theorem 9 (the No-Corner-Cutting Theorem), shows that for whichever form the last deduction of the shortest proof takes, there exists another proof that reaches the same conclusion using the other form. Thus, for example, if the shortest proof of $x \text{ ne } z$, in Figure 2-3, makes use of a northern neighbor y of z , then there exists another proof of $x \text{ ne } z$ that uses an eastern neighbor y' of z . The lemma below states that if $x \text{ ne } z$, then the shortest proof of x

ne z requires that z have either an eastern or a northern neighbor, but we don't know which.

Lemma 8: If p and q are side relations and r is a diagonal relation such that:

$$x p y \text{ and } y q z \text{ implies } x r z$$

then $x r z$ implies that there exists a y such that either

$$x p y \text{ and } y q z$$

or

$$x q y \text{ and } y p z.$$

Proof: Consider the shortest proof of a statement of the form $x r z$ where x , z , p , q , and r satisfy the conditions of this lemma, but there does not exist the implied value of y . The final statement in the proof is $x r z$, which can only be a drawn conclusion from the usage of an inverse axiom (from Table 2-1) or a composition axiom (from Table 2-2), since R1 through R5 do not mention diagonal relations.

Assume that the final statement was deduced from a usage of an inverse axiom. Then there exists a shorter proof that proves $z r^{-1} x$. Inspection of the composition axioms shows that whenever

$$x p y \text{ and } y q z \text{ implies } x r z$$

then

$$z q^{-1} y \text{ and } y p^{-1} x \text{ implies } x r^{-1} x$$

Since $z r^{-1} x$ has a shorter proof that $x r z$, it must be true that z , x , q^{-1} , p^{-1} , and r^{-1} satisfy the conditions of this lemma. This means that there exists a y such that

$$z q^{-1} y \text{ and } y p^{-1} x$$

This same y satisfies

$$x p y \text{ and } y q z$$

thus, contradicting the assumption that y did not exist. Therefore the final statement could not have been deduced from an inverse axiom.

Assume that the final statement was deduced from a usage of a composition axiom. For each diagonal relation r , there are exactly two side relations p and q that could satisfy the conditions of the lemma. The relevant axioms have the form

$$x p y \text{ and } y q z \text{ implies } x r z$$

or

$$x q y \text{ and } y p z \text{ implies } x r z$$

In both cases there must exist a y that satisfies the consequence of this lemma, because there had to be a y that satisfied the antecedent of one of these axioms so that it could be used in the proof. This contradicts the assumption that there exists values of x , z , p , q , and r that contradict the lemma. Since the negation of the lemma always leads to a contradiction, the lemma must be true.

We can now complete our proof of the No-Corner-Cutting Theorem.

Theorem 9: (The No-Corner-Cutting Theorem): If p and q are side relations and r is a diagonal relation such that

$$x p y \text{ and } y q z \text{ implies } x r z$$

then $x r z$ implies there exists a y such that

$$x p y \text{ and } y q z.$$

Proof: Given that

$$x p y \text{ and } y q z \text{ implies } x r z$$

and

$$x r z$$

we know from Lemma 8 that there exists a y such that either

$$x p y \text{ and } y q z$$

or

$$x q y \text{ and } y p z$$

If the first case holds in a given situation, then our theorem is true. Thus, we are left with considering the situation where the construction of Lemma 8 yields a y such that

$$x q y \text{ and } y p z$$

A simple enumeration of the possibilities will show that the theorem holds for all the subframes of F^0 and for all the subframes of F^1 . Let us consider the smallest j such that for two subframes of F^j , the theorem does not hold for some assignment of values to x , z , p , q , and r . Without loss of generality, let us assume that p , q , and r are n , e , and ne , respectively. Thus, we are assuming that there exists a y such that

$$x e y \text{ and } y n z$$

but there does not exist a y' such that

$$y' e z$$

Note that $x n y'$ would follow from the immediately above together with $x ne z$. By follows immediately, we mean:

$$\begin{aligned} x ne z \text{ implies } z sw x \\ y' e z \text{ and } z sw x \text{ implies } y' s x \\ y' s x \text{ implies } x n y' \end{aligned}$$

Since z does not have an eastern neighbor, it follows from the Negative Recessive Lemma that since z^f is the father of z , either

$$z = (z^f)^{SE}$$

or

$$z = (z^f)^{NE}$$

The first of these cannot be true because then

$$y' = (z^f)^{NE}$$

which would imply that z must have an eastern neighbor since the only way

the northeastern son of z^f , i.e., y' , can have an eastern neighbor is if z^f has an eastern neighbor which in turn implies that the southeastern son of z^f , i.e., z , would also have an eastern neighbor. Thus, we know that in order for the theorem to be false, it must be that z is the northeastern son of its father.

Since z is the northeastern son of its father, it follows from the Negative Recessive Lemma that since $y' \cap z$, z^f must have a northern neighbor, y'^f , which in turn has y' as a son. In particular, it follows from Table 2-2 that

$$y' = (y'^f)^{SE}$$

When this fact is combined with $x \in y'$, using reasoning analogous to that which has just been used, it follows that y'^f also has a eastern neighbor x'^f . Now, from the axioms of Table 2-2, this means that

$$x^f \cap z^f$$

which under the Subframe Invariance Theorem implies a corresponding relationship in F^{j-1} . By combining this relationship with the assumption that j was the first frame depth to contradict the No-Corner-Cutting Theorem and another usage of the Subframe Invariance Theorem, we get the result that there exists a y'' such that

$$x^f \cap y'' \text{ and } y'' \in z^f$$

This, because of the Positive Dominant Lemma, contradicts the assumption that z did not have an eastern neighbor.

Since analogous reasoning produces the same result for the other possible combinations of side relations, it follows that there can not be a least i , such that the No-Corner-Cutting Theorem does not hold among the subframes of F^i . Thus, by the well-ordering of the positive integers, it follows that the No-Corner-Cutting Theorem holds for all subframes of all frames.

We have now concluded our presentation of Neighbor Theory. In the following section, we will turn to the question of the computer representation of images whose constituent parts satisfy the axioms of Neighbor Theory.

2.2. THE DEFINITION OF TWO TYPES OF QUADTREES

Although the neighbor structure is the most fundamental concept involved in the theory of quadrees, it is not the only one. There is also the concept of the association of a color with an element of a frame. In discussing colors, we have to consider two possibilities. If the coloring is binary, then we choose our colors from elements of the set {BLACK, WHITE}. If the coloring is multicolored, then we choose

our colors from elements of a set that contains a distinguished member who is denoted by the color WHITE.

2.2.1. DEFINITION OF PICTURES AND THEIR PROPERTIES

We define a picture P as an ordered triple $\langle F^i, C, f \rangle$ consisting of a frame F^i , a set C of colors, and a coloring function f . The coloring function f is a total function from F^i into the set C of colors. It is also sometimes useful to view the picture P as the set

$$\{ \langle x, f(x) \rangle \mid x \in F^i \}$$

which is simply an alternative representation of the function f . Figure 2-4 shows how we represent a binary picture, P , where

$$\begin{aligned} P &= \langle F', \{ \text{BLACK}, \text{WHITE} \}, f \rangle \\ f &= \{ \langle \text{NW}, \text{WHITE} \rangle, \langle \text{NE}, \text{BLACK} \rangle, \langle \text{SE}, \text{BLACK} \rangle, \langle \text{SW}, \text{WHITE} \rangle \} \end{aligned}$$

One of the most basic operations on a picture is to embed it in a larger frame. The simplest embedding is defined as follows. Let x_1 denote one of $\{ \text{NW}, \text{NE}, \text{SE}, \text{SW} \}$. Let x_2, x_3 , and x_4 denote the remaining three elements. Then, we say

$$\text{embed}(x_1, \langle F^i, C, f \rangle) = \langle F^{i+1}, C, f' \rangle$$

where for each y in F^i

$$\begin{aligned} f'(x_1, y) &= f(y) \\ f'(x_2, y) &= \text{WHITE} \\ f'(x_3, y) &= \text{WHITE} \\ f'(x_4, y) &= \text{WHITE} \end{aligned}$$

The subframe(F^{i+1}, x_1) is said to be the projection of the picture $\langle F^i, C, f \rangle$. Thus, for example, Figure 2-5 shows $\text{embed}(\text{NW}, P)$ where P is the binary picture in Figure 2-4. The upper left hand quarter of Figure 2-5 is the subframe that corresponds to the projection of the picture of Figure 2-4. The justification for coloring the pixels that are not part of the projection of the original picture WHITE is that we are generally only interested in the pixels that have a color other than WHITE.

One of the problems with manipulating quadrees is that there is a set of nodes that do not have neighbors on one of their sides. Sometimes this boundary condition problem can be avoided by surrounding the nodes of interest with other nodes that

need not be further considered. We will see one usage of this technique when we later define shifting. We can define a function, based on embed, that performs this transformation as follows:

$$\text{surround}(\langle F^i, C, f \rangle) = \text{embed}(\text{NW}, \text{embed}(\text{SE}, \langle F^i, C, f \rangle))$$

The importance of surround comes from the fact that by the Positive Dominant Lemma, each element of the projection of the original picture $\langle F^i, C, f \rangle$ now has a neighbor on each side. In Figure 2-6, we find a representation of $\text{surround}(P)$ where P is the binary picture of Figure 2-4. Note that the lower right hand quarter of the upper left hand quarter of Figure 2-6 corresponds to the projection of the picture of Figure 2-4.

The final type of embedding that we will consider is the shift. The following definition of a shift is a bit complicated; so let us first look at what goes wrong with some simpler definitions. Suppose we defined a shift to be an isomorphism from one frame to another. This is unsatisfactory because it does not preserve any of the neighbor structure, as illustrated by Figure 2-7. The next step would be to say that a shift is an isomorphism between two frames such that all the neighbor relations are preserved. This is too confining. Indeed, only the original picture satisfies this condition. Next, we might say that a shift is an isomorphism that preserves neighbor relations only among the non-White pixels. This permits too much, as illustrated by Figure 2-8. Also, there is the fact that it is always meaningful to shift into a larger frame. Taking all of this into account, we arrive at the following definition.

We say that $\langle F^{i+j}, C, f' \rangle$ is a shift of $\langle F^i, C, f \rangle$ into F^{i+j} iff the following four conditions hold. Note that the mechanics of this definition are illustrated in Figure 2-9.

1. There exists a subset I , known as the image, of the frame of S where

$$S = \langle F^{2^{i+j}}, C, f' \rangle = \text{surround}^{j+1}(\langle F^i, C, f \rangle)$$

2. The image I must contain all the pixels in S that are not WHITE.
3. Further, there exists a 1-1 onto mapping p from the image I onto F^{i+j} , such that for every x and y in F^{i+j} , for every side relation r , if

$$x \text{ } r \text{ } y$$

then

$$p^{-1}(x) \text{ r } p^{-1}(y)$$

4. The last, but not least, condition is that for all x in F'^{-j}

$$f''(x) = f'(p^{-1}(x))$$

Note that the above definition is intended for use within a formally developed system. Other approaches to the definition of shift can be justified by proving equivalence to this definition. For example, it could be shown that an equivalent definition of shift could be presented in terms of fewer applications ($j/2$ instead of j) of the function surround, at the expense of introducing some minor complications.

When the shift is from $\langle F', C, f \rangle$ into $\langle F'', C, f'' \rangle$ i.e., when j is zero, there is the notion of the distance of a shift with respect to a direction. Note that the $\text{surround}(\langle F', C, f \rangle)$ equals $\langle F'^{-2}, C, f' \rangle$ where for each element x of the frame F'

$$f'(\text{NW}, \text{SE}, x) = f(x)$$

Using the projection function p from the above definition of a shift, if for each x in F''

$$\begin{aligned} p^{-1}(x) \text{ r } (\text{NW}, \text{SE})^{1+(j/2)} . x & \quad \text{if } j \text{ is even} \\ p^{-1}(x) \text{ r } (\text{NW}, \text{SE})^{(j/2)} . \text{NW} . x & \quad \text{if } j \text{ is odd} \end{aligned}$$

then the shift is said to be of distance 1 in direction r . Figure 2-10 shows a shift of distance 1 in the western direction for the picture in Figure 2-4.

For many basic picture properties, a picture and the shift of a picture have the same value. Two such properties that come readily to mind are the area of a picture and the length of the perimeter of a picture. The area of a picture is the number of pixels in the picture that are not WHITE. Thus, the area of the picture in Figure 2-4 is 2. The length of the perimeter of a picture P is the size of the set

$$\begin{aligned} \{ \{x, y\} \mid & x \text{ and } y \text{ are in } \text{surround}(P) \text{ and} \\ & x \text{ r } y \text{ for some side relation } r \text{ and} \\ & \text{the color of } x \text{ is not the color of } y \} \end{aligned}$$

Thus, the length of the perimeter of the picture in Figure 2-4 is 6. A pixel x is said to be adjacent to the perimeter of a picture P , if either 1) there exists a neighbor on some side of x that has a different color from x , or 2) x is not WHITE and has a side on which it has no neighbor. In Figure 2-4, all the pixels are on the perimeter.

Another useful operation to perform on pictures is to overlay them. The overlay

of picture $\langle F^i, C, f \rangle$ on top of the picture $\langle F^i, C', f' \rangle$ is defined as the picture $\langle F^i, C'', f'' \rangle$ where C'' is the union of C and C' , and for each x in F^i

$$\begin{aligned} f''(x) &= f(x) && \text{if } f(x) \text{ is not equal to WHITE} \\ f''(x) &= f'(x) && \text{if } f(x) \text{ is equal to WHITE} \end{aligned}$$

Figure 2-11 shows the overlay of two binary pictures.

2.2.2. DEFINITION OF REGION PYRAMIDS AND QUADTREES

Having explored the basic operations on pictures, it is now appropriate to consider the definition of the pyramid and the quadtree. Then we will look at two types of each. Given a picture $\langle F^i, C, f \rangle$, denoted P , we shall define a pyramid as an ordered quadruple $\langle P, C', h, g \rangle$, where C is a subset of C' , and g is total from the subframes of F^i into C' and is hierarchical in f and h . By g being hierarchical in f and h , it is meant that

$$\begin{aligned} g(\{x\}) &= f(x) && \text{if } x \in F^i \\ g(x) &= h(g(x^{NW}), g(x^{NE}), g(x^{SE}), g(x^{SW})) && \text{otherwise} \end{aligned}$$

Recall that if x is in F^i , then the singleton set $\{x\}$ is the same as $\text{subframe}(F^i, x)$. All pyramids that have the same values of C , C' , and h are viewed as pyramids of the same type.

It is sometimes useful to view the pyramid $\langle P, C', h, g \rangle$ as set of ordered pairs

$$\{ \langle x, g(x) \rangle \mid x \text{ is a subframe of } F^i \}$$

but it is important not to lose track of the values of C , C' , and h . When the pyramid $\langle P, C', h, g \rangle$ is viewed as a set of ordered pairs, there is a special subset of it called a quadtree. The quadtree of a pyramid $\langle P, C', h, g \rangle$ is defined as a subset of that pyramid where

$$\{ \langle x, g(x) \rangle \mid g(x) \in C \text{ and } g(x') \notin C \}$$

The most basic type of pyramid is one where: GRAY is not an element of C ; C' is the union of C and $\{\text{GRAY}\}$; and

$$\begin{aligned} h(a, b, c, d) &= a && \text{if } a = b = c = d \\ h(a, b, c, d) &= \text{GRAY} && \text{otherwise} \end{aligned}$$

A pyramid that satisfies these conditions shall be called a region pyramid. Such a pyramid is illustrated in Figure 2-12. Note that since the subframes appeared on our

basic picture representation. Figures 2-6 and 2-12 appear identical. The quadtree of a region pyramid is called a region quadtree. Such a quadtree is illustrated in Figure 2-13. A region quadtree is said to represent a region pyramid in the sense that given the region quadtree, we can reconstruct the region pyramid. This will be proved in the Region Quadtree Representation Theorem.

In order to prove the following theorem, we need the concept of a partial region quadtree. Let us define the partial region quadtree Q^i of $\langle\langle F^i, C, f \rangle, C', h, g \rangle$ as follows :

$$\begin{aligned} Q^0 &= \{ \langle \{x\}, f(x) \rangle \mid x \in F^i \} \\ Q^{j+1} &= \{ \langle x, g(x) \rangle \mid g(x) \in C \text{ and either} \\ &\quad x = y^f \text{ where } \langle y, g(y) \rangle \in Q^j \text{ or} \\ &\quad \langle x, g(x) \rangle \in Q^j \text{ but } g(x^f) = \text{GRAY} \} \end{aligned}$$

Figure 2-14 presents the partial quadtree Q^1 for the pyramid shown in Figure 2-12. Since if any of the arguments to h are GRAY, then h returns GRAY, it follows that Q^i is the region quadtree of $\langle\langle F^i, C, f \rangle, C', h, g \rangle$.

Theorem 10: (The Region Quadtree Representation Theorem): Given a region quadtree Q constructed from some region pyramid Y from a picture with frame F^1 colored with colors C , it is possible to reconstruct the region pyramid Y .

Proof: In order to reconstruct the region pyramid, we must determine the values of $\langle\langle F^1, C, f \rangle, C', h, g \rangle$. The value of F^1 and C are already given. Since Y is a region pyramid, we know that C' is the union of C with $\{\text{GRAY}\}$. Similarly, since Y is a region pyramid, the function h is known. Thus, if we can reconstruct the function f , we will be able to derive the function g and the reconstruction will be finished.

In order to reconstruct f , it is sufficient to show that we can reconstruct the partial region quadtree Q^0 from region quadtree Q , i.e., the partial quadtree Q^1 . Given the partial quadtree Q^{j+1} , we can reconstruct the partial quadtree Q^j by replacing each ordered pair $\langle x, y \rangle$ in Q^{j+1} where the length of the root of x is $i-j$ with $\langle x^{\text{NW}}, y \rangle$, $\langle x^{\text{NE}}, y \rangle$, $\langle x^{\text{SE}}, y \rangle$, and $\langle x^{\text{SW}}, y \rangle$. This works because it is the exact inverse of the process that constructed Q^{j+1} from Q^j . By using this process j times, we can construct Q^0 from Q . Since we can construct f from Q^0 (because $f(x) = y$ iff $\langle \{x\}, y \rangle$ in Q^0), we have completed the reconstruction of the region pyramid from the region quadtree.

2.2.3. DEFINITIONS RELATING TO AND INCLUDING LINE QUADTREES

Until now, we have been viewing colors as indivisible objects. It is sometimes useful to view colors as having a structure. This is the case when defining the difference picture $D(P)$ of a picture P .

The difference picture $D(\langle F^i, C, f \rangle)$ is defined as $\langle F^i, C^*, f^* \rangle$ where C^* is the set of ordered quadruples of $\{SET, CLEAR\}$ and f^* is defined as

$$f^*(x) = \langle \text{diff}(x,n), \text{diff}(x,e), \text{diff}(x,s), \text{diff}(x,w) \rangle$$

where diff is defined as

$$\begin{aligned} \text{diff}(x,r) &= CLEAR && \text{if for some } y, x \text{ } r \text{ } y \text{ and } f(x) = f(y) \\ \text{diff}(x,r) &= SET && \text{otherwise} \end{aligned}$$

We consider $\langle CLEAR, CLEAR, CLEAR, CLEAR \rangle$ to be the distinguished color WHITE.

Furthermore, let us define the notational convention

$$c = \langle c[n], c[e], c[s], c[w] \rangle$$

as a way of indicating the four parts of the color c . Figure 2-15 shows how we will represent difference pictures in our figures.

Subframes in a region pyramid that cannot be mapped onto one of the colors of the corresponding picture are mapped onto GRAY. In the case of a line pyramid, we will map subframes that do not have a reasonable interpretation as an ordered quadruple of $\{SET, CLEAR\}$ onto the color CROSSED. The color CROSSED corresponds to the notion that the subframe represents a region that is split by some boundary. Thus, for any region pyramid, there exists a corresponding line pyramid such that all the subframes mapped to GRAY in the region pyramid are mapped to CROSSED in the line pyramid.

A line pyramid $\langle \langle F^i, C^*, f^* \rangle, C^{*'}, h^*, g^* \rangle$ is one where: CROSSED is not an element of C^* ; $C^{*'}$ is the union of C^* and $\{CROSSED\}$; and

$$\begin{aligned} h^*(c_{NW}, c_{NE}, c_{SE}, c_{SW}) &= \text{merge}(c_{NW}, c_{NE}, c_{SE}, c_{SW}) \\ &\quad \text{if uncrossed}(c_{NW}, c_{NE}, c_{SE}, c_{SW}) \\ h^*(c_{NW}, c_{NE}, c_{SE}, c_{SW}) &= CROSSED \quad \text{otherwise} \end{aligned}$$

assuming the following definitions of uncrossed and merge . The predicate uncrossed is used to verify that all of the borders between siblings have the value CLEAR; thus,

$\text{uncrossed}(c_{NW}, c_{NE}, c_{SE}, c_{SW})$ is true iff $c_{NW}[s]$, $c_{NW}[e]$, $c_{NE}[s]$, $c_{NE}[w]$, $c_{SE}[n]$, $c_{SE}[w]$, $c_{SW}[n]$, and $c_{SW}[e]$ are all CLEAR. If any of c_{NW} , c_{NE} , c_{SE} , or c_{SW} are CROSSED, then uncrossed is false. The function merge is used to produce the quadruple x , where for each side relation r , $x[r]$ is SET if for the two siblings, x_1 and x_2 , that do not have brothers as neighbors on side r , both $x_1[r]$ and $x_2[r]$ have the value SET. If we let \wedge denote the infix operator that maps SET \wedge SET into SET and all other argument pairs into CLEAR, then we can define merge as follows:

$$\begin{aligned} \text{merge}(c_{NW}, c_{NE}, c_{SE}, c_{SW}) = & \langle c_{NW}[n] \wedge c_{NE}[n], \\ & c_{NE}[e] \wedge c_{SE}[e], \\ & c_{SE}[s] \wedge c_{SW}[s], \\ & c_{SW}[w] \wedge c_{NW}[w] \rangle \end{aligned}$$

Figure 2-16 shows a pictorial representation for a line pyramid.

The quadtree of a line pyramid is called a line quadtree. A line quadtree is said to represent a line pyramid in the sense that the line pyramid can be reconstructed from the line quadtree. Figure 2-17 shows a pictorial representation of the line quadtree.

As with the region quadtree, we would like to show that the line quadtree is sufficient to allow reconstruction of the line pyramid. Figure 2-18 illustrates the partial quadtree Q^i for the quadtree of Figure 2-17. In order to do this, we need the concept of a partial line quadtree. The definition is the same as for the partial region quadtree except that references to GRAY are changed to CROSSED. Since if any of the arguments to h^* are CROSSED, then h^* returns CROSSED, it follows that the partial line quadtree Q^{*i} is the quadtree of $\langle \langle F^i, C^*, f^* \rangle, C^{*'}, h^*, g^* \rangle$.

Unfortunately, our usage of the partial line quadtree to prove the Line Quadtree Representation Theorem is more complicated than our usage of the partial region quadtree to prove the Region Quadtree Representation Theorem. To solve this complication, we need the following three lemmas.

The first lemma states that two neighboring subframes in the domain of the partial quadtree will agree in the value of the bordering sides. The key to the proof of this lemma is that two subframes can only be neighbors if they are of the same depth.

Lemma 11: If $\langle x, c_x \rangle$ and $\langle y, c_y \rangle$ are in the partial line quadtree Q^{*i} , then if there exists a side relation r such that

$$x \ r \ y$$

then

$$c_x[r^{-1}] = c_y[r]$$

Proof: Note that the lemma holds when j equals 0. It follows from the definition of the partial line quadtree that if the lemma holds for Q^{*i} , then it holds for Q^{*i+1} . Thus, the lemma is true.

The next lemma states that the domain of the partial region quadtree is the same as the domain of the partial line quadtree. This allows us to pass back and forth between interpreting the process of forming a partial quadtree as forming a line quadtree or as forming a region quadtree. In later sections, this will mean that most results will apply to both kinds of quadtree. An instance of the truth of this lemma can be seen by noticing that Figure 2-18 and Figure 2-14 have the same number of subframes.

Lemma 12: If $\langle x, c \rangle$ is in the partial region quadtree Q^i , then there exists a c^* such that $\langle x, c^* \rangle$ is in the partial line quadtree Q^{*i} .

Proof: Again, note that the lemma holds for Q^0 and Q^{*0} . Assume i is the smallest number for which the lemma fails. The lemma could only fail because in one of the partial quadtrees four brothers were replaced by their father whereas in the other they were not. Both the assumption that the merger failed to occur in the partial line quadtree and the assumption that the merger failed to occur in the partial region quadtree lead to contradictions. Therefore, the lemma holds.

The last lemma states that, for partial line quadtrees, if on some side the father is CLEAR but one of the sons was SET, then the father does not have a neighbor on that side. This means that if the information that a given side is SET is lost by a merger of one of the two subframes that contains it, then the other will not be merged. Thus, in actuality, the information is never lost.

Lemma 13: If there exists a side relation r such that $x \ r \ y$ where $\langle x, c \rangle$ and $\langle y, d \rangle$ are in the partial line quadtree Q^{*i} and $c[r^{-1}]$ and $d[r]$ are SET but $\langle x', c' \rangle$ is in the partial line quadtree Q^{*i+1} and $c'[r]$ is CLEAR, then there is no d' such that $\langle y', d' \rangle$ is in the partial line quadtree Q^{*i+1} .

Proof: The assumption that the lemma is false, together with Lemmas 11 and 12, leads to a contradiction. Thus, the lemma holds.

We now have sufficient knowledge of partial line quadrees to prove the Line Quadtree Representation Theorem. This theorem simply states that given a line quadtree, we can reconstruct the line pyramid from which it came, and hence the difference picture from which the line pyramid was built.

Theorem 14: (The Line Quadtree Representation Theorem): Given a line quadtree Q^* constructed from some line pyramid Y from a picture with frame F^i colored with colors C^* , it is possible to reconstruct the line pyramid Y .

Proof: In order to reconstruct the line pyramid, we must determine the values of $\langle \langle F^i, C^*, f \rangle, C^{*'}, h^*, g^* \rangle$. As in the proof of the Region Quadtree Representation Theorem, the crux is to show that we can reconstruct the partial quadtree Q^{*0} from quadtree Q^* , i.e., the partial quadtree Q^{*1} . Given the partial quadtree Q^{*1} , we can reconstruct the partial quadtree Q^{*j} by replacing each ordered pair $\langle x, c \rangle$ in Q^{*j+1} where the length of the root of x is $i-j$ with

$$\begin{aligned} &\langle x^{NW}, \langle c[n], \text{CLEAR}, \text{CLEAR}, c[w] \rangle \rangle \\ &\langle x^{NE}, \langle c[n], c[e], \text{CLEAR}, \text{CLEAR} \rangle \rangle \\ &\langle x^{SE}, \langle \text{CLEAR}, c[e], c[s], \text{CLEAR} \rangle \rangle \\ &\langle x^{SW}, \langle \text{CLEAR}, \text{CLEAR}, c[s], c[w] \rangle \rangle \end{aligned}$$

Unfortunately, this does not directly recover Q^{*1} because the use of the \wedge operator in the function merge used by h^* , means that we cannot always directly recover the color of the son from its father. For example, if $x[n]$ is SET, then $s^{NW}[n]$ must also be SET, but if $x[n]$ is CLEAR, all we know is that $s^{NW}[n]$ and $s^{NE}[n]$ cannot both be set. However, as a consequence of Lemma 13, if we perform the above procedure starting with changing Q^{*1} into Q'^{*1-1} , Q'^{*1-1} into Q'^{*1-2} , etc., we eventually produce a Q'^{*0} . This Q'^{*0} can be changed into Q^{*0} by collecting all pairs of neighbors that violate Lemma 11 and changing their colors so that the appropriate side is SET for both members of the pair.

Now, from Q^{*0} we can reconstruct f^* , thus, proving the theorem.

Examples of the usage of region quadrees can be found in [3, 4, 12, 13, 17, 25, 26]. The basic algorithms for line quadrees can be derived from the corresponding region quadtree algorithm by using techniques discussed in [27].

2.3. RESULTS ON THE SIZE OF QUADTREES

In the previous sections, we have presented the picture, pyramid, and quadtree as abstract mathematical objects. However, we are ultimately interested in them because of our ability to represent and manipulate them within a computer. In this thesis, we shall use an unbounded-size-random-access-memory model of a computer. The main result of this choice of model, is that we will not consider implementation problems that arise from paging or the usage of disk and/or tape files. This random-access memory is organized (by software) into objects called records. A record consists of a fixed number of fields, some of which are viewed as pointers to other records.

A computer representation of a picture $\langle F^i, C, f \rangle$ that is consistent with the usage requirements of standard picture algorithms can be constructed in the following manner. Let each record R_x correspond to an element x of the frame F^i . Let the value field, denoted $\text{value}[R_x]$, of a record encode the color $f(x)$. The remainder of the record consists of four pointer fields, denoted $\text{kin}[R_x, r]$ (where r is a side relation). For a give side relation r , $\text{kin}[R_x, r]$ is R_y where $y \ r \ x$, and is considered NULL, if x does not have a neighbor on the r side. Figure 2-19 shows the record structure for the difference picture of Figure 2-15. Having drawn the distinction between a pixel of a picture and the record of the corresponding computer representation, we shall now proceed to ignore this distinction, for the sake of readability. Note that sometimes it is convenient to view each record as a node in a graph where the pointer fields indicate arcs.

The following theorem indicates how large a random access memory would have to be in order to contain this representation of a picture. In performing this and other calculations, we consider each field of a record to have the same size. Also, we are not interested in the particular constants involved in the calculations; so most results will be presented in terms of their order of magnitude.

Theorem 15: (The Picture Storage Requirements Theorem): The number of records required to represent a picture $\langle F^i, C, f \rangle$ is proportional to 4^i , i.e., $O(4^i)$.

Proof: The above picture representation has one record for each element of F^i ; therefore, the number of records required is equal to the cardinality of F^i , i.e., $|F^i|$. Referring to the definition of frame, the following holds:

$$|F^0| = 1$$

$$|F^j| = 4 \cdot |F^{j-1}|$$

Hence, the cardinality of F^i (and the number of records in the representation) is 4^i .

The computer representation of a pyramid $\langle\langle F^i, C, f \rangle, C', h, g \rangle$ is built from records that represent the view of a pyramid as a set of ordered pairs $\langle x, g(x) \rangle$ where x is a subframe of F^i . The pyramid record RP_x that corresponds to the subframe x has six fields. The value field, $\text{value}[RP_x]$ encodes $g(x)$. The remaining five fields are pointer fields, denoted:

$$\begin{aligned} \text{father}[RP_x] &= RP_{(x^f)} \\ \text{son}[RP_x, NW] &= RP_{(x^{NW})} \\ \text{son}[RP_x, NE] &= RP_{(x^{NE})} \\ \text{son}[RP_x, SE] &= RP_{(x^{SE})} \\ \text{son}[RP_x, SW] &= RP_{(x^{SW})} \end{aligned}$$

For all pointer fields, if the field refers to an object that does not exist, then the field is said to be NULL. The record structure of the line pyramid shown in Figure 2-16 is shown in Figure 2-20. Having drawn the distinction between a subframe-color pair and a record of the corresponding computer representation, we shall proceed to ignore the distinction.

When viewed as a graph, the nodes of a pyramid form a tree. If we consider the root of the graph to be the node corresponding the record that corresponds to the subframe whose root is λ , then by induction on the frame depth of the represented quadtree (and using the Subframe Invariance Theorem), it can be shown that the graph is connected and acyclic. Thus, viewing the pyramid as a graph introduces tree terminology, e.g., depth, nodes, and leaves.

The following theorem indicates the storage requirements of our pyramid representation. Note that although the pyramid almost always requires more space than the picture, their requirements are of the same magnitude.

Theorem 16: (Pyramid Storage Requirements Theorem): The number of records required to represent a pyramid $\langle\langle F^i, C, f \rangle, C', h, g \rangle$ is of the magnitude $O(4^i)$.

Proof: The number of records used to represent a pyramid is the same as the number of subframes of F^i . From the definition of subframes, we note that the number of subframes of F^i is equal to

$$|F^0| + |F^1| + \dots + |F^i|$$

which is in turn equal to

$$4^0 + 4^1 + \dots + 4^i$$

according to our calculations from the proof of the Picture Storage Requirements Theorem. The above reduces to

$$(4/3) \cdot (4^i - 1) + 1$$

which is of magnitude $O(4^i)$.

Now, let us construct a computer representation for a quadtree. We let the quadtree have the same record fields as the pyramid. However, just as the quadtree is a subset of the pyramid, so the records used to represent the quadtree will be a subset of the records used to represent a pyramid. Let the term full quadtree denote the smallest set that contains the quadtree and is closed under the operation of finding a subframe's father. With respect to a full quadtree, the quadtree from which it was built is referred to as the fringe. Henceforth, we will not draw a distinction between a quadtree and a full quadtree. The set of records that form the quadtree representation is that subset of the pyramid-representation records that correspond to subframes in the full quadtree with the modification that the son fields of records corresponding to the fringe have been set to NULL. An example of the record structure of a quadtree that corresponds to the line quadtree of Figure 2-17 is shown in Figure 2-21.

When viewed as a graph the quadtree representation forms a subgraph of the pyramid representation. The records of our quadtree representation also form a tree. The argument showing that they form a tree follows the same form as the one presented for the analogous statement with respect to pyramids. The same terminological ramifications apply, i.e., nodes of a quadtree have depth and are sometimes leaves. As well as the notion of the depth of a node, there is also the notion of the depth of the entire tree. Note that the depth of a quadtree is bounded from above by the maximum depth of a subframe that corresponds to a node of the tree.

The above Pyramid Storage Requirements Theorem presents the space requirements in terms of a function of the pyramid depth i . If we were to present the space requirements of a quadtree as a function of the quadtree depth, we would get a bound of the same order of magnitude as that of pyramids. This is an immediate consequence of considering the quadtrees for a sequence of checker-board pattern pictures, as illustrated in Figure 2-22.

It turns out that it is useful to consider the size of the representation of a quadtree, henceforth called the size of the quadtree, as a function of the length of the perimeter of the picture that the quadtree represents. Recall that the length of the perimeter of a picture P has been defined as the size of the set S where

$$S = \{ \{x, y\} \mid \begin{array}{l} x \text{ and } y \text{ are in } \text{surround}(P) \text{ and} \\ x \text{ } r \text{ } y \text{ for some side relation } r \text{ and} \\ \text{the color of } x \text{ is not the color of } y \end{array} \}$$

It can be shown that the length of the perimeter of P is equal to the length of the perimeter of $\text{surround}(P)$ and that the size of the quadtree of $\text{surround}(P)$ is eight plus the size of the quadtree of P . Before we can analyze the size of the quadtree in terms of its perimeter, we will need to consider some auxiliary concepts.

We say that P contains a path, if the above set S forms a path. A set S is said to form a path if and only if there is an enumeration of the elements of S , s_1 through s_j , where j equals $|S|$, such that for any pair of consecutive elements s_i and s_{i+1} , one of the following two conditions is satisfied. If s_i denotes $\{x, y\}$ where $x \text{ } n \text{ } y$ (the first case), then one of the following three conditions holds:

1. s_{i+1} has exactly one element, z , in common with s_i and the other element is either an eastern or a western neighbor of z .
2. Each element of s_{i+1} is an eastern neighbor of an element of s_i .
3. Each element of s_{i+1} is a western neighbor of an element of s_i .

Similarly, if s_i denotes $\{x, y\}$ where $x \text{ } e \text{ } y$ (the second case), then one of the following three conditions holds:

1. s_{i+1} has exactly one element, z , in common with s_i and the other element is either a northern or a southern neighbor of z .
2. Each element of s_{i+1} is a northern neighbor of an element of s_i .
3. Each element of s_{i+1} is a southern neighbor of an element of s_i .

Note that for i less than j , all possible values of s_i have been covered. Recalling the picture P of Figure 2-4 and $\text{surround}(P)$ of Figure 2-6, the path corresponding to the perimeter of $\text{surround}(P)$ is shown in Figure 2-23. A number i inside the square that represents an element of the frame of $\text{surround}(P)$ indicates that that element was in s_i . Note that in order to simplify the presentation, we do not indicate the color associated with the elements of $\text{surround}(P)$.

Given the notion of a path, defined above, it is useful to construct the notion of an augmented path and then the notion of the reduction of an augmented path. The notion of an augmented path is needed because there is not a simple notion of the reduction of a path that is useful in the analysis of the space requirements of quadrees. A set S that can be defined as having elements of the form $\{x,y\}$ where all the x 's and y 's are elements of the same frame is said to form an augmented path if and only if the following condition holds. There is a sequence of its elements (allowing duplication but not omission) where for each i less than j , one of the following holds:

1. s_i and s_{i+1} are both of cardinality 2 and they obey the conditions for a path listed above.
2. Exactly one of s_i and s_{i+1} is a set of cardinality 1 and it is a subset of the other element.
3. Both s_i and s_{i+1} are sets of cardinality 1, $\{x\}$ and $\{y\}$ respectively, and there exists a side relation r , such that $x \text{ } r \text{ } y$.

Note that it is permitted for $\{x,y\}$ to be an element of S even though x equals y . Since every path is also an augmented path, Figure 2-23 would be a trivial example of an augmented path. A reduction of an augmented path is formed by replacing every element of an element of an augmented path with its father. Figure 2-24 indicates the augmented path that happens to be the reduction of the augmented path shown in Figure 2-23. Note that depth of the frame of the reduction of the augmented path is one less than the depth of the frame of the original augmented path. That the reduction of an augmented path is always an augmented path is the content of the following lemma.

Lemma 17: (The Augmented Path Reduction Lemma): If the set S

forms an augmented path with respect to some frame F^{i-1} , then the reduction of S , denoted S' , forms an augmented path with respect to the frame F^i .

Proof: The new sequence for S' , is the result of applying the reduction to the sequence for S , and then eliminating successive duplicate entries. The proof of this lemma results from consideration of the result of performing a reduction upon two successive elements of S , s_i and s_{i+1} , where these successive elements satisfy either one of the conditions for a path, or they satisfy one of the last two conditions for an augmented path. If the successive elements satisfy one of the conditions for a path, then, one of the following holds:

1. Both s'_i and s'_{i+1} have cardinality 2, in which case either they are equal (and hence only one of them is left in the sequence) or they are distinct. If they are distinct, then it can be shown that they satisfy the conditions for a path.
2. Exactly one of s'_i and s'_{i+1} has cardinality one, in which case it must satisfy the second condition for an augmented path.
3. Both s'_i and s'_{i+1} have cardinality one, in which case either they are equal (and hence only one of them is left in the sequence) or they are distinct. If they are distinct, then they must satisfy condition 3) for an augmented path.

If s_i and s_{i+1} are in the sequence because they satisfy one of the last two conditions for an augmented path, then the reduction transformation will either leave them identical (and thus, one is removed from the sequence) or s'_i and s'_{i+1} will satisfy one of the last two conditions for an augmented path.

Each of the statements above can be verified by a proof by contradiction.

From our point of view, the most interesting result from reducing an augmented path is that the size of the reduced path is 4/5ths the size of the original path.

Lemma 18: (The Reduced Augmented Path Size Lemma): If j is the length of a sequence satisfying the conditions of an augmented path for a set S , then there exists a sequence satisfying the conditions of an augmented path for the reduced set S' and the length of that sequence is bounded from above by

$$4 + 4 \cdot (j / 5)$$

Proof: This lemma can be demonstrated by considering every possible form that an augmented path of length five can take and noting that in every case the result of reducing the path yields a sequence of length four or less. Having broken the path into a sequence of subpaths of length 5, it is possible for there to be a subpath of length 4 left over that will reduce to being still of length 4.

In the above lemma, we analyzed the effect of reduction on subpaths of length 5. We chose the length 5 because shorter subpaths need not always reduce. If we had chosen a longer subpath length, e.g., subpaths of length 125, we would expect to get a better reduction rate (and hence better constants in the Quadtree Storage Results Theorem). However, it should be noted that the reduction rate for subpaths of length 4 can be easily verified by hand calculation, whereas the reduction rate for subpaths of length 125 would require either further theoretical analysis or extensive calculations feasible only for a computer. Indeed, it is not clear how a computer could be programmed to perform this calculation in a manner that does not suffer from exponential growth.

We can now demonstrate the relation between the length of the perimeter of P and the size of the quadtree for P .

Theorem 19: (The Quadtree Storage Requirements Theorem): If Q is the quadtree of a picture P that has a path, then the number of records required to represent the quadtree Q is of the magnitude $O(p+d)$, where d is the depth of P and p is the length of the perimeter of P .

Proof: First, we note that for any quadtree node x , either x or one of x 's siblings is on the perimeter or has a descendant that is on the perimeter. Next, we note that if a node is on the perimeter or has a descendant that is on the perimeter, then it is a member of an element of the path or it is a member of an element of one of the augmented paths that result from performing a reduction. From an application of the Reduced Augmented Path Size Lemma, we deduce the following upper bound on the number of nodes in a quadtree of depth d representing an object of perimeter p that are on the path or have descendants that are on the path.

$$\sum_{i=0}^d 4 + (4/5)^i \cdot p$$

This upper bound is of order $O(p+d)$. Since no element of a path can have more than two elements itself, and since each node has three siblings, there must be an upper bound of order $O(p+d)$ on the number of nodes needed to represent a picture P that has a path.

The result of the above theorem can be extended to quadtrees by use of the following theorem [12, 13].

Theorem 20: (The Quadtree Overlay Storage Theorem): Let Q denote a quadtree that represents a picture $\langle F^i, C, f \rangle$ denoted P . Let Q' denote a quadtree that represents a picture $\langle F^i, C', f' \rangle$ denoted P' . Let Q'' denote a quadtree that represents a picture $\langle F^i, C'', f'' \rangle$ denoted P'' , where P'' is

the result of overlaying P on top of P' . The the number of records required to store the quadtree Q'' is bounded from above by the sum of the number of records needed to store the quadtrees Q and Q' .

Proof: Consider the partial quadtrees that correspond to Q , Q' , and Q'' . It can be shown by induction that for each j less than or equal to i , the size of Q''^j is bounded from above by the sum of the sizes of Q^j and Q'^j .

Thus, a picture P could be viewed as having been built from the overlay of j pictures, P_1 through P_j , where for all i from 1 to j , P_i has a path. Then the size of the quadtree that represents P would be bounded from above by the sum of the sizes of the quadtrees that represent P_i , where i is from 1 to j . Unfortunately, this does not always give a good bound on the number of nodes in a quadtree. Consider the sequence of quadtrees for modified checker-board patterns as illustrated in Figure 2-25. The modification referred to is that the BLACK pixels have been separated so that there is no path that connects the perimeter of any two BLACK pixels. Here, we would have to consider the picture as the union of $O(4^j)$ 2^j by 2^j pictures, for each of which, the length of the perimeter was 4. The Quadtree Overlay Storage Theorem would allow us to deduce the following upper bound:

$$O(\sum_{i=1}^j 4^i)$$

This upper bound is of magnitude $O(j \cdot 4^j)$, whereas the number of nodes in such a quadtree is of magnitude $O(4^j)$. Note that we have modified the checker-board pattern so that there is no other way of breaking the picture up into fewer pictures, all of which have paths.

Other approaches to the question of the storage requirements of a quadtree are possible. Given a particular geometric model for the neighbor axioms like that of Section 4.1, there are two known results on the size of a quadtree. Hunter's theorem [12, 13] states that the size of a quadtree that represents a polygon is proportional to the sum of the depth q of the quadtree plus the length p of the perimeter of the polygon. This result is analogous to the Quadtree Storage Requirements Theorem, although its proof is different. In particular, the exact upper bound that Hunter derives is $16q-11+16p$, which has constants that are considerably lower than those of the Quadtree Storage Requirements Theorem. However, the proof of the Quadtree Storage

Requirements Theorem could be modified to yield better constants at the expense of additional complications.

The other known result on quadtree size is due to Dyer [5]. He analyzes the average number of nodes in a quadtree that represents a 2^m by 2^m square embedded within a 2^q by 2^q picture. Assuming that each element of the frame of the picture is equally likely to contain the upper left hand corner of the square, Dyer shows that both the average and worst case quadtree sizes are $O(2^m + q - m)$. Although this result is subsumed by Hunter's Theorem, the constants involved are considerably lower; in particular, Dyer calculates an upper bound on the worst case for square representation to be (in Hunter's notation):

$$4 \cdot p + 16 \cdot (q + 2 - \log_2(p)) - 27$$

Figure 2-26 illustrates the wide variance (from 5 nodes to 53 nodes) in the number of nodes needed to represent a 4 by 4 square within an 8 by 8 picture. It should be noted that these quadtrees represent pictures that are shifts of each other. Considering the wide variance illustrated by Figure 2-26, it is interesting to consider the problem of developing an algorithm whose input would be a picture and whose output would be a shift of that picture that minimizes the number of quadtree nodes needed. The transformation resulting from this algorithm (the NM transformation) is considered in depth in Section 3.2.3 If P is a 2^q by 2^q picture, then Li, Grosky, and Jain [20] propose an algorithm that executes in time proportional to

$$2^{2 \cdot q} \cdot q$$

This algorithm uses dynamic programming. It is based on the following two results on the size of the quadtree representing the shift of a picture.

1. Given a picture $\langle F^i, C, f \rangle$, the minimum of the depth of the smallest quadtree corresponding to a shift of the picture is bounded from above by $i+1$.
2. Given that P' is a shift of P in some direction with distance 2^i and that q is the depth of the pyramid corresponding to P , then the number of nodes with depth d in the quadtree for P' is equal to the number of nodes with depth d in the quadtree for P , if d is greater than $q-i$.

A proof of the above two results within the Quadtree Theory developed above is beyond the scope of this dissertation. However, the class of quadtrees generated by this algorithm is considered in Section 3.2.3.

2.4. ON FINDING NEIGHBORING QUADTREE NODES

In the computer representation of pictures outlined in the previous section, finding the neighbor of a given picture element is simply a matter of dereferencing a pointer field. To determine the corresponding information about a node in a pyramid or a quadtree is more complicated. The neighbor-finding algorithm discussed below has played a key role in various applications, including: chain code to quadtree conversion [25], quadtree to chain code conversion [4], and connected component labeling [26]. A general discussion of the problem of locating side and diagonal neighbors in a quadtree, with particular emphasis on average case analysis, is presented in [28]. In the following, we are only interested in finding side neighbors. Furthermore, we are only concerned with the worst-case analysis of this algorithm.

The algorithm for finding a side neighbor in a pyramid is simply a procedural encoding of the information presented in Table 2-4. One encoding of this algorithm is presented in Table 2-5. It is assumed that applying any operation to ERROR yields ERROR as the result (with the exception of the query ISERROR, which always returns TRUE or FALSE). The correctness of the algorithm in Table 2-5 is demonstrated in the proof of the following lemma.

Lemma 21: (Neighbor Finding Correctness Theorem): Given the procedural definition of Table 2-5, for any subframes x and y , for any side relation r

$$y = \text{NEIGHBOR}(x, r) \text{ iff } x \text{ } r \text{ } y$$

Proof: As with most proofs of iff-type theorems, we will first prove the only-if-part and then prove the if-part. The only-if-part states that

$$y = \text{NEIGHBOR}(x, r) \text{ implies } x \text{ } r \text{ } y$$

First, by induction on the actual depth of x , we show that the actual depth of $\text{NEIGHBOR}(x, r)$ is equal to the actual depth of x . Then by induction on the actual depth of x and y , using the results in Table 2-4, it is clear that

$$y = \text{NEIGHBOR}(x, r) \text{ implies } x \text{ } r \text{ } y$$

The if-part states that

$$x \text{ } r \text{ } y \text{ implies } y = \text{NEIGHBOR}(x, r)$$

By the Neighbor Uniqueness Theorem and the only-if-part above, if $x \text{ } r \text{ } y$ and $\text{NEIGHBOR}(x, r)$ is not equal to ERROR, then

$$y = \text{NEIGHBOR}(x, r)$$

If $\text{NEIGHBOR}(x, r)$ equals ERROR, then we can use the Embedding Lemma (Lemma 6) to construct an x' that denotes the embedding of x in the appropriate quadrant q . Through careful choice of q , we obtain that

$\text{NEIGHBOR}(x',r)$ equals some subframe z that is not in quadrant q . However the embedding of y , denoted y' , is in quadrant q and $x' r y'$. From the only-if part of this lemma, we can deduce a contradiction with respect to the Neighbor Uniqueness Theorem. Therefore

$$x r y \text{ implies } y = \text{NEIGHBOR}(x,r)$$

Thus, the procedure is correct.

However, recalling the Neighbor Decidability Theorem (Theorem 3), we realize that we already have a correct algorithm for calculating neighbors. The difference is that the algorithm of Table 2-5 is practical from an execution time point of view. In particular, we have the following theorem.

Theorem 22: (Order Depth Neighbor Finding Theorem): The execution time of the algorithm for $\text{NEIGHBOR}(x,r)$ is proportional to the actual depth of x .

Proof: Inspection of the algorithm shows that its execution time is proportional to the number of times the function neighbor is invoked. Let $\text{Cost}(x,r)$ denote the number of times the algorithm invokes the neighbor function in order to calculate $\text{NEIGHBOR}(x,r)$. Thus, for any subframe x and side r , either

$$\text{Cost}(x,r) = 1$$

or

$$\text{Cost}(x,r) = 1 + \text{Cost}(x',r)$$

Using these relations, it can be shown by induction on the actual depth of x , that $\text{Cost}(x,r)$ is bounded from above by the actual depth of x . Therefore, the execution time for this algorithm is bounded from above by a function proportional to the actual depth of x .

The above theorem could be extended to cover diagonal neighbors by using the No-Corner-Cutting Theorem. For a fuller treatment of diagonal neighbors see [28].

In extending the above theorem to work with quadrees, we encounter the problem of what to do if the subframe $\text{NEIGHBOR}(x,r)$ is not in the quadtree Q . Let y denote the subframe $\text{NEIGHBOR}(x,r)$. First, we note that y will be in the partial quadtree Q^i where i is the difference between the maximum and the actual depths of x . Next, we observe that if y is not in Q , then it is not in the partial quadtree Q^{i+1} . Finally, we recall that if y is in Q^i , but not in Q^{i+1} , then y' is in Q^{i+1} and the color of y' is the same as the color of y . Now, $\text{NEIGHBOR}(x',r)$ must equal y' , since the only other alternative was that $x' = y'$ and this is impossible as x' has four sons (since x was in

Q) and y^f has no sons in Q^{i+1} (since y was not in Q). Let us modify the algorithm for the function $\text{NEIGHBOR}(x,r)$ given in Table 2-5 so that it calculates the function $\text{QUADNEIGHBOR}(x,r)$ defined as

$$\text{QUADNEIGHBOR}(x,r) = \begin{array}{ll} \text{NEIGHBOR}(x,r) & \text{if } \text{NEIGHBOR}(x,r) \text{ is in } Q \\ \text{QUADNEIGHBOR}(x^f,r) & \text{otherwise} \end{array}$$

Then, based on an induction over the observations mentioned above, $\text{QUADNEIGHBOR}(x,r)$ is in Q iff $\text{NEIGHBOR}(x,r)$ is not ERROR. Also, the color of $\text{QUADNEIGHBOR}(x,r)$ is equal to the color of $\text{NEIGHBOR}(x,r)$. Further justification of the function QUADNEIGHBOR is dependent on the model of Neighbor Theory presented in Section 4.1. It is an easy corollary of the Order Depth Neighbor Finding Theorem, to show that there is an algorithm for QUADNEIGHBOR that executes in time proportional to the actual depth of x .

Thus, the Order Depth Neighbor Finding Theorem covers the case of finding a neighbor in a quadtree or a pyramid. However, this does not complete our analysis of neighbor finding. Many algorithms, e.g., quadtree to chain code conversion, perform QUADNEIGHBOR 's over at most some constant portion k of the quadtree. Thus, if $\text{QuadCost}(Q,r)$ is the cost of executing $\text{QUADNEIGHBOR}(x,r)$ for each leaf x in the quadtree Q divided by the number of leafs in Q , then we can calculate an upper bound on the average cost of performing each of the QUADNEIGHBOR s as $\text{QuadCost}(Q,r)/k$. Note that this simply means that the cost of executing a fixed number of QUADNEIGHBOR s is bounded from above by the cost of executing all possible QUADNEIGHBOR s.

Before analyzing QuadCost , let us consider the analogous function PyrCost , i.e., the cost of executing $\text{NEIGHBOR}(x,r)$ for each subframe whose actual depth equals his maximum depth divided by the number of leaves in the pyramid. Note that this includes executions of NEIGHBOR on subframes that do not have a neighbor on the given side r . Thus, before analyzing PyrCost , it is convenient to calculate the number of subframes whose maximum depth equals their actual depth and who have no neighbors on the side r .

Lemma 23: (The Edge Size Lemma): The number of subframes of F^i whose maximum depth equals their actual depth and have no neighbors on side r is equal to 2^i .

Proof: This follows immediately from an application of induction on the depth of the quadtree together with the Negative Recessive Lemma.

Now, we are ready to calculate an upper bound on PyrCost .

Theorem 24: (The Pyramid Constant Aggregate Cost Theorem): The function PyrCost applied to a pyramid of maximum depth i is bounded from above by 4. Thus, the sum of the costs of finding the neighbors on the side r of each node in the pyramid is bounded from above by $4^i \cdot 4$.

Proof: We calculate the upper bound on PyrCost by induction of the maximum depth of the pyramid $\langle \langle F^i, C, f \rangle, C', g, h \rangle$, which we denote as P^i . Since there are 4^i leaves in P^i according Theorem 15, together with the definition of a pyramid, the following relations hold:

$$\text{PyrCost}(P^0) = 0$$

$$4^{i+1} \cdot \text{PyrCost}(P^{i+1}) = 4 \cdot 4^i \cdot \text{PyrCost}(P^i) + 2^i \cdot (2 \cdot i + 6)$$

That is, the average cost of finding a neighbor in a single subframe pyramid is zero and the average cost of finding a neighbor in a pyramid of depth $i+1$ can be calculated as follows: the total cost of all neighbor finding in a pyramid of depth $i+1$ is equal to 4 times the total cost of all neighbor finding in a pyramid of depth i plus the additional costs due to neighboring subframes whose nearest common ancestor is the root of the pyramid and subframes that have no neighbor.

The last formula relies on the following observations. For each of the $4 \cdot 4^i$ leaves in P^{i+1} , the cost of finding their neighbors in P^{i+1} is at least as much as it was in P^i . For all but $4 \cdot 2^i$ of these leaves, the cost is the same because both they and their neighbors are embedded in the same quadrant. Of the $4 \cdot 2^i$ leaves who did not have neighbors before being embedded, $2 \cdot 2^i$ of them still do not have neighbors, but it takes one step longer to verify it. The other $2 \cdot 2^i$ leaves must now have neighbors, and these neighbors are $i+2$ steps farther away than the cost of verifying that they didn't have neighbors in P^i .

If we divide both sides of the last formula by 4^i , we get

$$\text{PyrCost}(P^{i+1}) = \text{PyrCost}(P^i) + (i+3) \cdot 2^{-(i+1)}$$

Using the following two identities

$$\sum_{i=1}^n i \cdot 2^{-i} = 2 - (n+2) \cdot 2^{-n}$$

$$\sum_{i=1}^n 2^{-i} = 1 - 2^{-n}$$

we can derive that $\text{PyrCost}(P^i)$ is less than 4.

The analogous Order Quadtree Depth Neighbors Finding Theorem does not hold

because of the following counter-example. Consider a sequence of quadtrees having the general form shown in Figure 2-27. Note that a constant fraction of the nodes in the tree are adjacent to a vertical line drawn through the middle of the square representing the root of the tree. When east/west neighbor finding is performed for any one of these nodes, we note that the number of nodes visited is proportional to the depth of the node. Thus the total cost of neighbor finding for this example will be $O(N^2)$ for a quadtree of N nodes from this sequence. Thus, the average cost (QuadCost) of finding a neighbor in quadtrees from this sequence will be $O(N)$. Thus, any algorithm that finds a neighbor among a fixed proportion of the subframes will have a worst-case lower bound of the order of the number of nodes in the quadtree squared. In the next section, we will discuss ways of transforming the quadtree to reduce this overhead. But first, it is worth taking a look at an alternate approach that works satisfactorily when the order that the neighbor findings is to find the neighbors of each node visited by a preorder traversal of the quadtree.

If the starting nodes from which the neighbor finding is begun are visited in preorder, then we can construct a recursive algorithm using neighbor-passing techniques shown in [27, 29]. The basic idea is to use the results in Table 2-4 (from the Positive Dominant Lemma) to derive the neighbors of a node from the neighbors of its father. Thus, an algorithm of the form:

```

procedure ALG(X)
begin
    T[N] := NEIGHBOR(X,N);
    T[E] := NEIGHBOR(X,E);
    T[S] := NEIGHBOR(X,S);
    T[W] := NEIGHBOR(X,W);
    ...
    ALG(XNW);
    ALG(XNE);
    ALG(XSE);
    ALG(XSW);
end;
```

could be transformed into:

```

procedure NEWALG(X,T[N],T[E],T[S],T[W])
begin
    ...
    NEWALG(XNW,T[N]SW,XNE,XSW,T[W]NE);
    NEWALG(XNE,T[N]SE,T[E]NW,XSE,XNW);
    NEWALG(XSE,XNE,T[E]SW,T[S]NE,XSW);
    NEWALG(XSW,XNW,XSE,T[S]NW,T[W]SE);
end;

```

Clearly, the above recursive procedure does the neighbor finding portion of its task in time proportional to the number of nodes in the tree. In the next section, we show how to get the same result for more general patterns of node visiting. For example, we are interested in the cost of following the boundary of an object.

NW.NW	NW.NE	NE.NW	NE.NE
NW.SW	NW.SE	NE.SW	NE.SE
SW.NW	SW.NE	SE.NW	SE.NE
SW.SW	SW.SE	SE.SW	SE.SE

Figure 2-1: Layout of indices for F^2

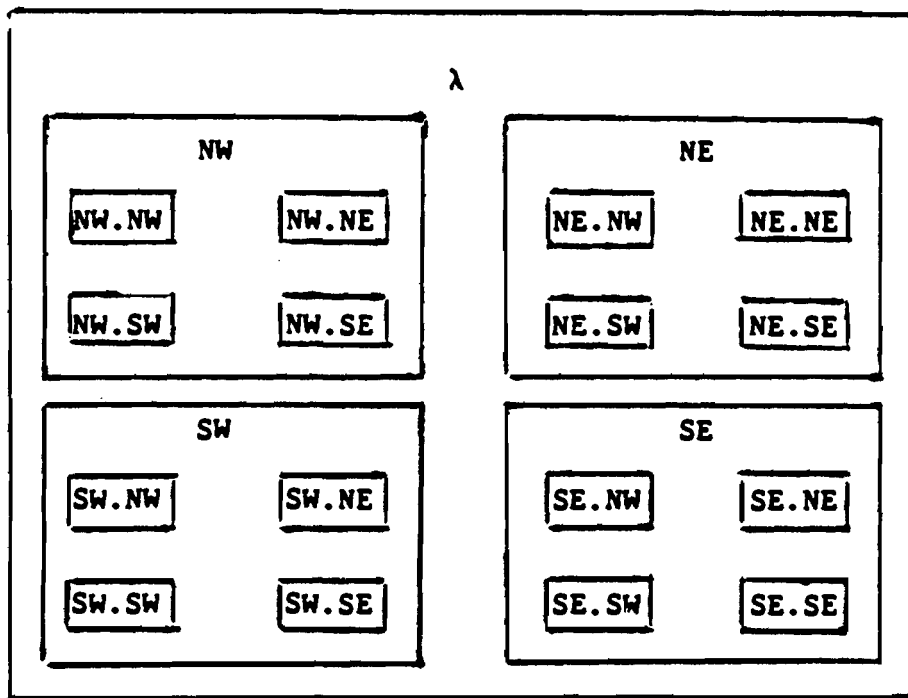


Figure 2-2: Layout of subframes for F^2

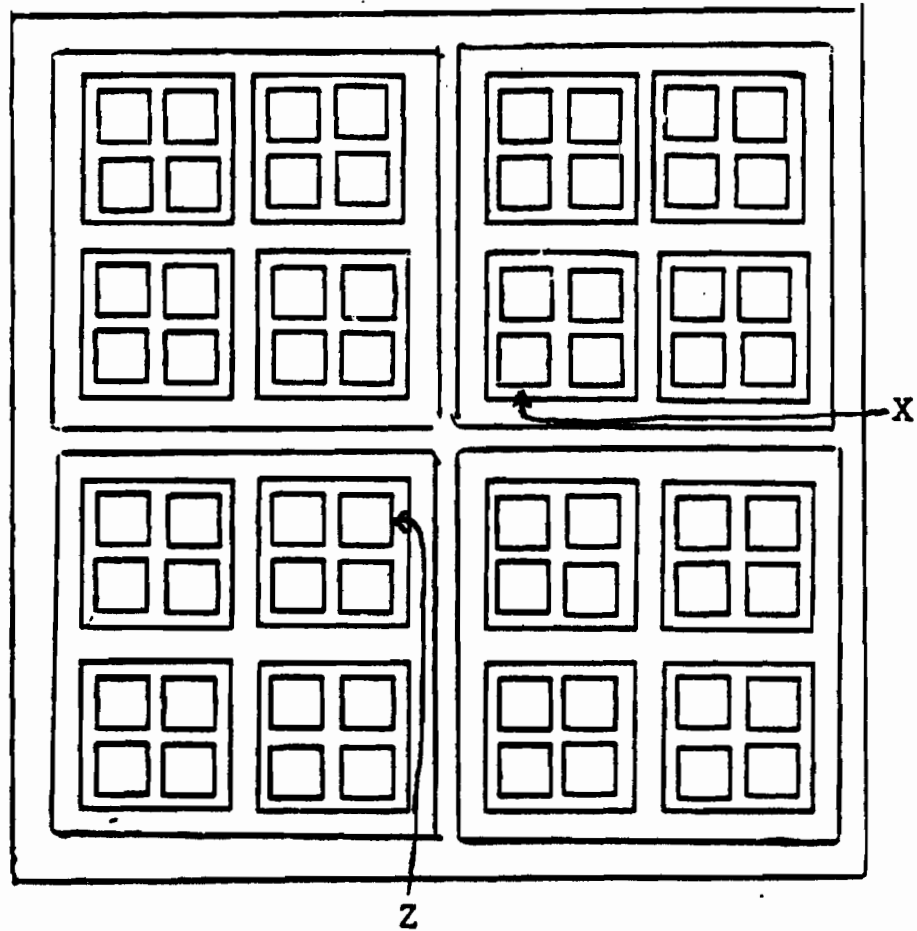


Figure 2-3: Example of a pair of subframes, x and z ,
where $x \neq z$

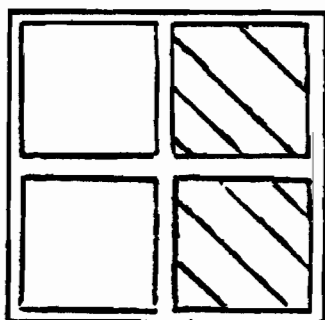


Figure 2-4: Representation of a binary picture denoted $\langle F^1, \{\text{BLACK, WHITE}\}, f \rangle$, with an area of 2 and a perimeter of length 6

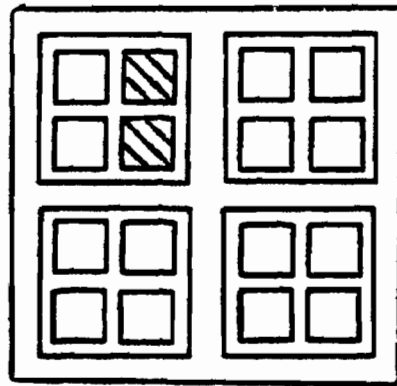


Figure 2-5: Representation of $\text{embed}(\text{NW}, P)$, where P is the binary picture of Figure 2-4

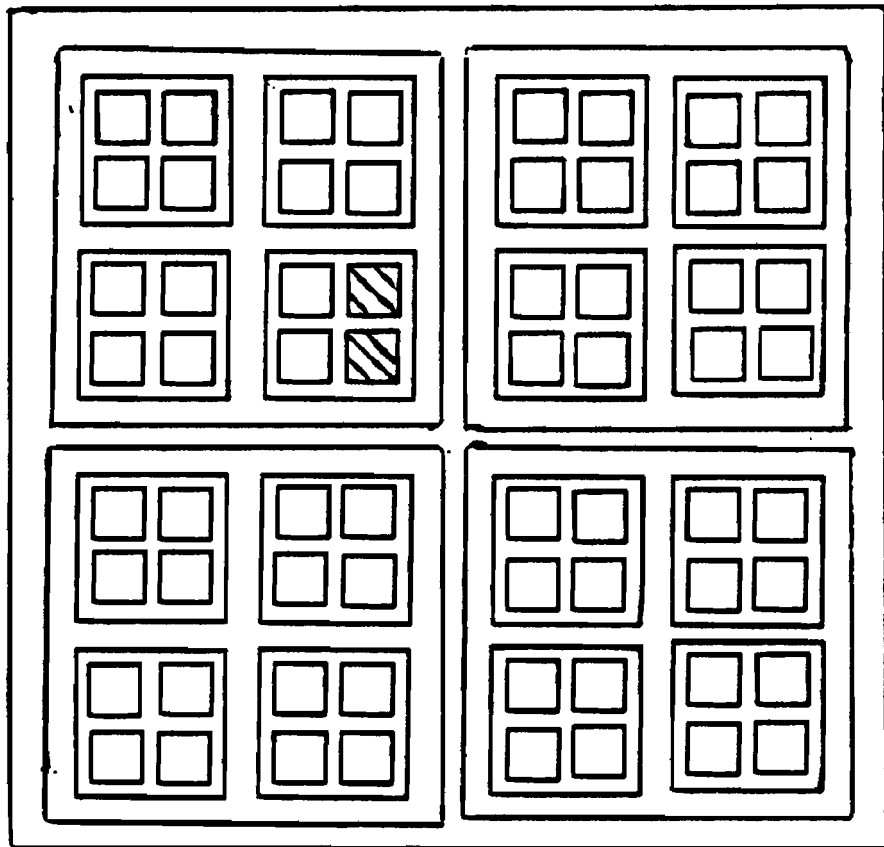


Figure 2-6: Representation of $\text{surround}(P)$, where P is the binary picture of Figure 2-4

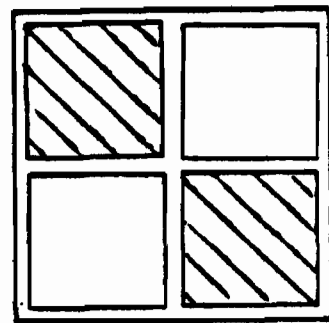
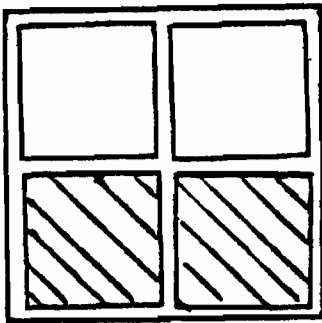


Figure 2-7: Two pictures related via a frame isomorphism

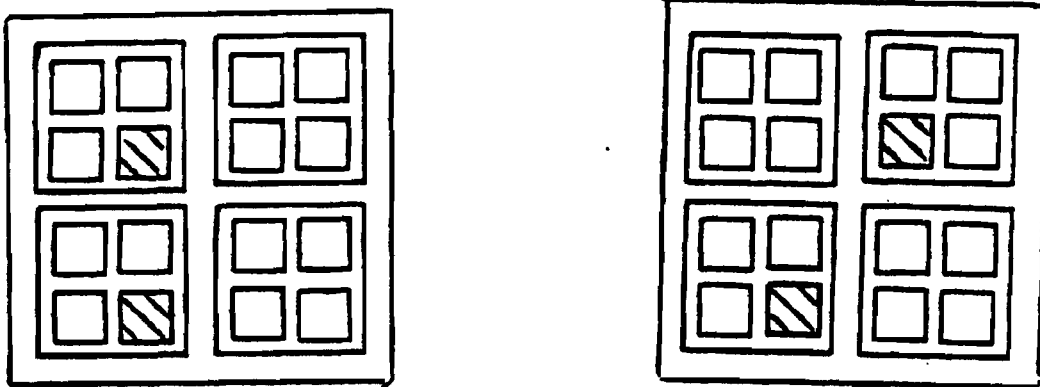


Figure 2-8: Two pictures related via a frame isomorphism preserving neighbor relations among non-white pixels

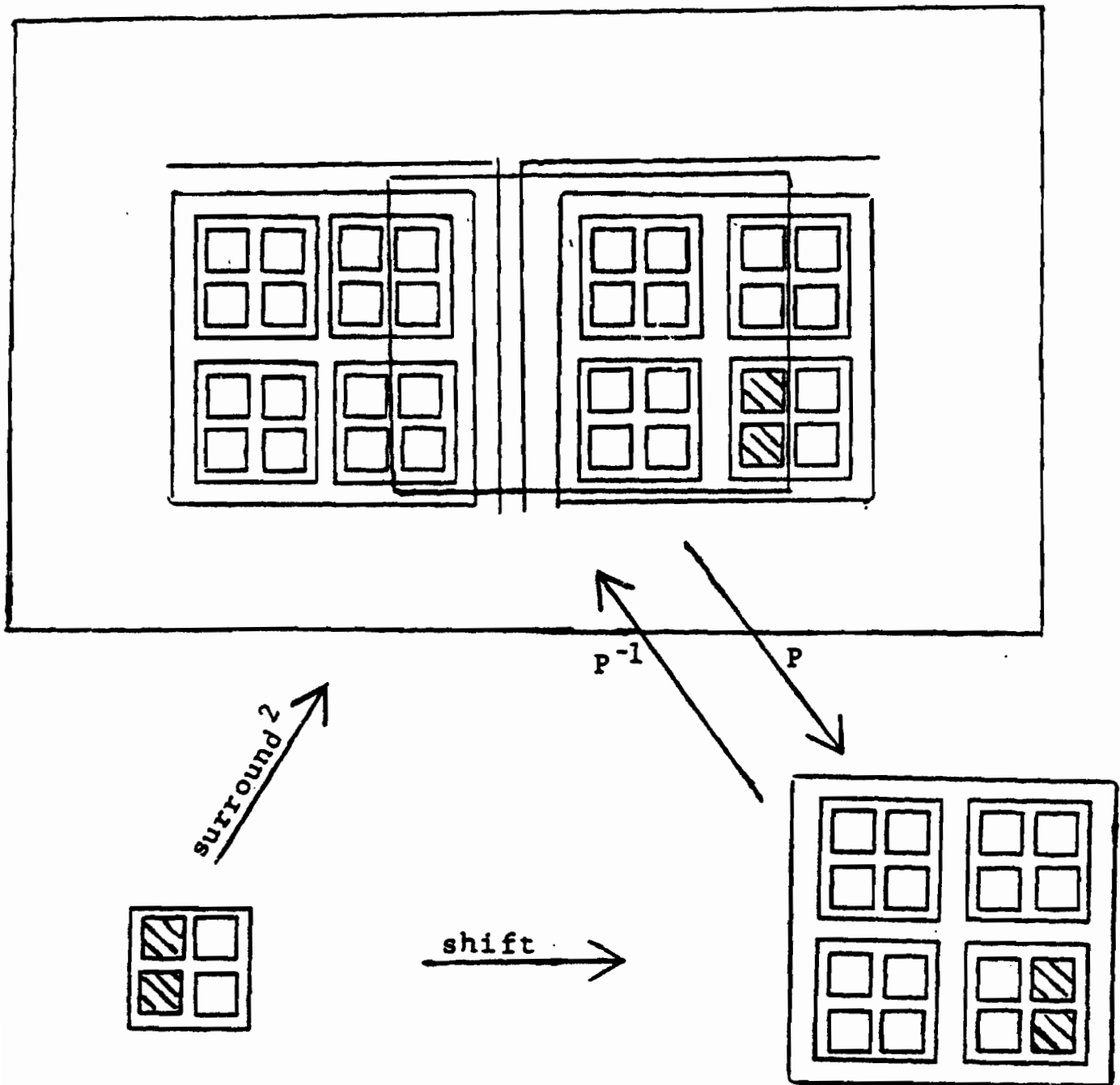


Figure 2-9: Illustration of a shift from $\langle F^1, C, f \rangle$
to $\langle F^2, C, f'' \rangle$

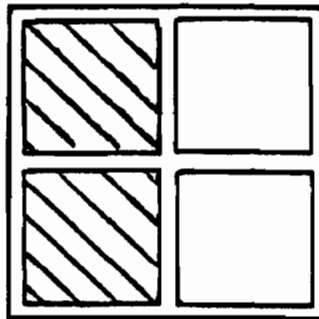


Figure 2-10: Example of a shift of picture in Figure 2-4 with distance 1 in the western direction

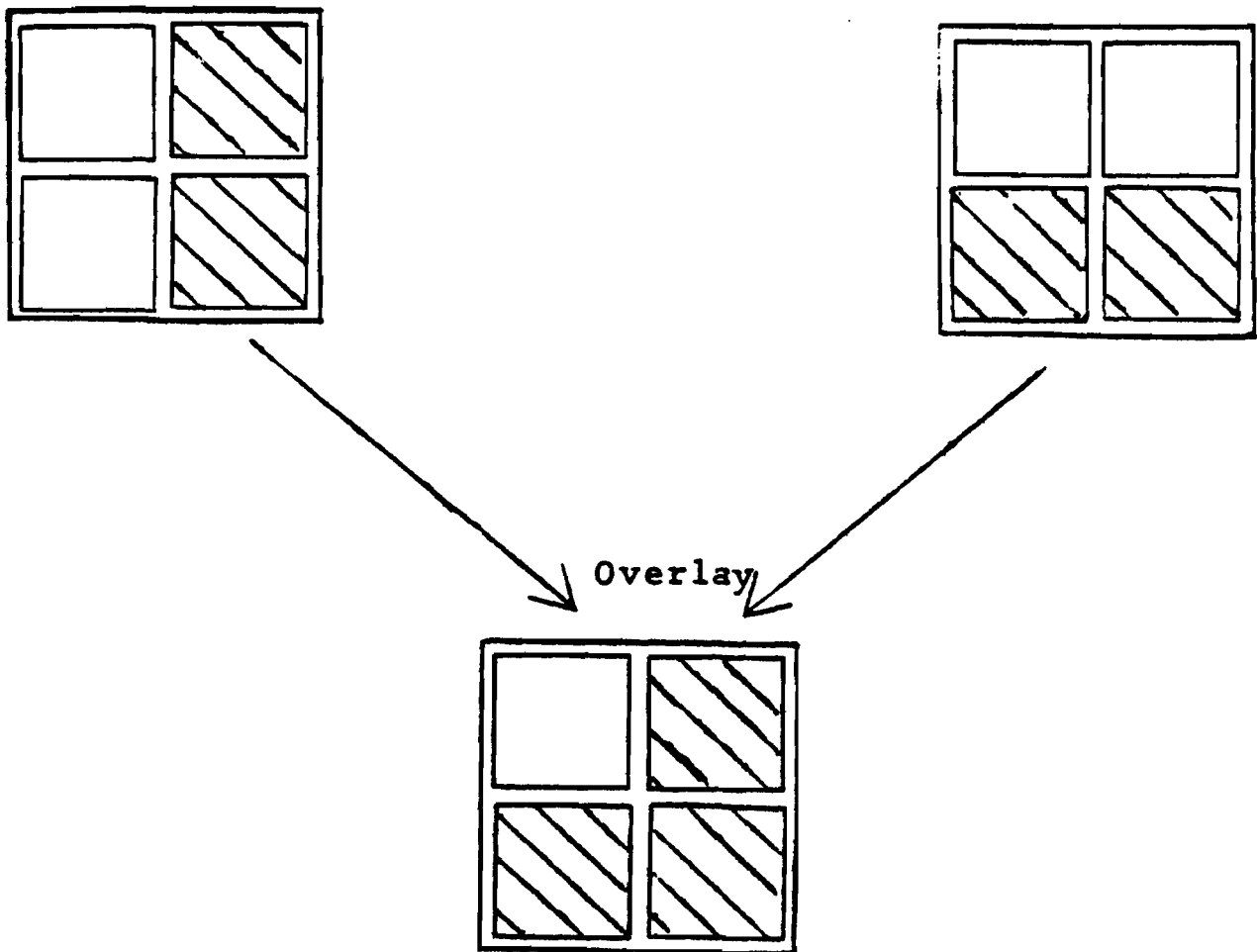


Figure 2-11: The overlay of two binary pictures

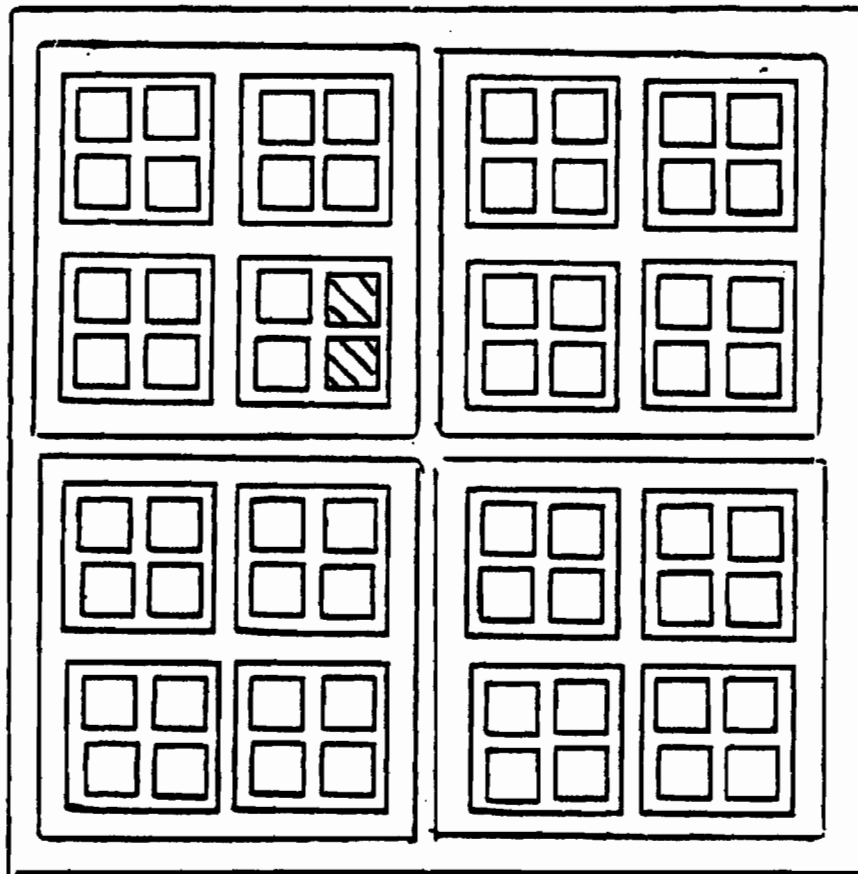


Figure 2-12: Representation of the region pyramid for picture in Figure 2-6

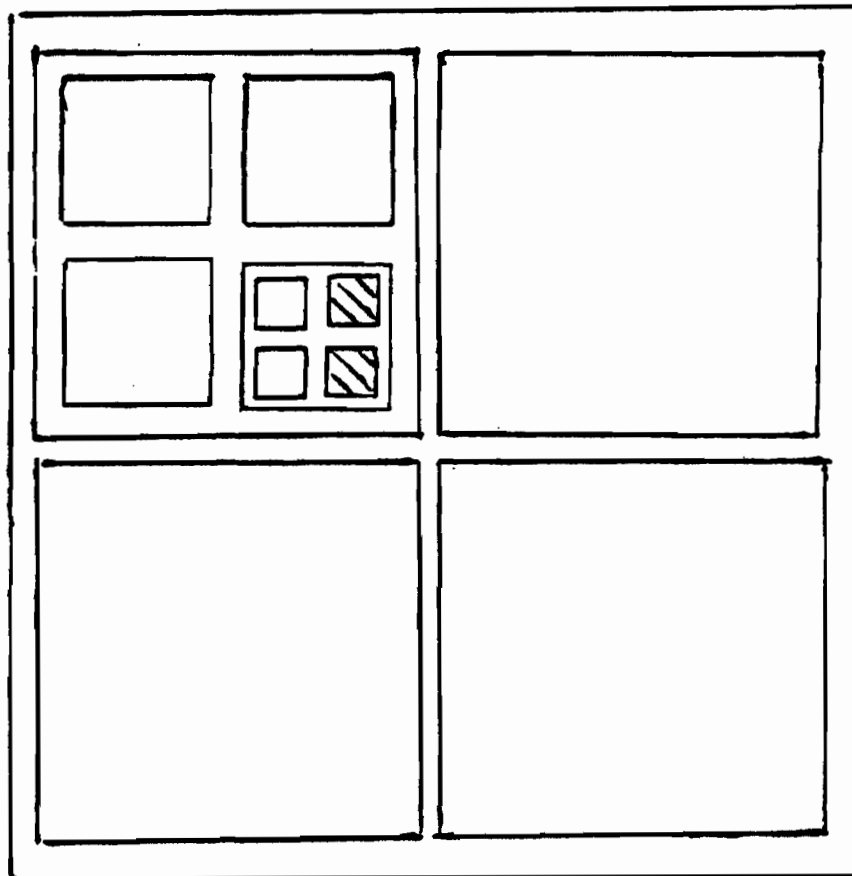


Figure 2-13: Representation of the region quadtree of the pyramid in Figure 2-12

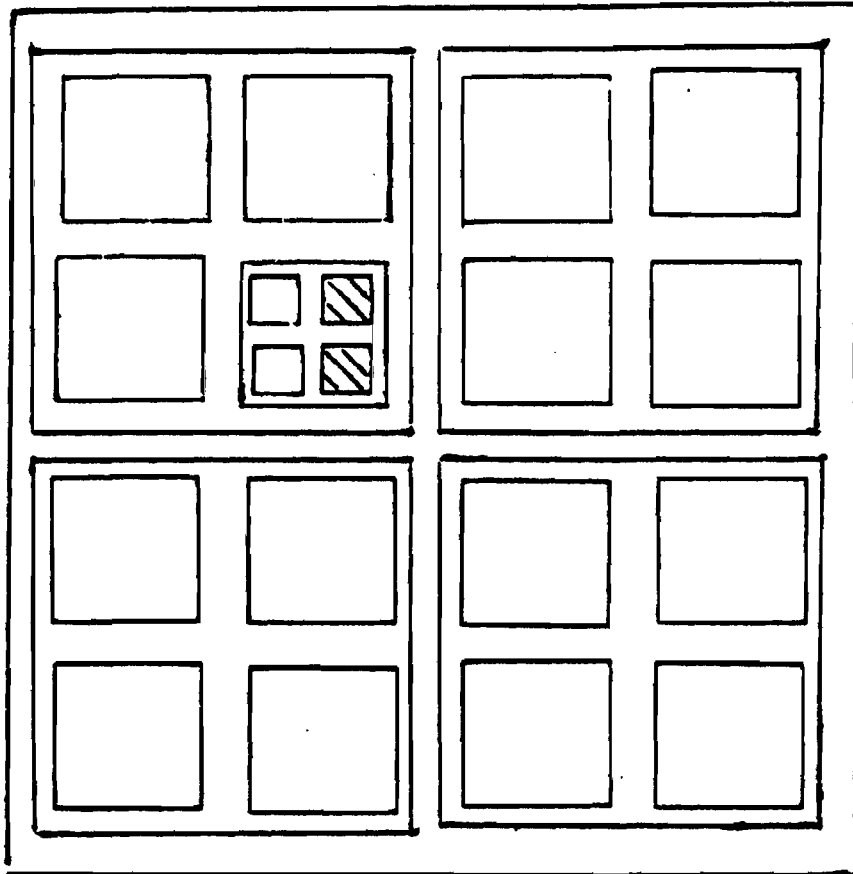


Figure 2-14: Representation of the partial quadtree Q^1 of the pyramid in Figure 2-12

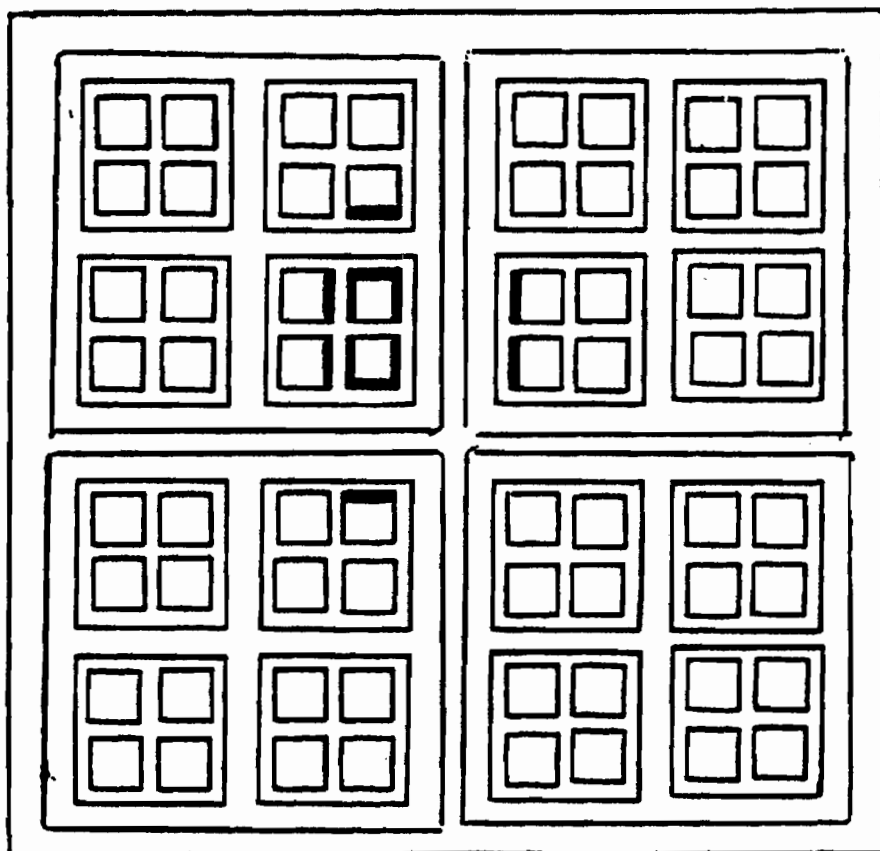


Figure 2-15: Representation of the difference picture of Figure 2-5

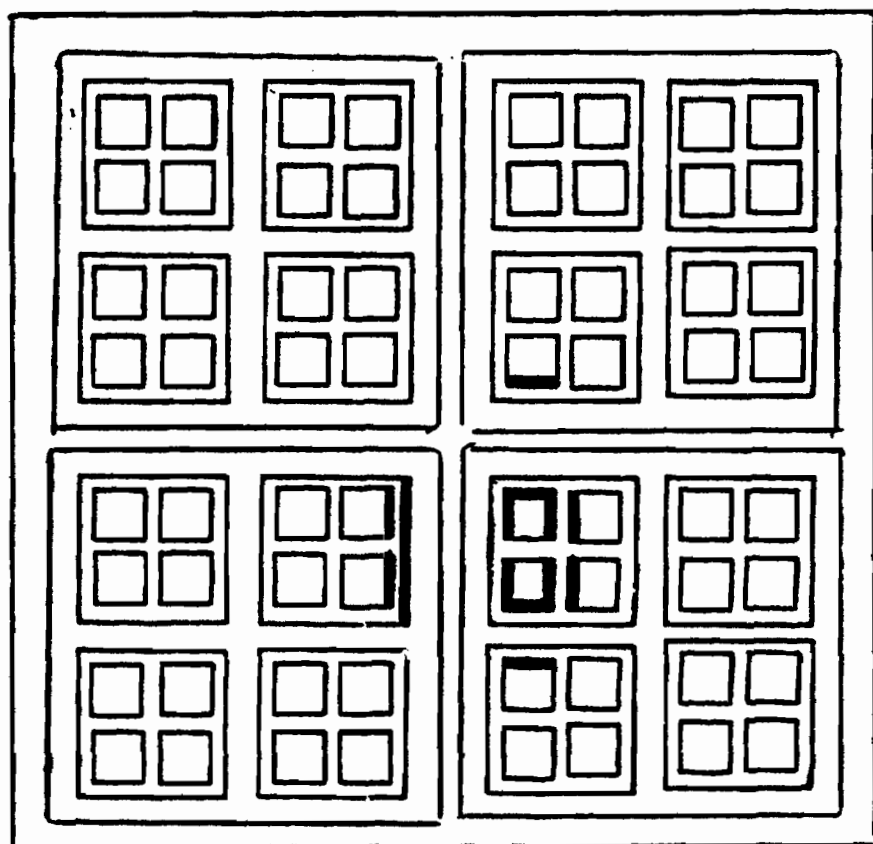


Figure 2-16: Representation of the line pyramid for
Figure 2-15

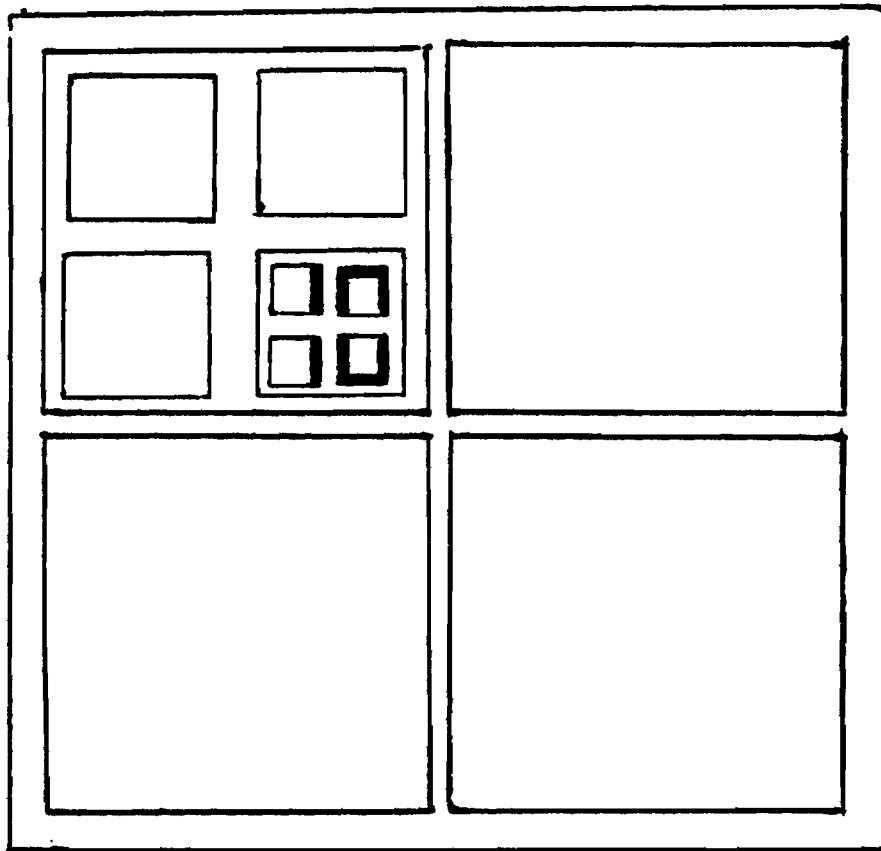


Figure 2-17: Representation of the line quadtree for
Figure 2-16

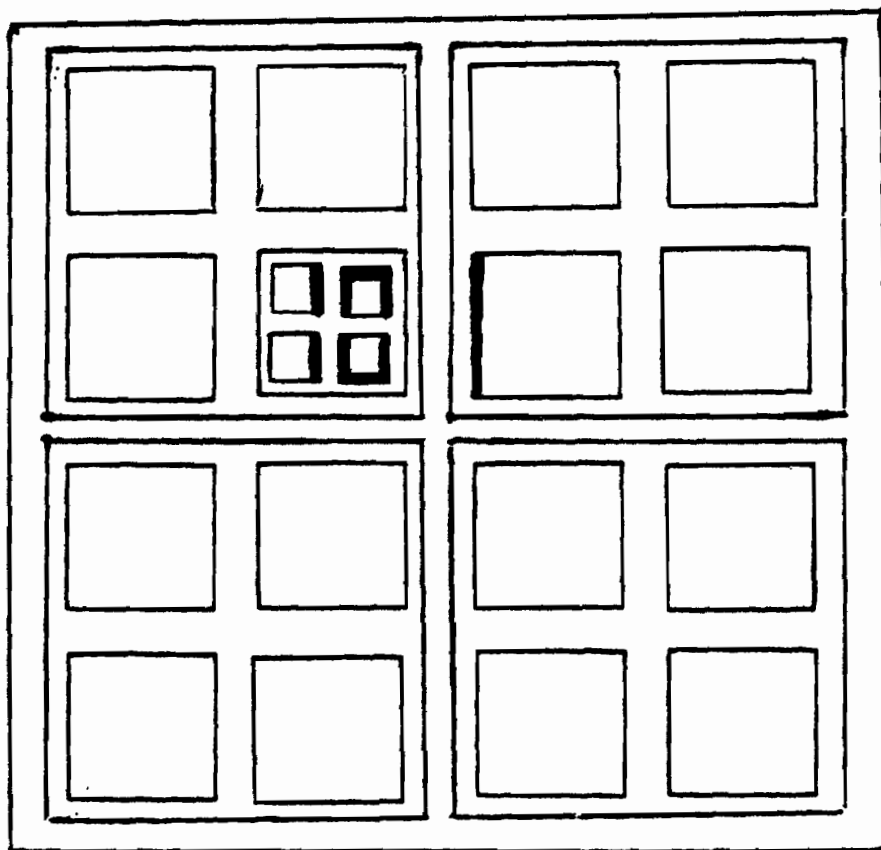


Figure 2-18: Representation of the partial quadtree Q^1 of the line quadtree equivalent of the region quadtree in Figure 2-13

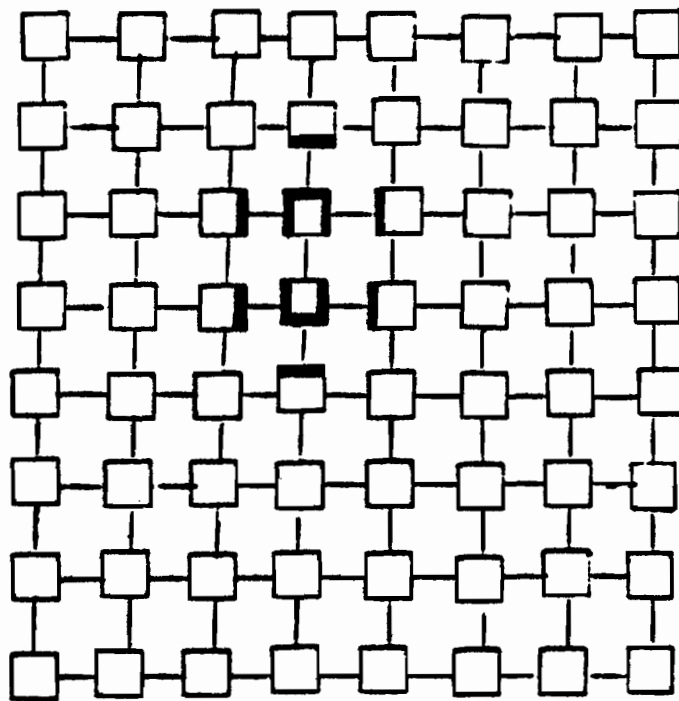


Figure 2-19: Record structure for difference picture of
Figure 2-15

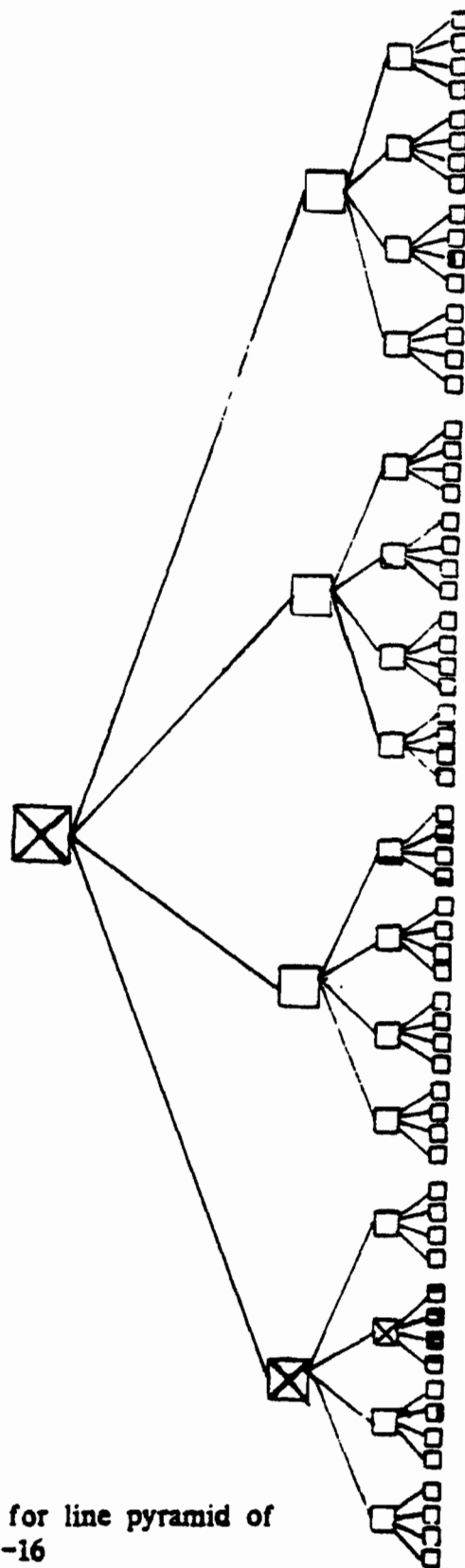


Figure 2-20: Record structure for line pyramid of
Figure 2-16

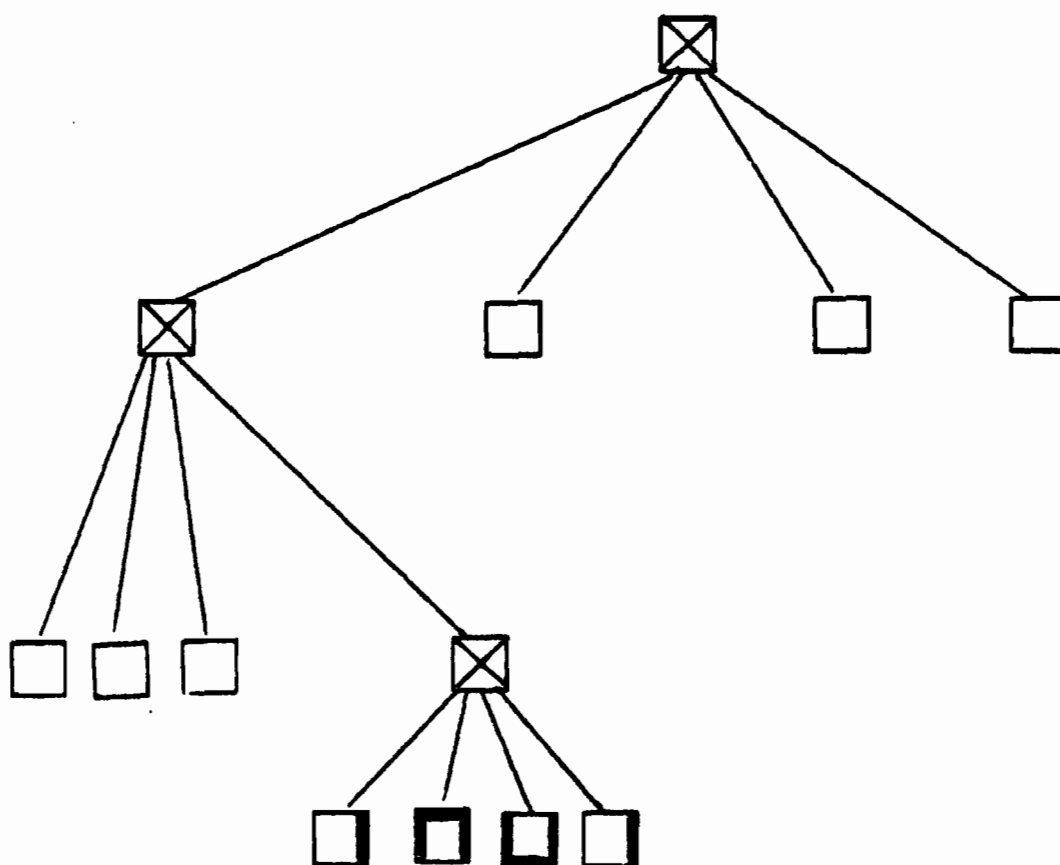


Figure 2-21: Record structure for line quadtree of
Figure 2-17

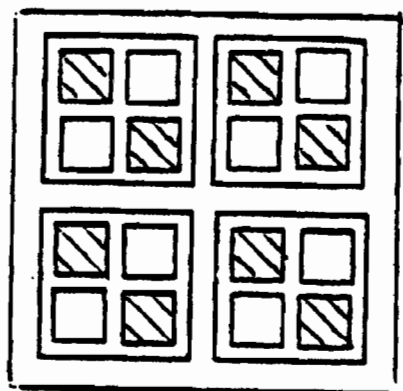
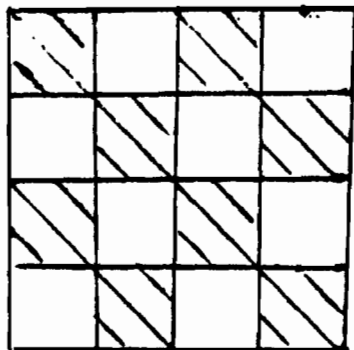


Figure 2-22: Example of 4 by 4 checker-board picture and corresponding region quadtree

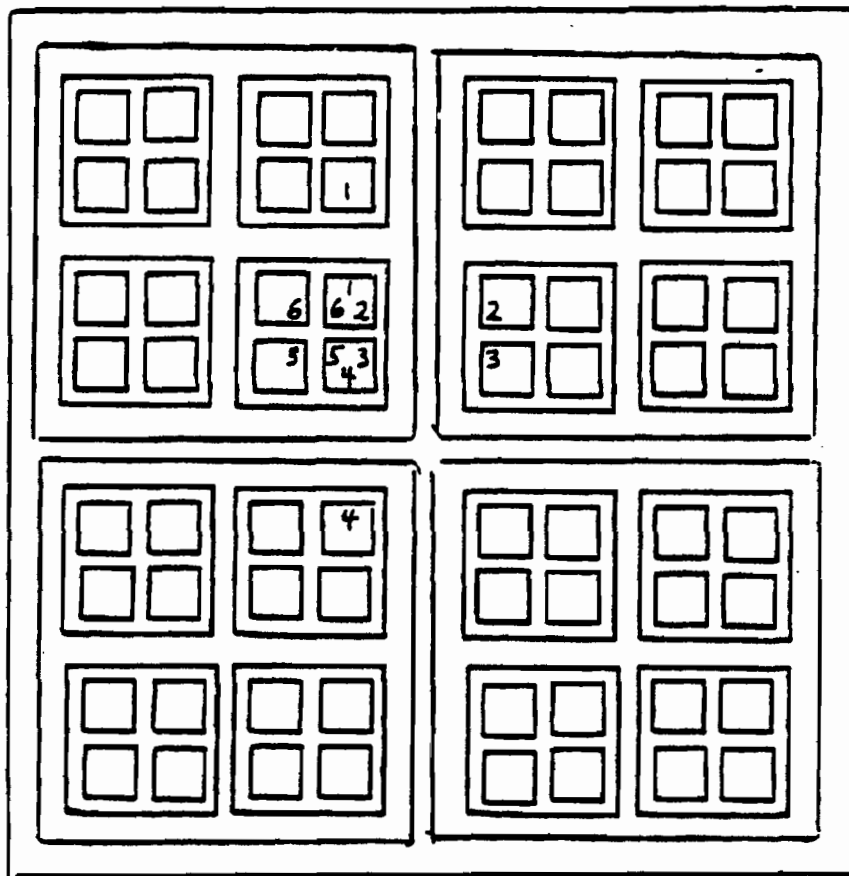


Figure 2-23: Illustration of the path sequence in surround(P) corresponding to Figures 2-4 and 2-6

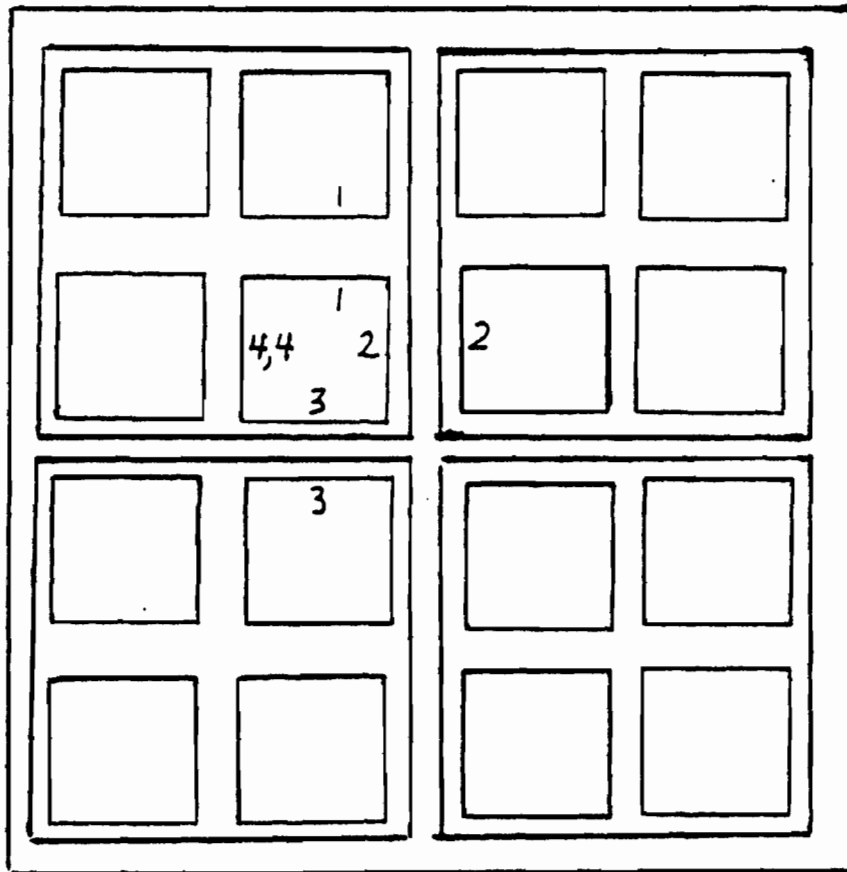


Figure 2-24: Illustration of the reduction of the augmented path of Figure 2-23

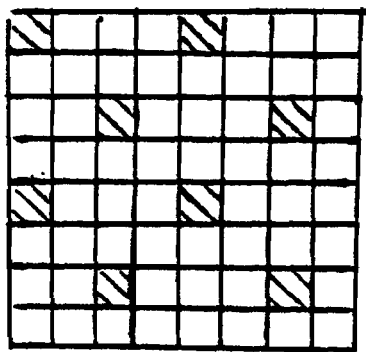


Figure 2-25: Example of a picture of a modified checker-board pattern

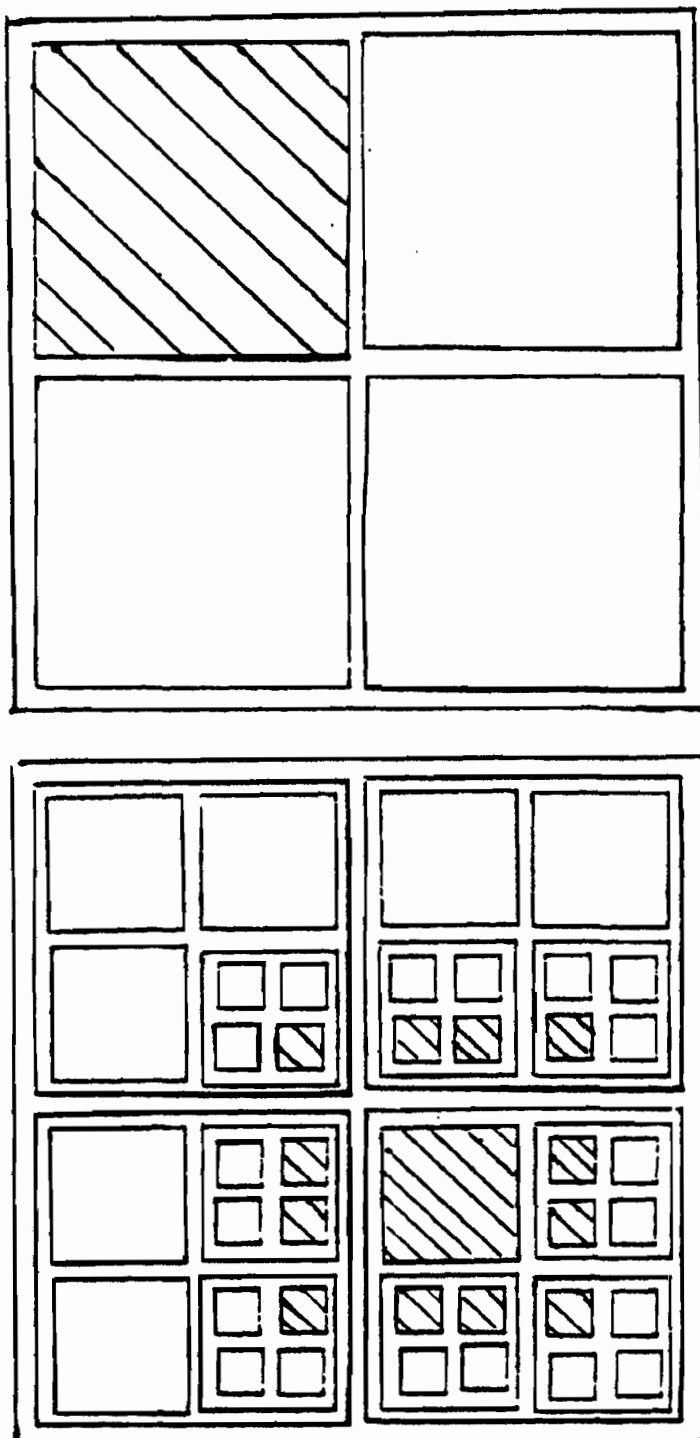


Figure 2-26: Two quadtrees representing a 4x4 square in an 8x8 picture

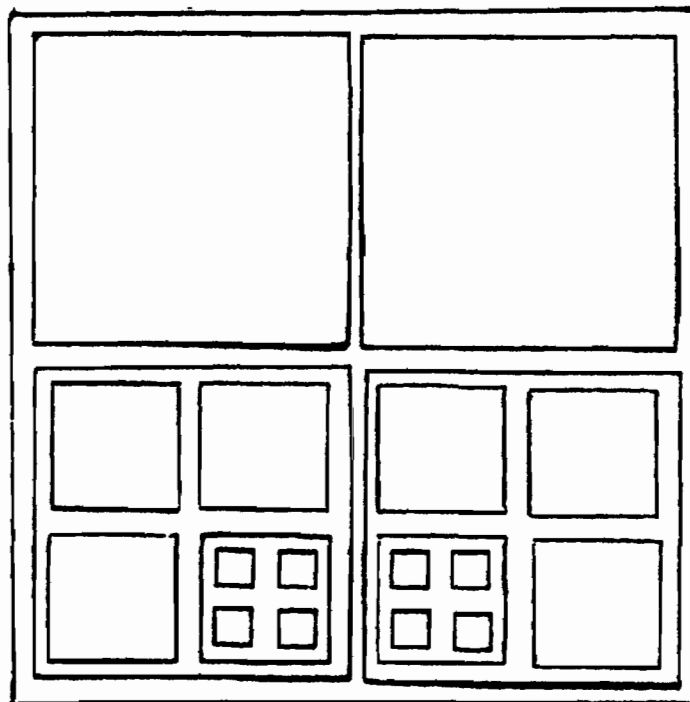


Figure 2-27: Example of quadtree whose aggregate neighbor finding cost is of the order of the number of nodes squared

Table 2-1: TABLE OF INVERSE NEIGHBOR AXIOMS

RELATION	INVERSE RELATION
n (north)	s (south)
e (east)	w (west)
ne (northeast)	sw (southwest)
nw (northwest)	se (southeast)
= (equality)	= (equality)

Table 2-2: TABLE OF NEIGHBOR COMPOSITION AXIOMS

	n	e	s	w	nw	ne	se	sw	=
n	*	ne	=	nw	*	*	e	w	n
e	ne	*	se	=	n	*	*	s	e
s	=	se	*	sw	w	e	*	*	s
w	nw	=	sw	*	*	n	s	*	w
nw	*	n	w	*	*	*	=	*	nw
ne	*	*	e	n	*	*	*	=	ne
se	e	*	*	s	=	*	*	*	se
sw	w	s	*	*	*	=	*	*	sw
=	n	e	s	w	nw	ne	se	sw	=

Table 2-3: A SAMPLE PROOF

STATEMENT	REASON
1) $\text{subframe}(F^2, \text{NW}) \text{ w } \text{subframe}(F^2, \text{NE})$	R1
2) $\text{subframe}(F^2, \text{NW.NE}) \text{ w } \text{subframe}(F^2, \text{NE.NW})$	1, R4
3) $\text{subframe}(F^2, \text{NE.SW}) \text{ s } \text{subframe}(F^2, \text{NE.NW})$	R2
4) $\text{subframe}(F^2, \text{NE.NW}) \text{ n } \text{subframe}(F^2, \text{NE.SW})$	3, Table 2-1
5) $\text{subframe}(F^2, \text{NW.NE}) \text{ nw } \text{subframe}(F^2, \text{NE.SW})$	2, 4, Table 2-2
6) $\text{subframe}(F^2, \text{NW.SE}) \text{ s } \text{subframe}(F^2, \text{NW.NE})$	R3
7) $\text{subframe}(F^2, \text{NW.SE}) \text{ w } \text{subframe}(F^2, \text{NE.SW})$	5, 6, Table 2-2
8) $\text{subframe}(F^2, \text{SW}) \text{ s } \text{subframe}(F^2, \text{NW})$	R2
9) $\text{subframe}(F^2, \text{SW.NE}) \text{ s } \text{subframe}(F^2, \text{NW.SE})$	8, R5
10) $\text{subframe}(F^2, \text{SW.NE}) \text{ sw } \text{subframe}(F^2, \text{NE.SW})$	7, 9, Table 2-2
11) $\text{subframe}(F^2, \text{NE.SW}) \text{ ne } \text{subframe}(F^2, \text{SW.NE})$	10, Table 2-1

Table 2-4: CONSTRUCTION OF SIDE NEIGHBORS GIVEN x r y

	x^{NW}	x^{NE}	x^{SE}	x^{SW}
n	x^{SW}	x^{SE}	y^{NE}	y^{NW}
e	y^{NE}	x^{NW}	x^{SW}	y^{SE}
s	y^{SW}	y^{SE}	x^{NE}	x^{NW}
w	x^{NE}	y^{NW}	y^{SW}	x^{SE}

Table 2-5: PROCEDURAL ENCODING OF TABLE 2-4

```

procedure NEIGHBOR(X,R)
/* where X is a node and R is a side relation.
   note: X and Y do not have the same meaning
   as in Table 2-4. */

begin
  if IERROR(Xf) then
    return(ERROR);
  else Y := Xf;
  case R
    N: if X = YNW then
      return(NEIGHBOR(Y,N)SW);
      else if X = YNE then
      return(NEIGHBOR(Y,N)SE);
      else if X = YSE then
      return(YNE);
      else /* it must be that X = YSW so */
      return(YNW);
    E: if X = YNW then
      return(YNE);
      else if X = YNE then
      return(NEIGHBOR(Y,E)NW);
      else if X = YSE then
      return(NEIGHBOR(Y,E)SW);
      else /* it must be that X = YSW so */
      return(YSW);
    S: if X = YNW then
      return(YSW);
      else if X = YNE then
      return(YSE);
      else if X = YSE then
      return(NEIGHBOR(Y,S)NE);
      else /* it must be that X = YSW so */
      return(NEIGHBOR(Y,S)NW);

```



```

/*  note:  this is a continuation of TABLE 2-5 */
W:  if X = YNW then
      return(NEIGHBOR(Y,W)NE);
    else if X = YNE then
      return(YNW);
    else if X = YSE then
      return(NEIGHBOR(Y,W)SW);
    else /* it must be that X = YSW so */
      return(YSE);
    end__of__case
end;

```

CHAPTER 3

NORMALIZING QUADTREES WITH RESPECT TO AGGREGATE NEIGHBOR FINDING

3.1. DEVELOPMENT OF ANF ALGORITHMS FOR PYRAMIDS

3.1.1. THE OPTIMAL POSITIONING OF CHAIN CODES

The notion of a path, presented in Section 2.3, can be used as the basis for a data structure for storing images. Such a data structure has been called a chain code [8]. The important aspect of the chain code with respect to quadtrees is that it specifies a sequence of subframes relative to an arbitrary first subframe (i.e., the first picture element). Assume that the chain code is presented as a string over an alphabet consisting of the set {N, E, S, W} of directions. Then, the sequence of subframes associated with a chain code is defined as follows:

1. The sequence of subframes associated with the empty string is $\langle f_0 \rangle$ where f_0 denotes the arbitrary first subframe.
2. The sequence of subframes associated with the string $\alpha\beta$, where α is an arbitrary string over our alphabet and β is a single element of our alphabet is the concatenation of the sequence $\langle f_0, \dots, f_{i-1} \rangle$ corresponding to α with the singleton sequence $\langle f_i \rangle$, such that the relation between f_{i-1} and f_i is defined as follows:
 - a. if $\beta = N$, then $f_i \text{ n } f_{i-1}$
 - b. if $\beta = E$, then $f_i \text{ e } f_{i-1}$
 - c. if $\beta = S$, then $f_i \text{ s } f_{i-1}$
 - d. if $\beta = W$, then $f_i \text{ w } f_{i-1}$

Thus, the sequence of subframes marked in Figure 3-1 indicates the sequence of subframes referred to by the chain code NNNNNEESSWSSESSWWWN once f_0 is chosen to be the subframe whose root is SW.SW.NE. This chain code can be viewed as

defining the object that results from associating the color BLACK with each subframe visited by it or marked with an X in Figure 3-1. The length of this chain code corresponds to the length of the perimeter of the object defined by the previous sentence. Note that this is a slight variation on the usage of boundary codes in other quadtree papers [25, 4] in that in this case, the code describes steps from one pixel to the next, instead of describing steps along the side of a pixel.

It turns out that when calculating the cost of traversing the subframes referred to by a chain code in a pyramid or quadtree, the initial assignment of the subframe f_0 is critical. A bad choice can result in excessive neighbor-finds that are performed at maximum cost. For example, in Figure 3-2, we see that we are constantly going back and forth among subframes in two columns whose nearest common ancestor is the root of the tree. However, the chain code with a better choice of initial subframe could result in Figure 3-3.

The task of this chapter is to show how to calculate the best location for the initial subframe. The best location for the initial subframe is specified as a location where the total cost of traversing the sequence of subframes defined by the chain code is proportional to the length of the chain code. A convenient measure of cost for quadtree algorithms is the number of nodes visited. Note that if a node is visited more than once, then it is counted more than once.

The algorithm that computes this location for the f_0 subframe of the chain code is called the Aggregate Neighbor Finding Transformation. The result of using this algorithm to shift the path in Figure 3-1 is shown in Figure 3-4. The following should be clear about the ANF Transformation: the amount of shift in the x direction is totally independent of the N/S steps in the chain code (and likewise for the y direction and the E/W steps). Thus, if we consider the f_0 to be initially corresponding to the point $\langle 0,0 \rangle$, we can then proceed to calculate the appropriate shifts according to the following recursive relation. Assume without loss of generality that we are shifting with respect to the x-axis.

```

procedure ANF(<X,Y>,CURRENTSHIFT,MAXSHIFT,CHAIN)
/* ANF calculates the correct shift for the chain code
CHAIN by considering each possible power of 2 shift
and making a shift only when it improves the current
cost. <X,Y> denotes the current start of the chain
code. CURRENTSHIFT denotes the current power of 2
that we are considering shifting by. MAXSHIFT is
the upper bound on CURRENTSHIFT. The cost of shifting
is stored in ONEOFF and the cost of staying in place
is stored in STAYPUT. Note that we are only con-
sidering the amount of shift in the E-W direction.*/
begin
  integer ONEOFF, STAYPUT;
  if CURRENTSHIFT = MAXSHIFT then return <X,Y>
  else begin
    STAYPUT := NUMBER OF FIND-NEIGHBORS COSTING EXACTLY
              2·CURRENTSHIFT, GIVEN THAT CHAIN
              STARTS AT <X,Y>;
    ONEOFF := NUMBER OF FIND-NEIGHBORS COSTING EXACTLY
              2·CURRENTSHIFT, GIVEN THAT CHAIN
              STARTS AT <X+2CURRENTSHIFT,Y>;
    if STAYPUT > ONEOFF
      then return ANF(<X,Y>,CURRENTSHIFT+1,
                     MAXSHIFT,CHAIN)
    else return ANF(<X+2CURRENTSHIFT,Y>,
                   CURRENTSHIFT+1,MAXSHIFT,CHAIN)
  end;
end;

```

Recall the Pyramid Constant Aggregate Cost Theorem of Section 2.4. There, we noted that if all possible neighbor findings were performed (in a pyramid), then the aggregate cost (i.e., the number of nodes visited) would be bounded from above by 4 times the number of neighbor findings done. Our chain code can be viewed as a specification of a skewing of the distribution of neighbor findings done. The ANF algorithm shifts the quadtree so that the most frequently performed neighbor findings have the cheapest costs.

Thus, for example, on the first invocation of the ANF procedure, we count the number of E/W neighbor findings done at every other column along the x-axis, starting at either <0,0> or <1,0>. We, then, compare these two groups of neighbor findings and shift the image so that the largest group of neighbor finding can be done in 2 steps (i.e., the father of one of the nodes involved is also the nearest common ancestor). Note that in the Pyramid Constant Aggregate Cost Theorem, exactly half of the

neighbor findings could be performed in 2 steps. Due to the shift performed by the ANF algorithm, at least half of the neighbor findings are performed in 2 steps. Thus, as the ANF algorithm processes the chain code, it constantly skews the cost of neighbor finding toward the least cost. Hence, when the transformation is completed, the cost of the neighbor finding performed is bound from above by 4 times the number of neighbor findings done.

The main properties of the above algorithm are outlined in the following two theorems.

Theorem 1: (The ANF Chain Code Transformation Theorem): For any path A , there exists a shifting of the path, denoted A' that can be traversed in the pyramid in time less than 4 times the length of A .

Proof: Following reasoning similar to the Pyramid Constant Aggregate Cost Theorem, we note that if half of the neighbor finding operations cost 2, half of the remaining cost 4, half of the remaining cost 6, etc., then the total cost of all the neighbor finding operations will be bounded from above by four times the length of the perimeter of the object. Such a chain code corresponds to one that performs the find-neighbor operation between adjacent nodes the same number of times per node pair. Now, if some of the costs are weighted more heavily than is dictated by this breakdown, then the total sum will be less than the cost of four times the length of the perimeter, because the ANF algorithm places the heaviest weights on the lowest path-length costs.

There remains the question of how expensive it is to calculate the ANF transform.

Theorem 2: (The Linear Time ANF Chain Code Theorem): The algorithm presented in Theorem 1 executes in time proportional to the length of the chain code.

Proof: The cost of the ANF algorithm depends on how the counting of the number of find-neighbors with a given cost is performed. The simplest way to count these find-neighbors is to use a 1-dimensional array whose size is twice the smallest power of 2 that bounds the length of the chain code from above. Hence, for a chain code of length 3, we would use an array of size 8. Starting at the middle of the array, and traversing the chain code, count how many find-neighbors cross each column orthogonal to the x-axis at each position. Then, to determine how many nodes have a given cost $2 \cdot N$ in a chain code beginning at $\langle X, Y \rangle$, the algorithm need only start at the position X and sum the values stored in the array for each position $X \cdot k \cdot 2^N$ where k runs from 0 to the size of the array divided by 2^N . Thus, the number of entries in the array that have to be visited by one recursive invocation is at most half those visited by the previous invocation. Therefore the cost of the algorithm is proportional to the length of the array used for counting plus the length of the chain code. Note that the length of the

chain code times 4 is an upper bound on the length of the array used for counting.

The next section begins the interface between the ANF transform and quadtrees.

3.1.2. THE CHAIN CODE TO QUADTREE ALGORITHM

Based on the chain code to quadtree conversion algorithm presented in [25] together with the ANF transformation, we deduce an alternative algorithm to accomplish the same task whose worst-case performance is proportional to the length m of the chain-code. This is an improvement over the $O(m \cdot \log_2(m))$ worst-case performance of the conversion algorithm [25] that occurs without the preprocessing done by the ANF transformation. Since the ANF algorithm is $O(m)$, the only reason for not applying it is that it might cause a collection of maps to be represented by unaligned quadtrees. Note that many algorithms (e.g., intersection) are more difficult when performed on quadtrees whose lower left hand corners do not correspond to the same global coordinate. That is, given a collection of quadtrees, it might be more important to make sure that all the quadtrees have the same global coordinate for their lower lefthand corner (see Section 4.1) than it is to optimize aggregate neighbor following.

Theorem 3: (Linear Time Chain Code to Quadtree Theorem): There exists an algorithm that builds a quadtree from a single connected chain code in time proportional to the length of the chain code.

Proof: The initial coloring of the nodes on the boundary of region in the quadtree can be done in time proportional to the length of the chain code when the chain code has been positioned by the ANF algorithm. From Hunter's Theorem, the number of nodes in the quadtree is proportional to the length of the chain code. The interior nodes of the region can be colored by a preorder traversal of the quadtree that passes down neighbors as described in Section 2.4 in time proportional to the number of nodes in the quadtree. Thus, the entire algorithm is bounded from above by a constant times the length of the chain code.

When constructing a quadtree from a chain code, before we perform the possible merging of nodes that are adjacent to the perimeter of the object, the number of nodes in the quadtree is proportional to the length of the chain code. After these nodes have been merged (where possible), it might be the case the the number of nodes in the quadtree is considerably less than the length of the chain code. Thus, although we were able to build the quadtree in time proportional to the length of the chain code,

we are not able to traverse the boundary in time proportional to the number of nodes in the quadtree. However, there is an alternative approach to defining the chain code for the boundary of an object that will permit traversal in time proportional to the number of nodes in the quadtree. This occurs by including in the chain code both a traversal of the WHITE nodes adjacent to a boundary and a traversal of the BLACK nodes adjacent to the boundary (with the provision that when this traversal bends around a corner, it first overshoots the corner by one step).

An example of the extended chain code corresponding to Figure 3-1 is shown in Figure 3-5. It should be noted that this extended chain code exactly describes a path that a perimeter-following algorithm might traverse. In general, assuming that the contribution from overshoots is insignificant, the length of this extended chain code is twice the length of the original chain code. However, because there exist images where the number of "overshoots" dominates the size of the extended chain code, we can only guarantee that it will be no larger than sixteen times the length of the original chain code (note that this still means that they are proportional in length). For example, we note that the chain code of Figure 3-1 is of length 19, whereas the corresponding extended chain code of Figure 3-5 is of length 54 (including 20 overshoots)

The optimality of placement resulting from the original application of the ANF algorithm is preserved under the node collapsing criteria of the region/line quadtree as shown in Figure 3-6. As a result of this preservation, we can calculate the chain code from a quadtree of a connected image in time proportional to the length of the chain code (cf., [4]). This is summarized by the following theorem.

Theorem 4: (Linear Time Chain Code From Quadtree Theorem):
Given a quadtree that has been constructed from an extended chain code by the algorithm in Theorem 3, it is possible to regenerate the original chain code in time proportional to its length.

Proof: Consider the number of links crossed in the traversal of a path in a pyramid. Now, consider the result of merging nodes in the pyramid in order to build a quadtree. Note that the number of find-neighbors done by the perimeter following algorithm corresponds to the number of links in the extended chain code. Each time four sons merge, the cost of traversing the perimeter reduces as links merge into longer links (note that the boundary of a link is precisely where it crosses the boundary of a quadtree node). Note that for each link removed from the path because it crossed between two nodes that are now part of the same node, there is a corresponding reduction

in the aggregate neighbor following cost for an extended chain code by four (or more), thus preserving (from Theorem 1) the average cost of four times the number of links in the extended chain code which is proportional to the number of nodes in the quadtree. Thus, the usage of the ANF algorithm on the extended chain code (henceforth always referred to as the chain code) will satisfy the requirements of this theorem.

Often, when dealing with real data, we must consider maps that represent a collection of disconnected objects. If the notion of a chain code is extended to be a collection of chain codes prefaced by relative addresses indicating the location of each chain code, then we note that the length of these addresses is proportional to the depth of the resulting quadtree. Thus, we can derive results analogous to those above for disconnected maps. Note that the cost of traversing the perimeter of some of the regions in the map may be quite large. However, the cost of traversing all the region's perimeter is still less than four times the sum of the length all these perimeters.

3.2. GENERALIZATION OF ANF ALGORITHM TO DIRECTLY TRANSFORM QUADTREES

3.2.1. OPTIMAL POSITIONING OF QUADTREES

The ANF algorithm for directly manipulating quadtrees is based on the fact that the contribution of a line segment of exactly 2^n to the ANF algorithm does not effect the costs calculated for any shift of length less than 2^n . The ANF algorithm for quadtree to quadtree conversion proceeds as follows (again we break the algorithm into solving for the x and y shifts separately, only showing one solution):

1. Perform a preorder traversal of the tree that passes down neighbors to determine the location of the steps used in the border following procedure. Build a linked list for each level of the count of steps that would be taken that correspond to the width of a leaf at that level. This linked list is the direct analog to the array described in the previous theorem. Since the access to the information stored is sequential, the linked list and the array can be used interchangeably.
2. Add additional nodes to the lowest level to take care of overshoots. Note that we are using an extended chain code.
3. Perform the ANF algorithm on the linked list of the smallest pixel width columns. Since we are working with linked lists instead of arrays, the counting is done differently. The process starts at the beginning of the linked list and traverses the list forming two lists, corresponding to the two groups of find-neighbors being compared.

4. After making the appropriate decision about shifting, merge the list corresponding to the least number of steps with the list for the next higher level. Note that the list with the larger total contains information that is no longer necessary to the calculation of the appropriate shift.
5. Repeatedly apply the previous two steps until the list corresponding to the root of the tree has been reached. By now, the ANF shift position $\langle X, Y \rangle$ has been calculated.
6. Shift the input quadtree so that its lower lefthand corner corresponds to the position $\langle X, Y \rangle$. This can be done by a preorder traversal of the input quadtree in time proportional to the size of the input and the output quadtrees.

The analysis of the above algorithm constitutes the following theorem.

Theorem 5: (The Linear Time Quadtree To Quadtree ANF Transformation Theorem): Given a quadtree, we can construct the result of reading the chain code from the quadtree, performing the ANF transformation on the chain code, and then re-encoding the chain code as a quadtree in time proportional to the size of the input and output quadtrees.

Proof: The above algorithm is correct because of the property of large line segments of length 2^n mentioned at the beginning of this section. The algorithm executes in linear time, because it visits each node in the two quadtrees a fixed number of times.

We now have a transformation of an input quadtree into an output quadtree, where the output quadtree's perimeter can be traversed in time proportional to the number of nodes on the perimeter of the quadtree. The amount of shifting necessary to produce this output quadtree can be calculated in time proportional to the size of the input quadtree.

3.2.2. USING ANF NORMALIZATION IN OTHER ALGORITHMS

Since connected component labelling is usually done on an unlabelled map, it is natural to view this task as one of labelling a line quadtree. Note that the above results for binary region quadtrees carry over to line quadtrees.

Theorem 6: (The Linear Connected Component Theorem): Connected component analysis can be performed on an ANF balanced line quadtree in time proportional to the number of nodes in that quadtree.

Proof: The basic algorithm is to use the ANF transformation on the quadtree to produce a shifted quadtree in time proportional to the number of nodes in both the original and the shifted quadtrees. Next, the quadtree is traversed in preorder, until an unlabelled border is found. Then the border

is traversed and each node along it is labelled (note that this can be done in linear time due to the ANF transformation). Then, the process returns to the original preorder traversal of the quadtree that passes down neighbors to use in determining how the colors propagate through the structure. It will always be the case that if a node is not on a border, then all of its neighbors that are colored will have the same color, which is also the appropriate color to label said node.

The preorder traversal that passes down neighbors can be performed in time proportional to the number of nodes in the quadtree. The traversal of the perimeters of all the regions in the quadtree can be done in time proportional to the number of nodes in the quadtree. Thus, the entire procedure can be performed in time proportional to the number of nodes in the quadtree.

The worst-case analysis of the above algorithm is considerably better than that of [26] assuming that the number of nodes was not more important in the total analysis than keeping the average cost per node down was. As was noted in [26], given a connected component algorithm, there exists a simple modification of the connected component algorithm that produces a function for calculating the Euler number of the image.

Also, the ANF transformation can be used to derive yet another algorithm that calculates the perimeter of an object in time proportional to the number of nodes in the transformed and untransformed quadtrees.

Alternatively, the ANF quadtree to quadtree transformation could be used to speed up the quadtree to chain code conversion process given in [4]. The resulting algorithm would be linear in the sum of the size of the transformed and the untransformed quadtrees.

There are two reasons why we might choose not to perform the ANF transformation in a practical application. First, it is often the case that the size of the transformed quadtree is considerably larger than the size of the untransformed quadtree. We will consider this point in more depth in the next section. Second, if this shifting technique is used heavily in a database of quadtrees, then unaligned intersections and unions are being constantly performed along with transformations to maintain consistency. Thus, it would be worthwhile to investigate the overhead involved with tasks like intersection of unaligned quadtrees.

3.2.3. COMPARISON OF ANF TRANSFORM WITH NODE MINIMIZATION

Note that in Section 2.3 we discussed the node minimization algorithm NM [20]. Referring back to the costs associated with the NM transformation, we note that it is significantly more expensive than the ANF transformation. The question arises as to which is better if the task is to minimize the total cost of perimeter following. This question is answered by the following theorem.

Theorem 7: (The ANF-NM Incomparability Theorem): With respect to minimizing the cost of following a border, neither ANF nor NM transformations will always be within a constant factor of the best case.

Proof: First, we note that ANF can be arbitrarily worse than NM. Consider a sequence of quadtrees based on the template shown in Figure 3-7. Since the find-neighbors resulting from traversing the perimeter of a large square region are uniformly distributed, its contribution to the position calculated by the ANF algorithm is over-ridden by an attempt to optimize the short twisty part on the lower right hand side of the figure. The degree to which the ANF quadtree can be worse than the result of the NM quadtree on the same image is a function of the size of the square.

Alternatively, we could have a map involving a small two by two square and a very long twisty part that is always of width one. Thus, the NM algorithm optimizes the placement of the square at the expense of the cost of traversing twisty region. An example is shown in Figure 3-8.

Therefore, there are sequences of images for which either algorithm is arbitrarily bad.

Thus the practicality of these two transformations depends on what an average quadtree looks like in a particular application. The ANF transformation has an additional theoretical benefit, in that it allows a guarantee on the worst-case cost of certain operations as a function of the size of the quadtrees operated on.

As a final note on these two types of transformations, we might consider what set-theoretic operations preserve membership in the class of optimal quadtrees (without reapplication of the transformation). It is easy to show that both ANF and NM are preserved under image complementation, but not under image intersection or union.

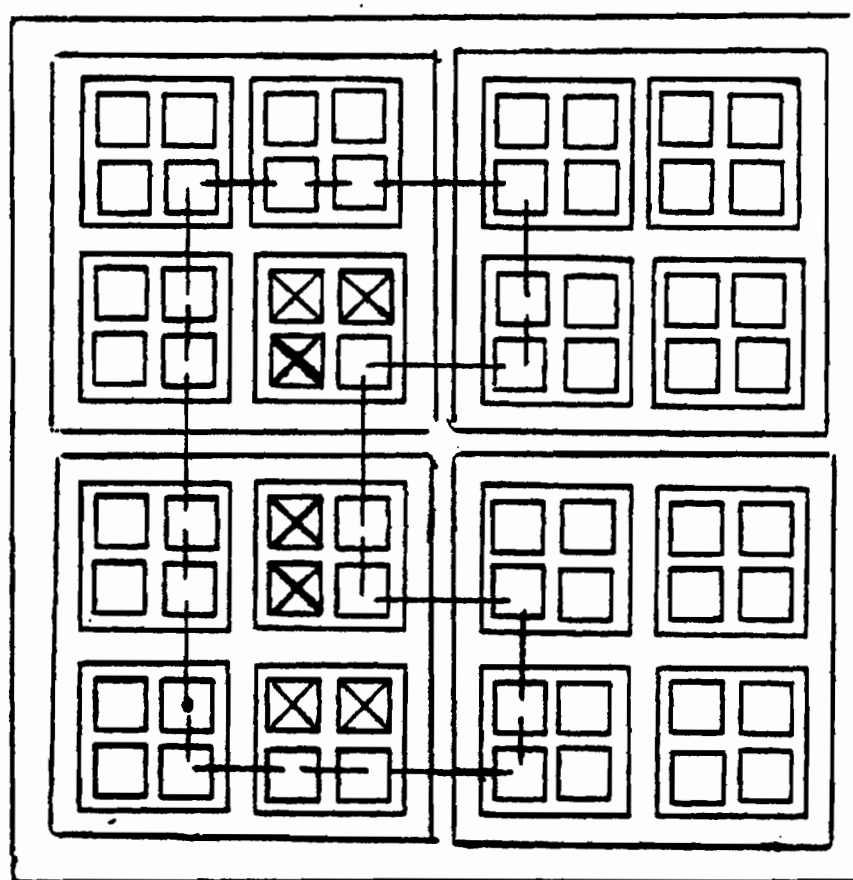


Figure 3-1: The region pyramid corresponding to the chain code for NNNNNEESSWSSESSWWWN, where f_0 is SW.SW.NE

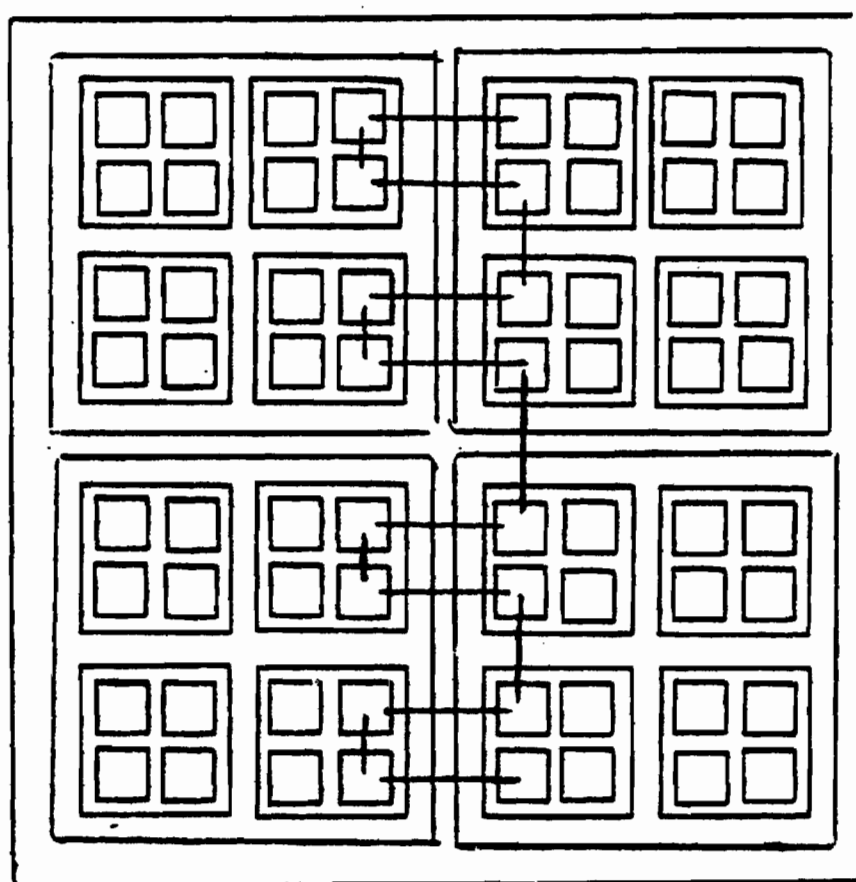


Figure 3-2: Example of the centered snake path in a pyramid of depth 3

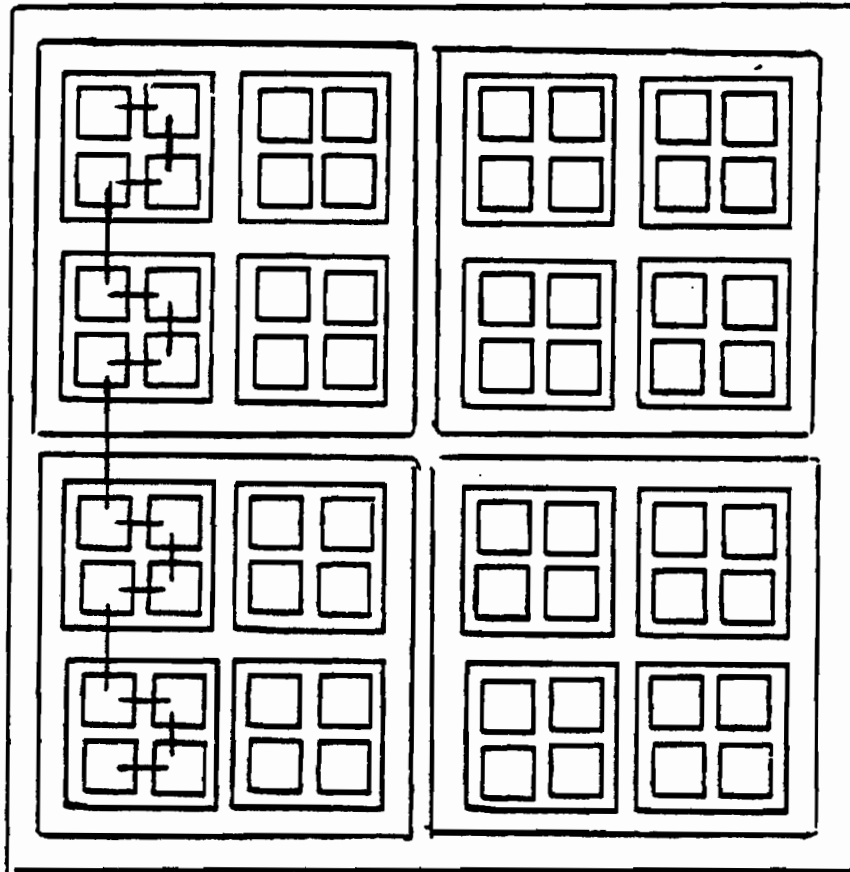


Figure 3-3: Example of the path in Figure 3-2 shifted to a more efficient position

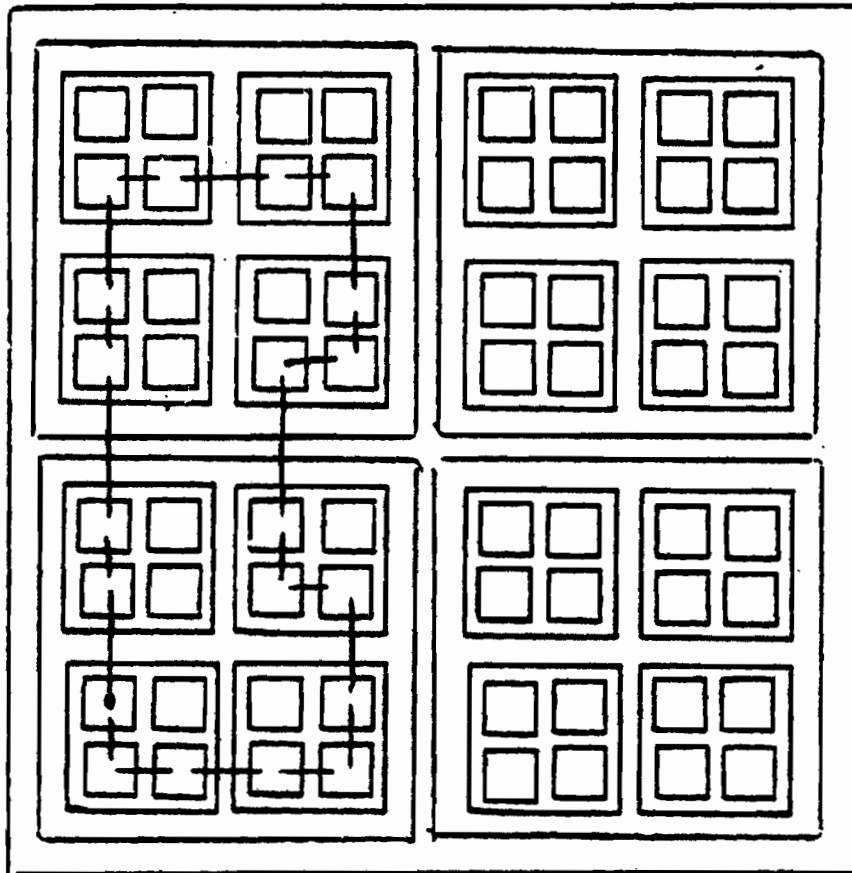


Figure 3-4: Result of ANF shifting of Figure 3-1

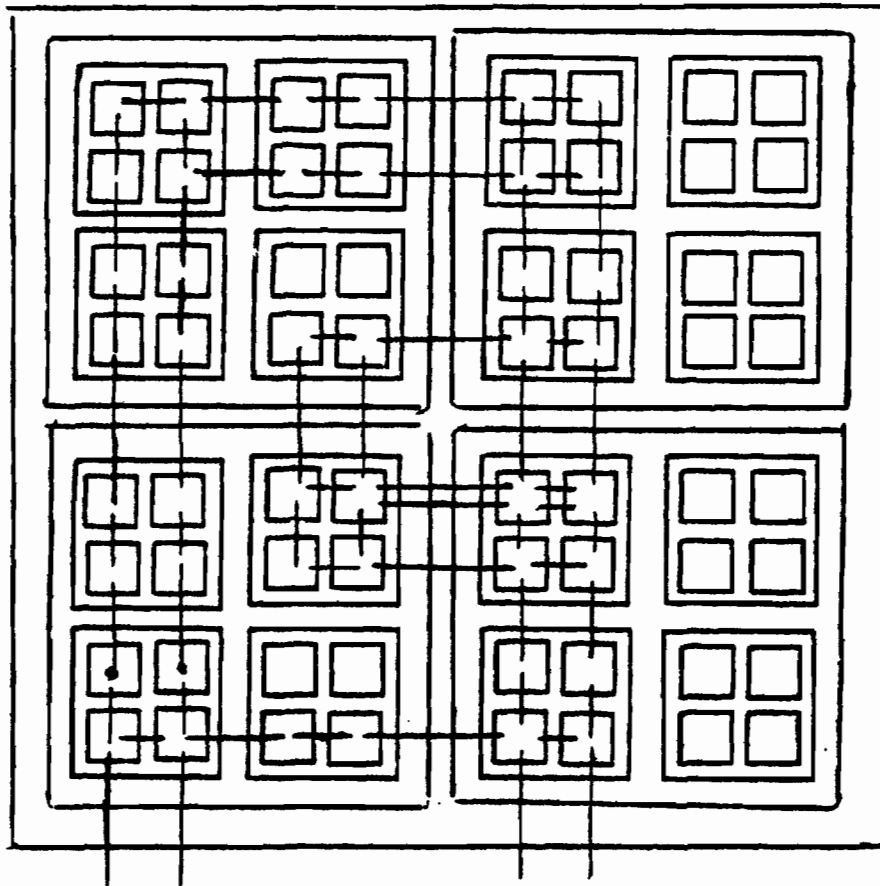


Figure 3-5: Actual chain code for perimeter following algorithm applied to image of Figure 3-1

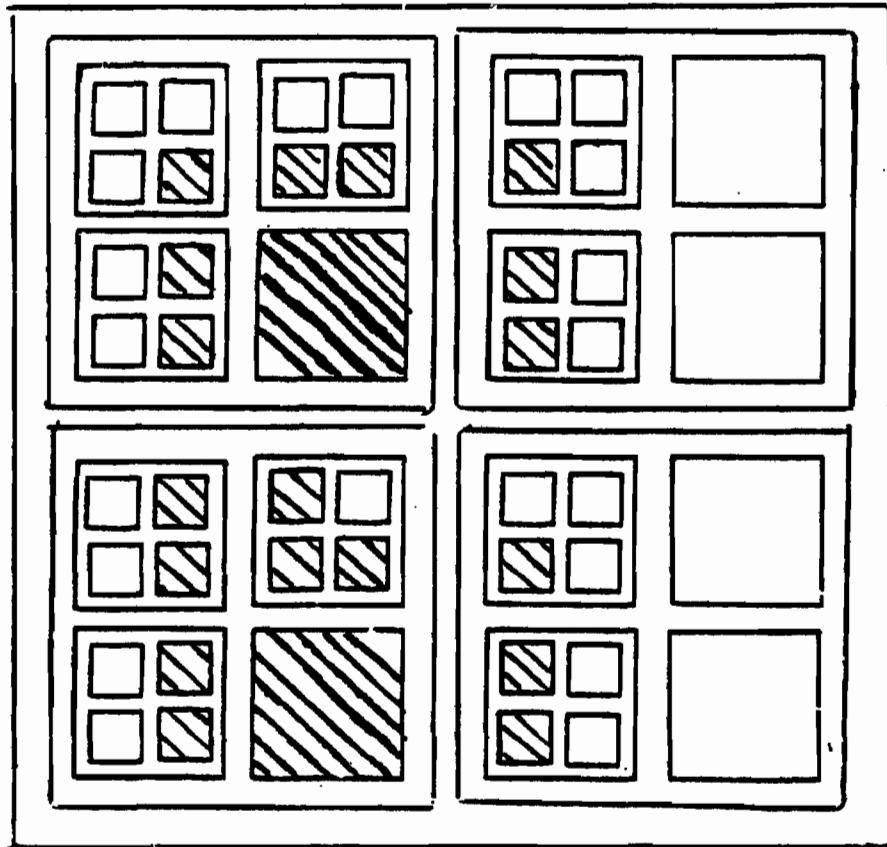


Figure 3-6: Quadtree corresponding to Figure 3-5

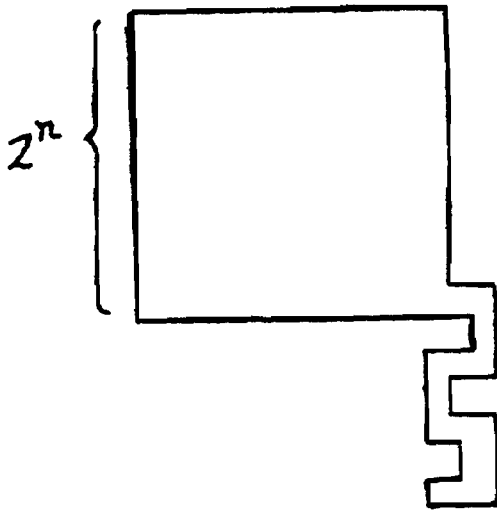


Figure 3-7: A sketch of the construction of a sequence where ANF creates too many superfluous nodes

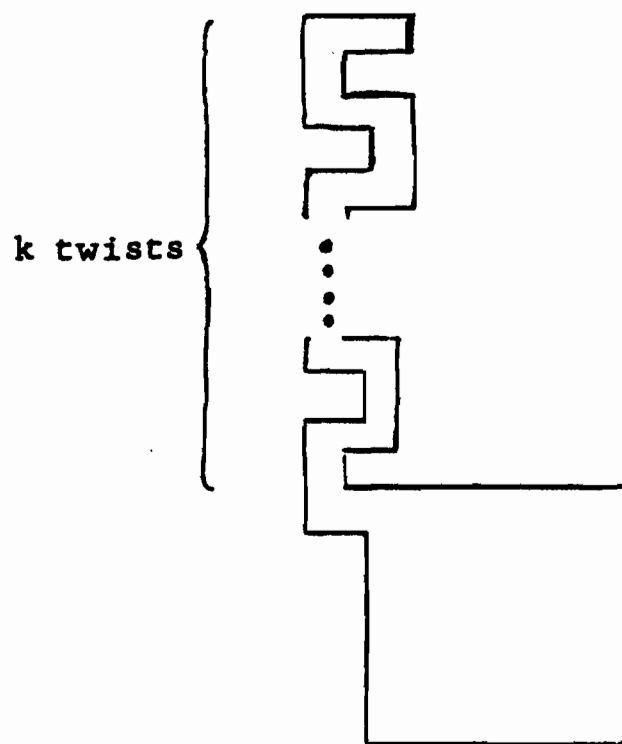


Figure 3-8: A sketch of the construction of a sequence where NM causes neighbor finding to be too laborious

CHAPTER 4

USING QUADTREES TO STORE POLYGONAL MAPS

In this chapter, we consider the problems inherent in storing polygonal maps. In particular, we discuss three ways of attempting to construct a quadtree-like data structure for storing polygonal maps that is more compact than the region quadtree.

In order to analyze the costs associated with storing a polygonal map, it is necessary to describe the correspondence between subframes and regions in a plane. As an added bonus for formalizing the connection between our Neighbor Theory and Euclidean Geometry, we demonstrate the relative consistency of Neighbor Theory in terms of Euclidean Geometry.

The first attempt to construct a compact quadtree representation is to build a region quadtree and, then perform common subtree elimination on the resulting structure. The other attempts abandon the notion of storing a region in the manner of a region quadtree and instead, look for significant collections of points that can be stored in a point or PR quadtree. These points must in some way correspond to the regions of the polygonal map.

The second attempt to construct a compact quadtree representation is motivated by the observation that the region quadtree that represents a Voronoi diagram is much larger than the set of points that generated the diagram. Since not every polygonal map is a Voronoi diagram, we instead view a polygonal map as a Voronoi diagram in which some of the regions have been merged. This places no restrictions on the permissible polygonal maps. Thus, our second attempt is to derive a set of points that generate a Voronoi diagram where labelling some regions identically produces our polygonal map. This transforms the problem of point in polygon search applied to a polygonal map into the problem of nearest neighbor search applied to a set of points. Nearest neighbor searches are efficient using PR quadtrees.

The third attempt is a more straightforward application of the PR quadtree to the polygonal map. Here, we store the vertices of the polygonal map in separate leaves. Then, we store the edges of the map by their order of intercept with respect to the leaves generated by the storage of vertices. This last attempt yields our most easily manipulable structure. However, our first attempt yields more interesting theoretical results. There are still too many open questions associated with the second attempt to judge its utility.

4.1. A GEOMETRIC MODEL FOR THE NEIGHBOR AXIOMS

Until now, we have refrained from restricting ourselves to any particular interpretation of the neighbor axioms presented in Section 2.1. This was possible because we were considering general properties of the neighbor axioms and not attempting to apply these properties to specific applications. However, in this chapter, we are concerned with representing polygonal maps, which is generally viewed as a task defined with respect to the standard Euclidean (analytic) geometry of a continuous plane. Thus, we must now assert a relation between the elements of a frame and the interior of a unit square. In the process of defining this relation, we also show that the neighbor axioms have a model in Euclidean geometry, and hence, that the neighbor axioms are consistent (if Euclidean geometry is consistent).

The natural interpretation of the quadtree partition of an Euclidean plane (as exemplified by the various figures herein) could be described informally as follows. The subframes correspond to square regions. Subframes of the same depth correspond to square regions of the same width. The squares corresponding to a subframe's children are enclosed by the square corresponding to their father. The width of the square representing a child of a subframe is half the width of the square representing the subframe. The neighbor relations among squares of the same size use the standard cartographic interpretation of north, south, etc.

To develop this correspondence for usage as a model of our Neighbor Theory Axioms, using $\langle x, y, w \rangle$ to denote a square of width w with a lower left hand corner with coordinates x and y , we can define the appropriate relations between Neighbor Theory and Analytic Geometry by the following. Note that Figure 4-1 illustrates the result of the application of these correspondences to F^3 .

1. Let $\text{subframe}(F^i, \lambda)$ correspond to the square $\langle 0,0,1 \rangle$.
2. If $\text{subframe}(F^i, \alpha)$ corresponds to the square $\langle x,y,w \rangle$, then
 - a. $\text{subframe}(F^i, \alpha.SW)$ corresponds to the square $\langle x,y,w/2 \rangle$;
 - b. $\text{subframe}(F^i, \alpha.NE)$ corresponds to the square $\langle x+(w/2), y+(w/2), w/2 \rangle$;
 - c. $\text{subframe}(F^i, \alpha.NW)$ corresponds to the square $\langle x+(w/2), y, w/2 \rangle$;
 - d. $\text{subframe}(F^i, \alpha.SE)$ corresponds to the square $\langle x, y+(w/2), w/2 \rangle$.
3. If A corresponds to $\langle x,y,w \rangle$, then
 - a. $A \text{ s } B$ implies B corresponds to $\langle x+w, y, w \rangle$;
 - b. $A \text{ n } B$ implies B corresponds to $\langle x-w, y, w \rangle$;
 - c. $A \text{ e } B$ implies B corresponds to $\langle x, y-w, w \rangle$;
 - d. $A \text{ w } B$ implies B corresponds to $\langle x, y+w, w \rangle$;
 - e. $A \text{ ne } B$ implies B corresponds to $\langle x-w, y-w, w \rangle$;
 - f. $A \text{ nw } B$ implies B corresponds to $\langle x-w, y+w, w \rangle$;
 - g. $A \text{ sw } B$ implies B corresponds to $\langle x+w, y+w, w \rangle$;
 - h. $A \text{ se } B$ implies B corresponds to $\langle x+w, y-w, w \rangle$.

In the following theorem, we show that this model satisfies the Neighbor Theory developed in Section 2.

Theorem 1: (The Cartesian Modelling Theorem): The natural interpretation of Neighbor Theory satisfies all the axioms of Neighbor Theory.

Proof: The proof of this theorem involves the individual verification of each of the axioms of Neighbor Theory with respect to the model stated above. Below, we will show how these verifications are done, by working out typical examples of the verification process for each of the three groups of axioms of Neighbor Theory.

The first axiom presented in Table 2-1 states that if A and B are frames, then $A \text{ n } B$ implies $B \text{ n } A$. Now, if A is a frame corresponding to $\langle x_1, y_1, w_1 \rangle$ and B is a frame corresponding to $\langle x_2, y_2, w_2 \rangle$, then $A \text{ n } B$ implies the following

$$\begin{aligned} w_2 &= w_1 = w \\ x_2 &= x_1 - w \\ y_2 &= y_1 \end{aligned}$$

which in turn implies that

$$w_1 = w_2 = w$$

$$x_1 = x_2 + w$$

$$y_1 = y_2$$

which is exactly what $B \supset A$ implies. The remaining axioms of Table 2-1 can be verified in an analogous manner.

The first axiom of Table 2-2 states that $A \wedge B$ and $B \supset C$ implies $A \supset C$. Assuming that A , B , and C refer to $\langle x_1, y_1, w_1 \rangle$, $\langle x_2, y_2, w_2 \rangle$, and $\langle x_3, y_3, w_3 \rangle$ respectively, then $A \wedge B$ implies

$$w_2 = w_1 = w$$

$$x_2 = x_1 - w$$

$$y_2 = y_1$$

and $B \supset C$ implies that

$$w_3 = w_2 = w$$

$$x_3 = x_2$$

$$y_3 = y_2 - w$$

which in turn implies that

$$w_3 = w_1 = w$$

$$x_3 = x_1 - w$$

$$y_3 = y_1 - w$$

which is exactly what $A \supset C$ implies, thus verifying this axiom. The other axioms of Table 2-2 can be verified analogously.

The third set of axioms contains axioms of two types. The first type is an expression of an identity, e.g.,

$$\text{subframe}(F^i, x.NW) \supset \text{subframe}(F^i, x.NE)$$

which works out perfectly once we have made the substitutions indicated by parts 2b and 2c of the natural model given above. The other type of axiom is exemplified by R4, i.e.,

$$\text{if } \text{subframe}(F^i, x) \supset \text{subframe}(F^i, y)$$

$$\text{then } \text{subframe}(F^i, x.NE) \supset \text{subframe}(F^i, y.NW)$$

This also yields quite readily to substitutions that define the natural model.

This concludes our overview of the proof for this theorem.

The natural interpretation of shifting (cf. Section 2.2) is that if P and P' are pictures with the same number of non-WHITE pixels, then there exists an n and m such that for every non-WHITE pixel in P , represented by $\langle x, y, w_1 \rangle$, the pixel $\langle x+m, y+n, w_2 \rangle$ in P' has the same color. This assumes that the width of the pixel in picture P' , denoted w_2 , is less than or equal to the width of the pixel in picture P ,

denoted w_i . Given that the notion of a picture is mapped into a set of ordered pairs $\langle n, c \rangle$ where n is the ordered triple defining the appropriate square and c is a color, substitutions analogous to those made to prove the above theorem result in a verification of this interpretation of the definition of the term shift.

4.2. USING COMMON SUBTREE ELIMINATION TO COMPACT QUADTREES

The first approach to the problem of storing polygonal maps is to try to take advantage of the regularity of digitizations of straight lines [24, 36] by performing a common subtree elimination procedure on the quadtree of the digitized form of the polygonal map. The result of this operation will henceforth be termed a CSE quadtree.

The conversion of a line quadtree to a CSE quadtree can be accomplished in a straightforward manner using the following $O(N \log_2 N)$ algorithm (where N is the number of nodes in the line quadtree).

1. Traverse the line quadtree and for each level i , insert nodes of that level into a queue Q_i .
2. Let d denote the maximum level for the input line quadtree and perform the following for each level i , from d to 1 (assuming that the root is at level 0 and that there is an ordering on colors and pointer values):
 - a. Sort the nodes x_j in Q_i lexicographically with respect to the vector (henceforth referred to as the descriptor of the node x_j)

$$\langle \text{son}[x_j, \text{NW}] . \text{son}[x_j, \text{NE}] . \text{son}[x_j, \text{SE}] . \text{son}[x_j, \text{SW}] . \text{value}[x_j] \rangle$$
 - b. For each pair of nodes with duplicate descriptors, remove one of them from the queue and change its father so that the father points to the other duplicate that was left in the queue.

The analysis of the above algorithm is as follows. Separating the nodes into queues is done in time proportional to the number of nodes, N . For each queue, we sort the queue (requiring time $M \log_2 M$ where M is the size of the queue) and then we remove duplicates in time proportional to the number of nodes in the queues. Since no node from the original quadtree appears in two queues and given the identity

$$(a + b) \log_2(a + b) \geq (a \log_2 a) + (b \log_2 b)$$
 the costs of the various parts of the algorithm sum to $O(N \log_2 N)$.

The best order of compression for a quadtree into a CSE quadtree that we can

expect is illustrated by the quadtree shown in Figure 4-2. It has a corresponding pointer representation shown in Figure 4-3. Now, we consider the result of the above algorithm as given in Figure 4-4. Here, we see that as we increase the resolution of the digitization of the same line, the number of nodes in the CSE quadtree grows linearly with the logarithm of the length (in pixels) of the line. This is a vast improvement over the normal quadtree where the number of nodes would grow linearly with the length of the line.

On the other hand, there are maps like the one shown in Figure 4-5 with pointer representations as illustrated in Figure 4-6 that have less spectacular results under the CSE transformation (see Figure 4-7). Looking at the reduction from Figure 4-6 to Figure 4-7, it is not obvious what formula would cover the series of which this is a member. So, the following compression results were deduced, not from extrapolation from examples, but rather as what resulted from applying a certain approach to calculating the number of nodes in a tree.

The general approach used in each of the proofs of the following theorems is one where we count the number of nodes used on each level of the quadtree and sum this over the number of levels. The formula for the number of nodes at a given level is the minimum of two values: 1) the maximum number of nodes at that level in a region quadtree that represents the same image, and 2) the maximum number of different subtrees rooted at that level. First, we consider the most general case, i.e., a map with no restrictions.

Theorem 2: (The CSE Arbitrary Map Compression Theorem): The size of the CSE quadtree for an arbitrary 2^n by 2^n binary map is $O(4^n/n)$.

Proof: Since there is no restriction on the arrangement of pixels, it is possible for each level i of the quadtree of a 2^n by 2^n picture to have 4^i nodes in it. Each node at level i represents a collection of 4^{n-i} pixels and each of these pixels can be either BLACK or WHITE. Thus, at level i , there are $2^{(4^{n-i})}$ possible different subtrees. Using the general approach outlined above, now, we want to calculate the point at which the dominating factor in the minimum of these two values changes (note that both of these are monotonic functions). Thus, we look for the smallest value of i such that

$$4^i > 2^{(4^{n-i})}$$

First, we take the logarithm of both sides and manipulate yielding the following:

$$2i > 4^{n-i}$$

$$2i > 4^n / 4^i$$

$$2i \cdot 4^i > 4^n$$

which holds when i equals $n - .5 \cdot \log_2(n)$. Thus, an upper bound on the sum:

$$\sum_{i=0}^n \min(4^i, 2^{i \cdot 4^{n-i}})$$

is given by the sum

$$\sum_{i=n}^{1+n-.5 \cdot \log_2(n)} 2^{i \cdot 4^{n-i}} + \sum_{i=0}^{n-.5 \cdot \log_2(n)} 4^i$$

We note that both of these sums are dominated by their last term. Hence they are bounded from above by a term proportional to

$$4^{n-.5 \cdot \log_2(n)}$$

which in turn is proportional to

$$4^n / n$$

thus completing this proof.

Next, we consider the result of applying a common restriction on the object stored in the structure, i.e., we assume that the object is convex. Note that the following proof and the one after that follow the same line of reasoning as the previous proof.

Theorem 3: (The Convex CSE Compression Theorem): The size of the CSE quadtree for a 2^n by 2^n map of a convex object is at most $O(2^n/n)$.

Proof: There are two properties of convex objects that make this proof work. First, is the fact that quartering a convex object yields four convex objects. Second, is the fact that the length of the perimeter of a convex object residing in a 2^i by 2^i square is

$$4 \cdot 2^i$$

From this plus the Quadtree Storage Size Theorem (Theorem 19) we can immediately note that the maximum number of nodes at a given level is proportional to $4 \cdot (2^i)$. We derive the maximum number of different subtrees that represent convex shapes

$$4^i \cdot 8^{i \cdot 2^{n-i}}$$

from the consideration that each picture built for a convex object could be built by a process that starts at one of 4^i given points and then moves in one of eight possible directions until it has completed the perimeter of the object.

When we calculate the upper bound on the analogous summation of

minimums of the two upper bounds and then evaluate it (as done in the previous theorem) we arrive at the result that the number of nodes in a CSE quadtree representing a convex object is $O(2^n/n)$ (assuming that n is the maximum depth of the quadtree).

Note that a corollary of the above theorem would be that straight lines could be represented in a CSE quadtree with the number of nodes proportional to the length of the line divided by the logarithm of the length of the line. This is a significant, though minor, improvement in the storage requirement of a CSE quadtree as opposed to a regular region or line quadtree. Of course, a straight line has more structure than a convex object. Thus, we might expect to get a better result than the one mentioned above. However, fixed portions of digitizations of lines with irrational slopes can exhibit rather varied behavior. The situation is considerably simpler when considering lines with slopes that are rational numbers, as is shown in the following theorem.

Theorem 4: (The Rational Slope CSE Compression Theorem): The size of the CSE quadtree for a 2^n by 2^n map representing a line with rational slope is at most $O(2^{n/2})$.

Proof: Let the rational value of the slope of the line be expressed as r/s where r and s are integers and r is between s and $-s$. Then, the chain code for this line will contain a subsequence of length s , such that the line is simply an infinite concatenation of this subsequence. The maximum number of nodes at a level i is 2^i (according to the Quadtree Storage Size Theorem of Section 2.3). The maximum number of different subtrees at level i is $s \cdot 2^{n-i}$, because there are 2^{n-i} different intercept points along the side of the node and there are s different parts of the line that could make the interception. Thus, we are presented with the following calculations:

$$\begin{aligned} 2^i &> s \cdot 2^{n-i} \\ i &> \log_2(s) + n - i \\ i &> (\log_2(s) + n)/2 \end{aligned}$$

which for fixed values of s yields an expected growth of the order of the square root of the length of the line.

We note that there have been no lower bound results corresponding to the above upper bound results. Thus, there is still the hope that even better results are possible. However, there are less complicated techniques for storing polygonal map that would result in more favorable compressions. Also, it is not clear how to manipulate CSE quadtrees in such a manner that the nodes do not constantly have to be resorted (as

they were in the original generation of the CSE quadtree from a region quadtree as described at the beginning of this section).

4.3. USING VORONOI DIAGRAMS AND PR QUADTREES TO STORE POLYGONAL MAPS

A second approach to using quadtrees to represent polygonal maps is to find a collection of points that characterize the regions of the maps. As mentioned in Section 1, there are two methods of storing point data using quadtrees. One is the point quadtree [7] and the other is the PR quadtree. In this section (as well as the next section) we will be using the PR quadtree.

The major reason for using the PR quadtree in an application is the ease of programming algorithms that manipulate it. However, the analysis of such algorithms is not always optimal because the number of nodes in a PR quadtree for a map of two points (we are not considering edges yet) is proportional to minus the logarithm of the distance between them. Such considerations indicate a favoring of the point quadtree where the number of nodes in the tree is exactly the number of points being stored. However, the access time for these points is also proportional to the number of nodes inserted in the point quadtree because there is no satisfactory dynamic balancing algorithm. Thus, the analysis we perform below favors working with PR quadtrees as we are concerned with the cost of accessing various parts of the map. The question still remains of what the points that reside in the PR quadtree characterize with respect to the original map.

One significant set of characterization points can be derived from the map by considering the map as the result of the merger of some of the regions in a Voronoi diagram. In this section, we shall discuss how to derive a set of Voronoi points for an arbitrary polygonal map.

The Voronoi diagram is one of the basic concepts of computational geometry [34]. Many basic problems in computational geometry can be reduced to problems of manipulating Voronoi diagrams. A Voronoi diagram is defined as a partitioning of a plane by a set of points (referred to herein as the Voronoi points) where a region in

the plane is defined with respect to one of the Voronoi points as being the locus of points in the plane that are closer to that Voronoi point than to any other Voronoi point. An example of a Voronoi diagram (shown as the boundaries between regions defined by Voronoi points) is given in Figure 4-8.

Algorithms for generating the Voronoi diagram (which is an example of a polygonal map) from the Voronoi points have been extensively studied [34]. The main motivation for this study has been to use the construction of Voronoi diagrams to transform questions about nearest neighbors into questions about points in polygons. Such a transformation is not necessary for solving nearest neighbor problems using quadrees, because nearest neighbors can be found by directly searching the PR quadtree. Thus, we are left with the possibility that we might be able to use the transformation in the other direction, i.e., transform point in polygon problems into nearest neighbor problems.

Unfortunately, there are maps that do not correspond exactly to Voronoi diagrams. Indeed, since it is easy to prove that every region in a Voronoi diagram must be convex, any map containing a concave region is sufficient as a counterexample. However, we might allow that a single region in the polygonal map might correspond exactly to a set of contiguous regions in the Voronoi diagram. Whether or not this is always possible is proven by giving a constructor of the appropriate Voronoi diagram for an arbitrary polygonal map.

Let us consider an algorithm that performs the following steps:

1. Calculate a value D that is $1/4$ th the minimum distance between a vertex of the polygonal map and an edge that does not coincide with that vertex. (Note: $1/4$ th is used to prevent any complications from arising in the last step of this algorithm. There are values closer to $1/2$ that would also work, but clearly using $1/2$ would still leave room for points being inserted to control one edge having the effect of interfering with another edge.)
2. Draw circles around the vertices of the polygonal map and place a point at the bisector of each of the arcs subtended by two adjacent line segments in the polygonal map. The result of this step is shown in Figure 4-9.
3. Again, calculate a value of D' that is $1/4$ th the minimum distance between the point in the augmented polygonal map and an edge that does not contain it.
4. For each side of each arc, place a point that is D' away from the line that

bounds the arc on that side. The result of this step is shown in Figure 4-10.

5. Finally, starting at each end of the line segment's intersection with an arc and moving toward the other such intersection, at intervals of D' place a point on each side of the line segment whose distance from the line segment is D' . The result of performing this last step is shown in Figure 4-11.

First, we note that this algorithm terminates in a rather straightforward manner. Its complexity is bounded from above by the number of points formed and the cost of calculating D and D' . For the size of the quadtree, the critical value to calculate is the number of points formed. If E is the number of edges and P is the length of the perimeter, then the number of points is

$$O(E+(P/D'))$$

Note that this algorithm is building sets of points that are potentially as large as the number of nodes in the region quadtree representation of the same map. Hence, we are not getting any guaranteed compression. However, this algorithm makes no attempt to remove unnecessary points; so, it is possible that there exist an alternative algorithm that terminates with better results than this algorithm. For example, this algorithm could never have derived the set of Voronoi points that defined Figure 4-8 from the diagram of the same figure.

On the other hand, recovering the original diagram from the collection of points produced by this algorithm is relatively easy, because each significant line segment is coated with points. For example, we note that it would be extremely messy to calculate the set of Voronoi points that correspond to overlaying the Voronoi diagrams for two arbitrary sets of Voronoi points. However, the same calculation for the sets of points produced by the second algorithm could be guided by the line coating pattern of points involved.

We might try to derive a Voronoi construction algorithm that makes a less conservative first estimate of the points needed for the Voronoi set and then tries to fix the portions of the diagram that are wrong. The correctness of such an algorithm is implicit in its derivation. However, there remains a question as to whether or not the construction defined by the algorithm will terminate after a finite number of steps.

One such algorithm is:

1. Determine the closest approach between a vertex and an edge not containing that vertex in the polygonal map. Call this distance D .
2. Draw a circle with radius $.5 \cdot D$ around each vertex of the polygonal map.
3. For each arc segment of the circles drawn in step 2 subtended by edges of the polygonal map, place a point on the bisector of the arc. These points on bisectors will be part of the ultimate Voronoi set.
4. For each point in the Voronoi set, follow the circle in both directions until a line segment of the polygonal map is reached. If an equivalent distance in the same direction can be moved without encountering a Voronoi point, then a point is inserted at that position. Add the points inserted in this phase to the set of Voronoi points.
5. Repeatedly consider each point in the set of Voronoi points. For each point, if it interferes with a line segment from the original map, then insert a point into the map that will cancel the effects of this point. This new point is immediately added to the set of Voronoi points.

Since step 5, in essence, says to keep modifying the set of Voronoi points as long as the original map is not a submap of the result, it is clear that if the algorithm halts, i.e., the corrections converge in a finite number of iterations, then the algorithm works. However, the question of whether or not this algorithm ever halts is far too complicated to be resolved here.

4.4. USING MAP VERTICES AND PR QUADTREES TO STORE POLYGONAL MAPS

4.4.1. BACKGROUND FOR PM QUADTREE DEVELOPMENT

The quadtree that we develop for storing polygonal maps will be referred to as the PM quadtree. It will be seen to be an adaptation of the PR quadtree. Our goal in designing this data structure is to derive a compact representation that satisfies the following three criteria:

1. It stores polygonal maps with absolute accuracy.
2. It is not overly sensitive to the positioning of the map (i.e., shift and rotation operations do not drastically increase the storage requirements of the map).
3. It is easy to manipulate.

To solve this problem, we will develop three closely related quadtree structures: PM_1 , PM_2 , and PM_3 .

In general, it is difficult to evaluate a data structure in a vacuum. We evaluate the PM quadtree in the context of the following three tasks: point-in-polygon determination, line insertion, and map overlay. We assume that the polygonal map is being manipulated in a dynamically changing environment. We also assume that associated with each line segment that forms the boundary of a region, there is a pair of names indicating which region is on which side of the line segment. This reduces the point-in-polygon task to the less restricted problem of locating a line segment that borders the region containing the query point.

Before adapting the PR quadtree, let us reconsider using one of the line quadtree schemes of [27]. Those schemes prove unsatisfactory for several reasons. First, line quadtrees store only a digitization of a map. Certain properties of polygonal maps cannot be directly represented (e.g., it is impossible for five line segments to meet at a vertex because the line quadtree is made up of a collection of square-like regions). Second, shifting or rotating a line quadtree leads to a possible loss of accuracy with respect to the map that was originally digitized. Third, line quadtrees may require much space. For example, for the polygonal map of Figure 4-12, we have the corresponding line quadtree in Figure 4-13. Note that although Figure 4-12 consists of just five vertices and six edges, its line quadtree requires 105 leaf nodes.

The quadtrees presented in this section will be defined in terms of their decomposition criteria. First, let us consider the definition of PR quadtrees in terms of their criterion for decomposing a quadrant. The decomposition criterion that defines the PR quadtree is denoted C1 and is given below:

- C1: At most one vertex can lie in a region represented by a quadtree leaf.

Figure 4-14 is the PR quadtree formed from the vertices of the map of Figure 4-12. We will use this example to illustrate the various issues involved with basing a map representation on PR quadtrees of map vertices.

The analysis of each of the PM quadtree variants relies heavily on the value of the worst-case tree depth, which is, in turn, a function of the input polygon. Thus, it

is worthwhile to determine this value for the PR quadtree. The worst-case PR quadtree depth is obtained as follows. Assume that the polygonal map is embedded in a unit square. As the depth of the PR quadtree increases, the maximum separation between two points in the same node is halved. The maximum separation between any two points in the unit square is $\sqrt{2}$. Points that are this far apart require a tree with depth 1 to separate them. Generalizing this observation, we see that when vv is the minimum separation between two distinct vertices, then an upper bound on the depth of the corresponding quadtree is

$$D1 = 1 + \log_2 (\sqrt{2} / vv)$$

In the PM quadtree structure, it is assumed that the region labels are associated with the edges. For example, the map of Figure 4-12 partitions the plane into three regions, labeled 1, 2, and 3. Thus, in Figure 4-14, edge AD is marked to indicate that region 2 lies to the right of AD and region 1 lies to the left of AD where right and left are with respect to a vantage point at the origin (i.e., the lower left corner of the enclosing unit square). Here, lines are viewed as if they have been extended to intersect the enclosing square. When a line segment is inserted in a map, the appropriate region labels are given with the line segment. Any changing of region labels of existing line segments is handled by deleting and then re-inserting the appropriate line segment. No effort is made to force the region labelings to be consistent.

In the subsequent discussion, we frequently need to refer to segments of edges of the polygonal map (we also use the term graph). We use the term q-edge to refer to a segment of an edge that spans an entire block (e.g., RS in Figure 4-14) or all of the block of which its corresponding edge is a member (e.g., ER in Figure 4-14). For example, edge EB consists of the q-edges ER, RS, ST, and TB.

4.4.2. THE PM₁ QUADTREE

A criterion analogous to C1, called C2, which takes edges into account is given below.

- C2: At most one q-edge can lie in a region represented by a quadtree leaf.

Unfortunately, C2 is inadequate because there exist polygonal maps that would require a

PM quadtree of infinite depth to satisfy C2. For example, consider vertex E and q-edges ER and EU in Figure 4-14. Assume that the x and y coordinates of E cannot be expressed (without error) as a rational number whose denominator is a power of two (e.g., let both coordinates be $1/3$). This means that E can never lie on the boundary between two quadrants. Thus, by virtue of the continuity of the q-edges, no matter how many times we subdivide the quadrant containing the vertex E and the q-edges ER and EU, there will always exist a pair of (possibly infinitesimally small) q-edges EH and EI which will occupy the same quadtree leaf. Vertices at subdivision points may, at times, avoid the infinite depth problem because of certain predefined conventions about which node they are in.

One solution to the above problem lies in replacing C2 with criteria C2' and C3 given below.

- C2': If a region contains a vertex, then it can contain no q-edge that does not include that vertex.
- C3: If a region contains no vertices, then it can contain at most one q-edge.

A quadtree built from the criteria C1, C2' and C3, representing the polygonal map of Figure 4-12, termed a PM_1 quadtree, is shown in Figure 4-15.

Since criterion C2' allows an arbitrary number of q-edges to be stored at one PM_1 quadtree leaf, the question of how these q-edges are organized arises. The simplest approach, consistent with our interest in worst-case tree-depth, is to store the q-edges in an AVL tree [1] where the q-edges are ordered by the angle that they form with a ray originating at the vertex and parallel to the positive x-axis. Since the number of q-edges passing through a leaf is bounded from above by the number of vertices belonging to the polygonal map, say V, the depth of the AVL tree is proportional to

$$A1 = \log_2 (V)$$

The depth of the PM_1 quadtree can be determined as the maximum of the depth required independently by each of the three criteria for building the quadtree. The factor contributed by criterion C1 has already been noted to be D1. If ev denotes the minimum separation between an edge and a vertex not on that edge (for a given

polygonal map), then by reasoning similar to the derivation of D1, the depth of the PM_1 quadtree required to fulfill criterion C2' is

$$D2' = 1 + \log_2 (\sqrt{2} / ev)$$

Analogously, if ss denotes the minimum separation between two non-intersecting q-edges (i.e., portions of edges bounded by either a vertex or the boundary of a PM_1 quadtree leaf of the PM_1 quadtree of the given polygonal map), then the PM_1 quadtree depth required to fulfill criterion C3 is

$$D3 = 1 + \log_2 (\sqrt{2} / ss)$$

The factors D1 and D2' are functions of the polygonal map and are independent of the positioning of the underlying digitization grid. However, the factor D3 is dependent on the positioning of the digitization grid and thus it can vary as the polygonal map is shifted. Recall that each of these factors, D1, D2', and D3, is an upper bound on some aspect of the quadtree's construction that could contribute to the depth of the resulting quadtree. The actual depth of the quadtree built could be considerably less than any of these factors. For maps of the complexity of the one shown in Figure 4-12, the D3 factor can become arbitrarily large. For example, suppose we shift the polygonal map in Figure 4-12 to the right. As vertex E (see Figure 4-14) gets closer to the eastern boundary of the quadrant containing it, the minimum separation between q-edges RS and UV (i.e., RU) gets smaller and smaller resulting in the growth of D3 to unacceptable values. While this is better than the impossibility associated with C2, it still behooves us to find a better decomposition criterion than C3.

4.4.3. THE PM_2 QUADTREE

In order to remedy the deficiency associated with criterion C3, it is necessary to determine when it dominates the cost of storing a polygonal map. In particular, D3 is greater than D2' only if ss is smaller than ev, which happens only when the two nearest non-intersecting q-edges are segments of edges that intersect at a vertex. For example, Figure 4-16 is the PM_1 quadtree for polygonal map ABCD where D3 is greater than D2', because ss (the distance between q-edges XY and WZ) is smaller than ev (the distance between C and BD). Note that XY is a q-edge of BD, WZ is a q-

edge of CD, and BD intersects CD at vertex D. This analysis leads us to replace criterion C3 with criterion C3' defined below.

- C3': If a region contains no vertices, then it can contain only q-edges that meet at a common vertex exterior to the region.

A quadtree built from criteria C1, C2', and C3', for the polygonal map of Figure 4-12, termed a PM_3 quadtree is shown in Figure 4-17.

The worst-case tree-depth is again proportional to the sum of the depth of the quadtree plus A1, the maximum depth of the AVL trees. However, the depth of the quadtree is bounded from above by the maximum of D1 and D2', the factors attributed to criteria C1 and C2' respectively. Note that by virtue of our definition of C3', the maximum depth resulting from its use is bounded from above by D2'.

As an example, consider Figure 4-18, which represents the same polygonal map as Figure 4-16 except that it uses C3' instead of C3. The analog of ss, termed ss', is defined as the minimum separation between two q-edges that are not segments of two intersecting edges. In this example, D3' is less than D2' because ss' (the distance between UB and SC) is greater than ev (the distance between C and DB). Note that the distance between QR and ST and the distance between RB and TC are irrelevant to D3', because, if necessary, these segments could be in the same leaf.

We have now achieved a structure for which the worst-case tree-depth is less sensitive to shift and rotation of the polygonal map. The only question that remains is whether we can do better. Can the contribution of criterion C2' be removed or reduced?

4.4.4. THE PM_3 QUADTREE

We consider a quadtree, termed a PM_3 quadtree, which is built using only criterion C1, but that could represent any polygonal map. For this version of the PM quadtree we revert to the original PR quadtree (i.e., Figure 4-14) except that more information is stored at each terminal node that corresponds to a region surrounding a vertex of this map. Since the depth factor D1 is always less than or equal to the maximum of the factors D1 and D2', the quadtree component (as opposed to the AVL

tree component) of the worst-case tree-depth is lower than in our previous structures. Indeed, the only time D_1 is greater than D_2' is when the polygonal map contains isolated edges (i.e., edges with both endpoints of degree 1). However, this structure does have the problem that the number of q -edges that can be stored in a leaf is now bounded by the number of q -edges in the graph, instead of the number of vertices. This does not affect the order of the worst-case tree-depth, because, in a planar graph, the number of edges is bounded from above by a linear function of the number of vertices (this is a corollary of Euler's formula [11]).

There still remains the problem of how to organize the q -edges in a leaf's region. We propose to partition the q -edges in a leaf's region into seven classes, each of which can be ordered by an AVL tree. Note that in any given leaf, some of these classes will often be empty.

The most obvious class of q -edges is the class of q -edges that meet at a vertex within the leaf's region. This class can be ordered in an angular manner as has been done previously. The remaining q -edges that pass through the leaf's region must enter at one side and leave via another. This yields six classes: NE, NS, NW, EW, SW, and SE, where NE denotes q -edges that intersect both the northern and the eastern boundaries of the leaf's region. Note that the q -edges are non-directional. For example, the q -edges in class NE (the other 5 classes are handled analogously) are ordered according to whether they lie to the left or to the right of each other when viewing them in an easterly direction from the northern boundary of the leaf's region. Q -edges that coincide with the border of a leaf's region are placed in either NS or EW as is appropriate. Note that any given leaf's boundary can only contain one such q -edge, because if it contained two, then it would have to contain two vertices and thereby violate C1.

4.4.5. ALGORITHMS FOR PM QUADTREES

Now that we have developed the PM quadtree, it is appropriate to examine how it can be used to achieve the three tasks that we specified in Section , i.e., point-in-polygon determination, line insertion, and map overlay. For each of the tasks, our discussion starts with the PM_1 quadtree, after which we show how the PM_2 and PM_3 quadtrees perform it. We first consider point-in-polygon determination.

4.4.6. POINT-IN-POLYGON DETERMINATION

For PM_1 quadtrees (built from $C1$, $C2'$, and $C3$) this problem has three cases which are illustrated by queries with respect to the points x , y , and z in Figure 4-15. The first case is illustrated by the point labeled x . In this case, the point lies in a leaf containing exactly one q -edge. Since region information is stored at each q -edge indicating the regions associated with the q -edge, this reduces to determining the side of the q -edge on which the point lies.

The second case is illustrated by the point labeled y . In this case, the query point lies in a leaf containing a vertex, C in this example. This situation reduces to finding a q -edge in the AVL tree that would neighbor a hypothetical q -edge from C and passing through y . Such a neighboring q -edge must border the region containing y . Thus, once again our task is reduced to determining on which side of a q -edge a point lies (i.e., y).

The third case is illustrated by the point labeled z . In this case, the query point lies in a leaf, say q , containing no q -edges. This means that all the points in the region represented by the leaf q lie in the same region of the polygonal map. It also means that one of q 's brothers must be the root of a subtree that contains a q -edge that borders the region containing z . In order to find this (not necessarily unique) brother, we move clockwise among the brothers of q . This prevents us from prematurely considering the diagonally neighboring brother of q (for an explanation, see the next paragraph). When considering a brother, one of two subcases arises. Either q 's clockwise neighboring brother, say r , (the first subcase) contains a q -edge, say b , lying on the boundary between q and r or (the second subcase) it doesn't. In the first subcase, the problem reduces to determining the side of q -edge b on which z lies. The second subcase is slightly more complex.

We postulate a hypothetical point z' in region r that is infinitesimally close to q 's region and recursively reapply the point-in-polygon procedure to z' . Figure 4-19 shows why we don't want to prematurely consider a diagonal brother. In this case placement of our hypothetical point z' in the SE brother of the quadrant containing z will lead us to conclude that z lies in region 2 rather than region 1 by virtue of its

relative position to edge segment ST which is the only edge segment in the quadrant. Note that point R is associated with the NE brother of the quadrant containing z by virtue of the conventions adopted in the introduction with respect to points that lie on quadrant lines emanating from subdivision points.

As an example of the case where z lies in a leaf containing no q -edges, consider Figure 7). Since the leaf containing z , call it q , is empty, we examine its clockwise brother, say r . Since r does not contain a q -edge on the boundary between q and r , the second subcase applies. Thus we postulate a point z' that is just across the boundary between q and r . Determining the polygon in which z' lies (in this example) is equivalent to determining the polygon in which x lies. Note that in second subcase, if r contains no q -edges, then the algorithm proceeds to examine r 's clockwise brother. It should be clear that one of the brothers must contain a q -edge as otherwise the brothers would have been merged to yield a larger node.

The worst-case execution time of point-in-polygon determination using a PM_1 quadtree constructed with criteria C1, C2', and C3 is proportional to the depth of the entire structure - i.e., the depth of the quadtree built from C1, C2', and C3 plus A1 (where A1 is the maximum depth of the AVL trees at the quadtree leaf nodes).

Replacement of C3 by C3', resulting in a PM_2 quadtree, does not lead to significant changes in the point-in-polygon determination procedure. The situation arising when q -edges are ordered about a point exterior to their region is handled in the same way as q -edges that are angularly ordered about their point of intersection. Of course, it is necessary to store with each AVL tree the point about which the ordering is being performed.

Point-in-polygon determination in PM_3 quadtrees is accomplished by finding a bordering q -edge with respect to each of the seven classes and then using the closest of the seven as the true bordering q -edge. The worst-case cost of point-in-polygon determination when using a PM_3 quadtree is proportional to D1 plus A1.

4.4.7. LINE SEGMENT INSERTION IN PM QUADTREES

Initially, we are given a PM_1 quadtree that satisfies the given criteria. To insert a line segment AB, we insert a q-edge of AB into each quadrant that AB intersects. In some of these quadrants, the insertion of a q-edge of AB would cause a violation of one of the criteria. In that case, the quadrant in question is subdivided and insertion is re-attempted.

The above subdivision of a quadrant can cause q-edges of line segments that had been previously inserted to be further subdivided. For example, consider Figure 4-20. First, we insert the line segment AB, which entails inserting the q-edges: AV, VW, and WB. We insert the line segment BC, which entails not only inserting the q-edges CZ and ZB, but also the q-edges WX, XY, and YB. Thus, the ultimate cost of inserting a line segment into a PM_1 quadtree is often paid for over many insertions as q-edges of the line segment are further subdivided to accommodate line segments that are being subsequently added.

In order to handle this situation for our worst-case analysis, we do not consider the total cost of inserting a particular line in a tree. Instead, we consider the ultimate cost of inserting that portion of the map that is currently built. This cost, henceforth known as the running-sum worst-case cost, assumes that the map is being built dynamically, i.e., that we don't know about future line segments at the time a line segment is initially inserted. Note that the running-sum worst-case cost (when summed over the insertions that built the map) is an upper bound on the actual cost of building the map so far. Implicit in the calculation of the running-sum cost at any instant during the building of a map is the assumption that we know the ultimate depth to which the tree will be expanded.

The running-sum worst-case map building cost is the product of the cost of inserting a q-edge and the number of q-edges that would have to be inserted. The cost of inserting a q-edge is the depth DMAX of the quadtree (the maximum of D1, D2', and D3) plus the depth of the AVL tree (A1). The calculation of an upper bound for the number of q-edges is slightly more complicated. We define L, the length of the perimeter of a polygonal map, to be the sum of the lengths of all the

line segments that form the map. In the following we show that the upper bound on the number of q -edges in the representation of the map is a function of L and the maximum depth, $DMAX$, of the quadtree structure.

Let us consider the structure of the q -edges that form a single line segment. First we note that for each line segment there are at most two q -edges that have the property of being incident with one of the vertices of the graph. Thus the number of such q -edges is proportional to the number of line segments in the graph. Since the factor $D1$ is both bounded above by $DMAX$ and requires that no line segment is less than 2^{-DMAX} units long, we deduce the following upper bound on the number of line segments, denoted S , in a map.

$$S \cdot 2^{-DMAX} \leq L$$

$$\text{or } S \leq L \cdot 2^{DMAX}$$

Of the remaining P q -edges in the map, all begin and end on the boundary of a square of size 2^{-DMAX} by 2^{-DMAX} . Of the P q -edges that begin and end on square boundaries, $\lfloor P/2 \rfloor$ can be grouped into disjoint pairs of contiguous q -edges. Each pair of contiguous q -edges forms a straight line that enters one square and exits the other. The length of such a line is bounded from below by the width of the squares that it passes through. This leads to the following upper bound on P :

$$\lfloor P/2 \rfloor \cdot 2^{-DMAX} \leq L$$

$$\text{or } P-1 \leq L \cdot 2 \cdot 2^{DMAX}$$

$$\text{or } P \leq L \cdot 3 \cdot 2^{DMAX}$$

To summarize, the number of q -edges is equal to the number of q -edges that are incident on a vertex plus the number of q -edges that are left over after the pairing process plus the number of q -edges that participate in the pairing process. For each of these values, we have an upper bound proportional to the length of the perimeter of the map times 2^{DMAX} . Thus the total number of q -edges is also bounded from above by the length of the perimeter of the map times 2^{DMAX} . Recall that the running-sum worst-case map building cost was proportional to the depth of the entire structure ($DMAX$ plus $A1$) times the number of q -edges inserted, for which we have just derived an upper bound. Thus, we have an upper bound on the cost of building a PM quadtree by inserting one line at a time.

Note that the analysis of line insertion for PM_2 quadtrees is the same as for PM_1 , except that DMAX now denotes the maximum of D1 and D2. Similarly, for the PM_3 quadtree, the analysis need only be modified in that DMAX now corresponds to D1.

The above costs for point-in-polygon determination and line insertion are not optimal under the assumption that these are the only two tasks we wish to perform. While it is true that others (e.g., [19]) have shown structures that have better worst-case analysis for the point-in-polygon and line-segment insertion tasks than the PM quadtree described above, the PM quadtree has a major advantage over other structures in that it organizes the data without a directional bias. By directional bias, we mean that the cost of moving from one node to its neighbor is independent of the direction in which the neighbor lies. Thus PM quadtrees are generally better for range queries, whose analysis is a natural extension of the above discussion of point-in-polygon determination for PM quadtrees.

While the lack of bias in its treatment of the x and y coordinates is a major help to quadtree based structures in the performance of range query tasks, the regular decomposition quadtree schemes also favor the performance of set operations (e.g., union and intersection). Thus we consider below the task of map overlay which corresponds to calculating the cross product of a map, i.e., the regions in the resulting map have a natural labeling in terms of ordered pairs of the labels of the two overlaid maps.

4.4.8. OVERLAY ALGORITHM FOR PM QUADTREES

We first consider the computation of overlay for PM_3 quadtrees. The overlay algorithm can be decomposed into four procedures: OVERLAY, MERGE, CAN_MERGE, and QUARTER. Procedure OVERLAY takes two PM_3 quadtrees as parameters. It traverses the two quadtrees in parallel. When one tree is a leaf and the other tree is not, the leaf is split into a node with four sons, each of which are leaf nodes (and correspond to a description of the same region as the original leaf) and the OVERLAY procedure is applied recursively to the corresponding sons. When both quadtrees are leaf nodes, the information about line segments in each of them is merged to form a leaf in the output tree.

```

procedure OVERLAY(SUBTREE1, SUBTREE2);
/* Compute the overlay of the quadtrees SUBTREE1 and
   SUBTREE2. */
begin
  value quadtree SUBTREE1, SUBTREE2;
  quadtree QUARTERED, THE_SUBTREE, TREE_TO_RETURN;
  quadrant X;
  if IS_LEAF(SUBTREE1) and IS_LEAF(SUBTREE2) then
    return (MERGE(SUBTREE1, SUBTREE2))
  else if IS_LEAF(SUBTREE1) or IS_LEAF(SUBTREE2) then
    begin
      QUARTERED := QUARTER(WHICHEVER_WAS_LEAF(SUBTREE1
                                                , SUBTREE2));
      THE_SUBTREE := WHICHEVER_WAS_NOT_LEAF(SUBTREE1,
                                             SUBTREE2);
      TREE_TO_RETURN := NEW_NODE();
      foreach X do
        SON(TREE_TO_RETURN, X) := OVERLAY(
                                   SON(QUARTERED, X),
                                   SON(THE_SUBTREE, X));
      return (TREE_TO_RETURN);
    end
  else begin
    TREE_TO_RETURN := NEW_NODE();
    foreach X do
      SON(TREE_TO_RETURN, X) := OVERLAY(
                                   SON(SUBTREE1, X),
                                   SON(SUBTREE2, X));
    return (TREE_TO_RETURN);
  end;
end;

```

Procedure MERGE produces the subtree that results from merging two leaf nodes (from a pair of PM_3 quadtrees) according to whether or not the q -edges involved intersect. Recall that the information about line segments that is stored in the leaf nodes is ordered with respect to various intercepts (either a vertex or a side of the node). Thus the merger of this information is simply the merger of the corresponding trees. The routine that performs the actual merging is termed AVL_MERGE and is not given here. The worst case execution time of MERGE is proportional to the number of nodes merged plus the cost of executing the procedures: CAN_MERGE and QUARTER.

The coding of the procedure MERGE uses WHICHEVER_HAD_AVL_VERTEX, which returns NULL if neither leaf contains a distinguished vertex and otherwise returns the AVL tree connected to the distinguished vertex. Note that the function CAN_MERGE has a side effect of removing redundant references to the same vertex

(i.e., with same x and y coordinates). Two the information in two AVL trees is merged to form a new AVL tree by the function AVL_MERGE.

```

procedure MERGE(LEAF1, LEAF2);
/* Perform the overlay algorithm on the simple case
   where both quadtrees, LEAF1 and LEAF2, are leaf
   nodes. */
begin
  value quadtree LEAF1, LEAF2;
  quadtree LEAF_TO_RETURN, QUARTER1, QUARTER2;
  quadrant Q;
  side X, Y;
  if not CAN_MERGE(LEAF1, LEAF2) then
    begin
      LEAF_TO_RETURN := NEW_NODE();
      QUARTER1 := QUARTER(LEAF1);
      QUARTER2 := QUARTER(LEAF2);
      foreach Q do
        SON(LEAF_TO_RETURN, X) := MERGE(SON(QUARTER1, Q),
                                          SON(QUARTER2, Q));
      return(LEAF_TO_RETURN)
    end
  else begin
    LEAF_TO_RETURN := NEW_NODE();
    AVL_VERTEX(LEAF_TO_RETURN) :=
      WHICHEVER_HAD_AVL_VERTEX(LEAF1, LEAF2);
    foreach X do
      foreach Y do
        AVL_SIDE(LEAF_TO_RETURN, X) :=
          AVL_MERGE(AVL_SIDE(LEAF1, X),
                    AVL_SIDE(LEAF2, Y));
      return (LEAF_TO_RETURN);
    end;
  end;
end;

```

Procedure CAN_MERGE determines whether a pair of leaf nodes of PM₃ quadtrees can be merged. In order to be mergible, the q-edges in the two leaf nodes cannot intersect and if there is a vertex in both of the leaf nodes then, it must have the same x and y coordinate values. Since the checking of intersection (done by the procedure LINES_INTERSECT) can take advantage of the ordering of the q-edges, the execution time of CAN_MERGE is proportional to the number of q-edges in its leaf parameters.

```

Boolean procedure CAN_MERGE(LEAF1, LEAF2);
/* Returns TRUE if and only if the merger of the leaf
   nodes, LEAF1 and LEAF2, would not create any new
   vertices. Note that in the case where neither leaf
   node contains a vertex, it is possible for one
   intersection to occur and yet the nodes would
   still be mergible. The counter, N, records the
   number of known vertices in the pair of nodes. If
   this counter is zero, then LINES_INTERSECT, upon
   noticing that exactly one intersection occurs, has
   the side effect of incrementing N and updating the
   AVL_VERTEX field of its last parameter, which is
   always LEAF1. Of course, if more than one
   intersection occurs, then LINES_INTERSECT will
   cause CAN_MERGE to return FALSE.
*/
begin
  reference quadtree LEAF1, LEAF2;
  side X;
  integer N;
  N := 0;
  if HAS_VERTEX(LEAF1) and HAS_VERTEX(LEAF2) then
    if SAME_XY_VERTEX(LEAF1, LEAF2) then
      begin
        AVL_VERTEX(LEAF1) := AVL_MERGE(
                                AVL_VERTEX(LEAF1),
                                AVL_VERTEX(LEAF2));
        AVL_VERTEX(LEAF2) := NULL;
      end
    else return (FALSE);
  if HAS_VERTEX(LEAF1) or HAS_VERTEX(LEAF2)
  then begin
    N := 1;
    THE_VERTEX := WHICHEVER_HAD_VERTEX(LEAF1,
                                         LEAF2);
    foreach X do
      if LINES_INTERSECT(THE_VERTEX,
                          AVL_SIDE(LEAF1, X), N, LEAF1)
      or LINES_INTERSECT(THE_VERTEX,
                          AVL_SIDE(LEAF2, X), N, LEAF1)
      then return(FALSE)
    end
    foreach X do
      if LINES_INTERSECT(AVL_SIDE(LEAF1, X),
                          AVL_SIDE(LEAF2, X), N, LEAF1)
      then return (FALSE);
    return (TRUE);
  end;
end;

```

The final procedure to consider is QUARTER, which takes a leaf as a parameter and returns a subtree containing four leaves that represents the same map. This

procedure involves visiting each q -edge in its leaf parameter and determining which parts of it will lie in which sons of the new subtree. Its execution time is proportional to the number of q -edges processed. We don't give its code here.

We now consider an analysis of the OVERLAY algorithm. Let N be the number of q -edges in the PM quadtree built by OVERLAY. Recall that D_{MAX} is an upper bound on the depth of the PM quadtree (not including the depth of the AVL trees). Although OVERLAY performs a preorder traversal of two subtrees in parallel, its calculation could be performed by a breadth-first traversal of the two trees. This reformulation is used in the following analysis.

First, we note that the cost of executing OVERLAY is proportional to the number of nodes in the input quadtrees (as is the case for the region quadtree intersection and union algorithms) except that the cost of the CAN_MERGE and QUARTER procedures is unbounded (whereas the analogous procedures for a region quadtree are bounded). Instead, we find that the cost of performing these functions is proportional to the number of q -edges in the two leaf parameters of these functions. We now consider the worst case of the execution time of the OVERLAY algorithm. This occurs when we overlay a pair of PM quadtrees having two corresponding leaf nodes at level k containing vertices which are arbitrarily close to each other. Alternatively, this worst case also results when two new intersection points are created that are arbitrarily close to each other. With respect to the analysis, the significance of two vertices being arbitrarily close to each other is that D_{MAX} becomes arbitrarily larger than k .

Let i be a level between k and D_{MAX} . The execution of OVERLAY on each leaf at level i will result in an invocation of CAN_MERGE and QUARTER (which creates the leaf nodes of level $i+1$). Hence the cost associated with the processing at level i is proportional to the number of q -edges at level i . This cost must be paid at each of the levels between k and D_{MAX} . The number of such levels is bounded from above by D_{MAX} . The number of q -edges at any given level is bounded from above by the number of q -edges in the final result, i.e., N . Thus, it follows that the entire OVERLAY algorithm will execute in time proportional to $N \cdot D_{MAX}$.

Note that the above analysis holds for PM_1 , PM_2 , and PM_3 quadtrees. At first

glance, it might appear that the OVERLAY algorithm could be done just as effectively by repeatedly performing line insertions from one of the PM quadtrees into the other. However, the analysis for such an approach turns out to be of order $N \cdot (D_{MAX} + A_1)$. Our OVERLAY procedure does better than this because the q -edges occurring within a given AVL tree are processed sequentially instead of randomly.

4.4.9. CONCLUSIONS

We have taken an iterative approach to developing a quadtree-like data structure for storing polygonal maps. We started with the PR quadtree and developed the PM quadrees. The final formulation, PM_3 , uses the same decomposition rule as the PR quadtree but stores considerably more information in the terminal nodes. It should be clear that the PM quadtree enables storing polygonal maps with absolute accuracy and that its worst-case tree-depth is less sensitive to the positioning of the polygonal map. Thus unlike the region quadtree and the line quadrees of [Same82], the PM_2 and PM_3 quadrees can be shifted or rotated without distortion or unreasonable change in the storage requirements of the structure. Nevertheless, the storage requirements are still somewhat dependent on the positioning of the space within which the map is embedded. For example, the polygonal map of Figure 4-21a requires 7 PR quadtree leaf nodes. However, if we shift the map slightly, we get Figure 4-21b, which requires only 4 PR quadtree leaf nodes. We also observe that our proposed quadrees are relatively compact. As a comparison, we note that the line quadtree in Figure 4-13 required 105 quadtree leaf nodes, whereas the PM_2 quadtree of Figure 4-17 required 13 quadtree leaf nodes and 21 AVL data nodes (scattered among 11 AVL trees), and the PM_3 quadtree of Figure 4-14 required 7 quadtree leaf nodes and 17 AVL data nodes (scattered among 9 AVL trees). Note that many of the AVL trees consist of single data nodes.

In addition, we have shown that point-in-polygon determination using the PM quadtree can be performed in time proportional to the depth of the structure. We also gave an upper bound on the worst-case cost of insertion of a portion of a map dynamically. Finally, we have shown how to overlay two polygonal maps that are represented by PM quadrees. Some possible future work includes the development and analysis of algorithms for other operations, e.g., shift and rotation.

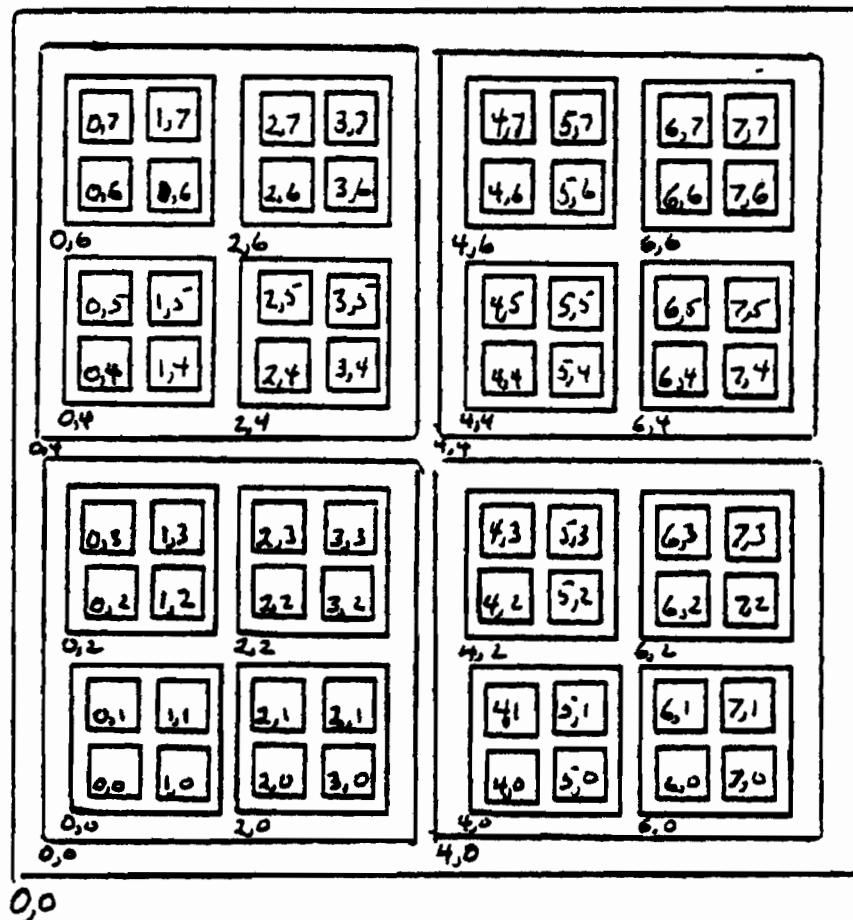


Figure 4-1: Subframes of F^3 labeled by Cartesian coordinates of lower left hand corner

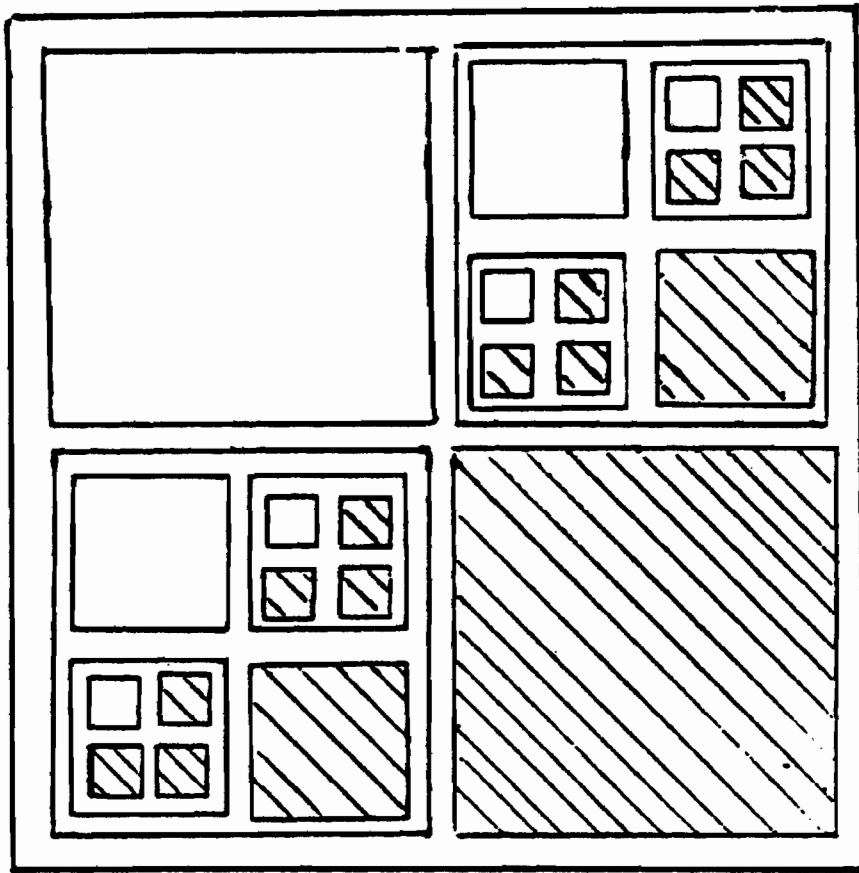


Figure 4-2: A quadtree representing two regions separated by a 45 degree line

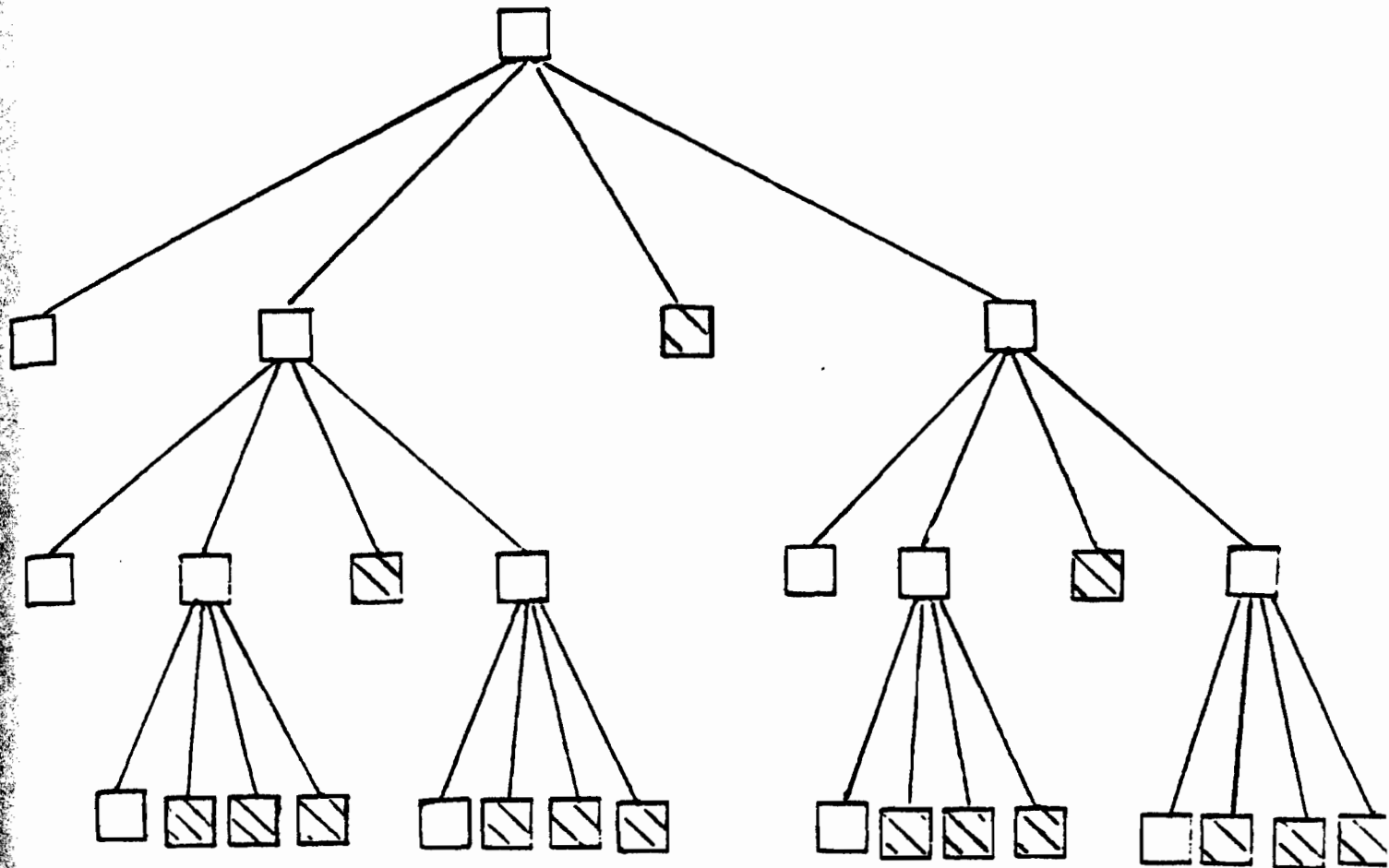


Figure 4-3: Pointer representation of Figure 4-2

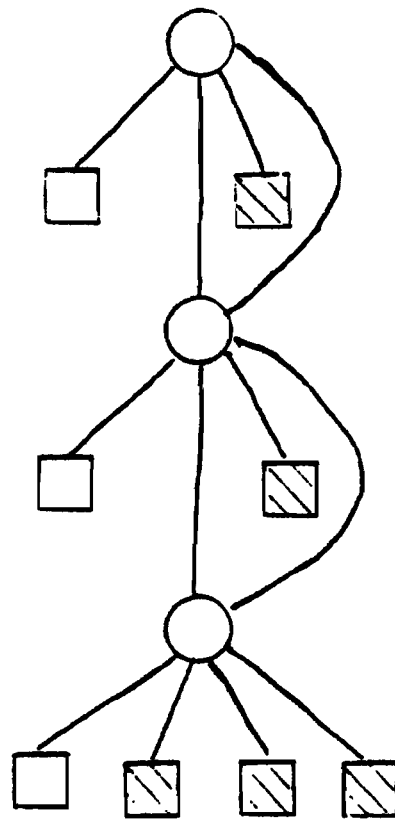


Figure 4-4: Result of common subtree elimination on
Figure 4-3

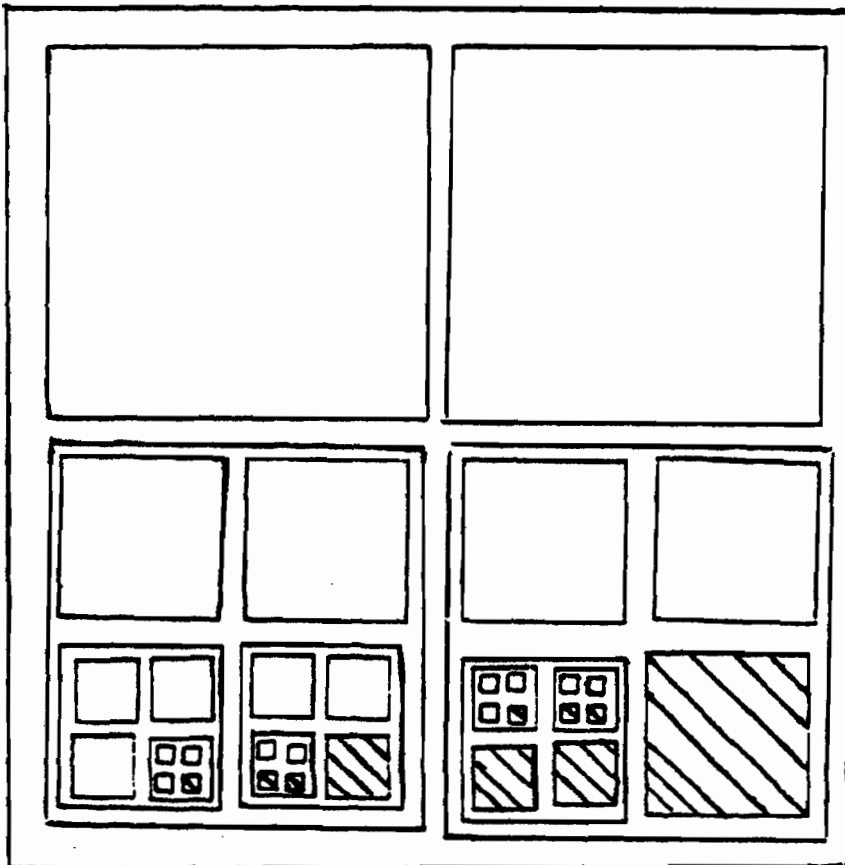


Figure 4-5: A quadtree representing two regions separated by a line with slope approximately $4/16$ ths

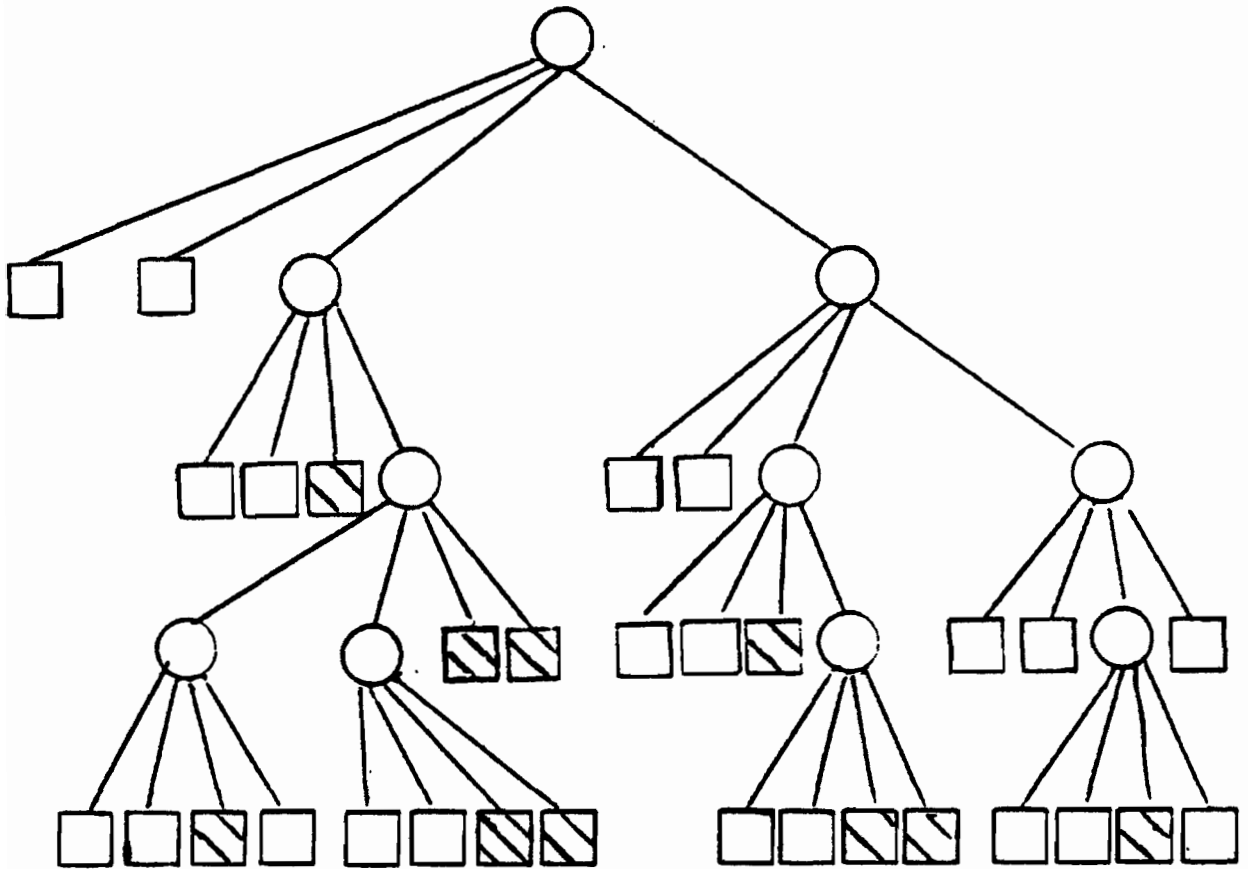


Figure 4-6: Pointer representation of Figure 4-5

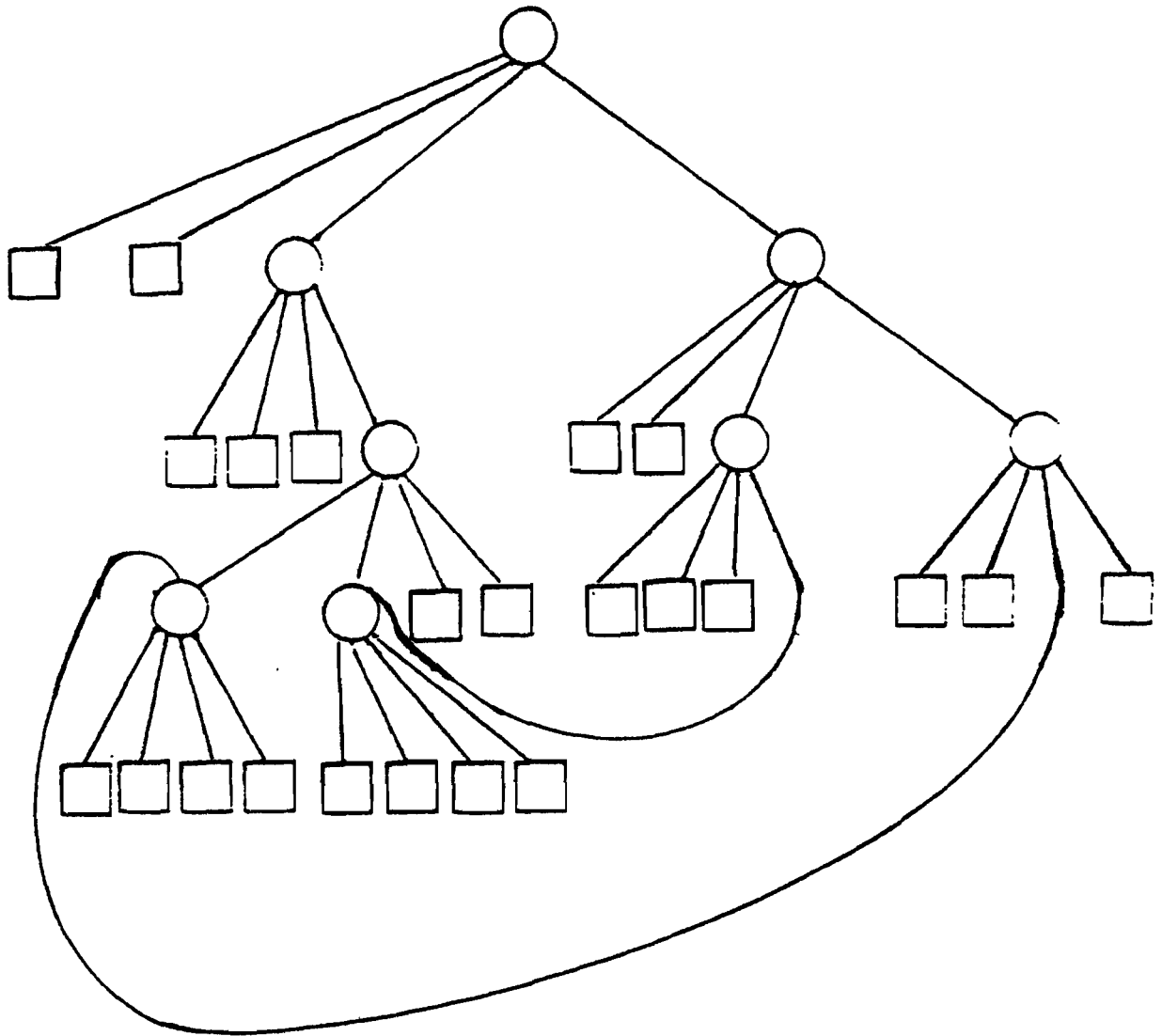


Figure 4-7: Result of common subtree elimination on Figure 4-6

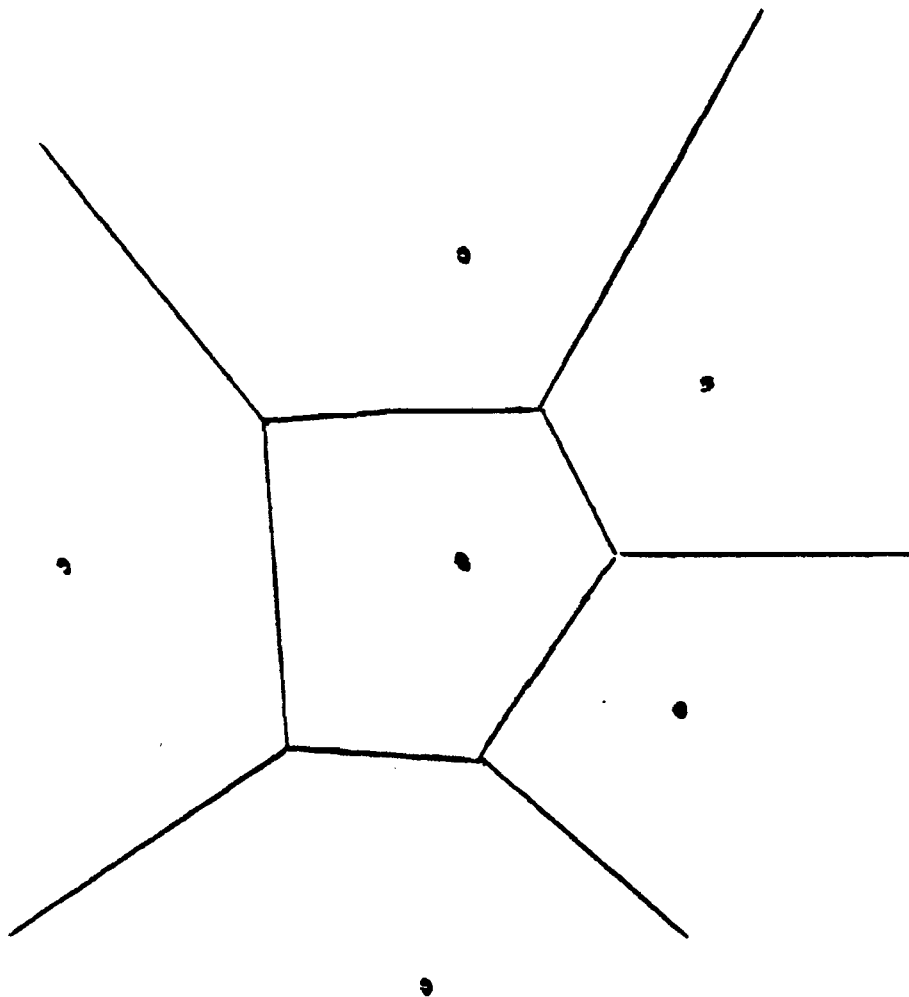


Figure 4-8: A Sample Voronoi Diagram

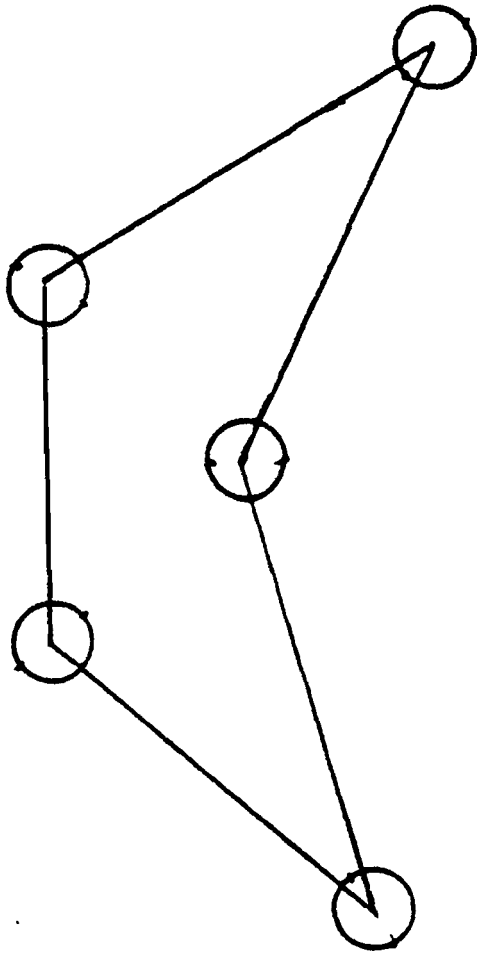


Figure 4-9: First Step in Finding Voronoi Points for Map

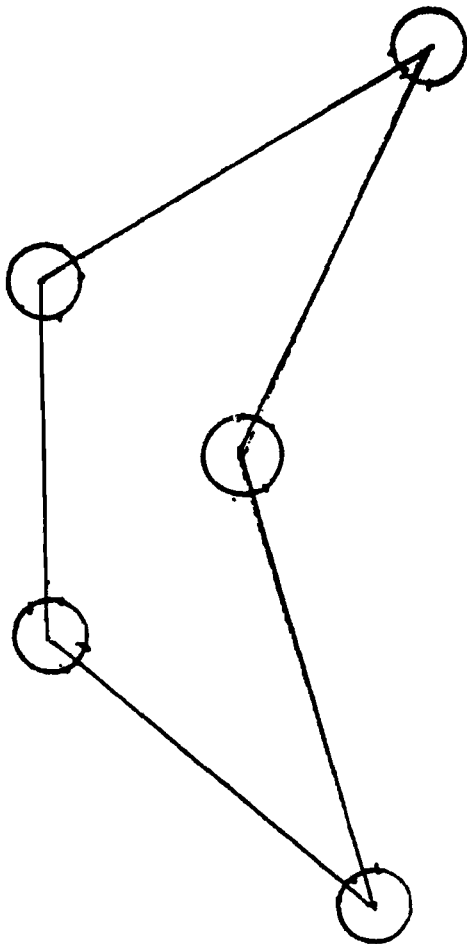


Figure 4-10: Second Step in Finding Voronoi Points for Map

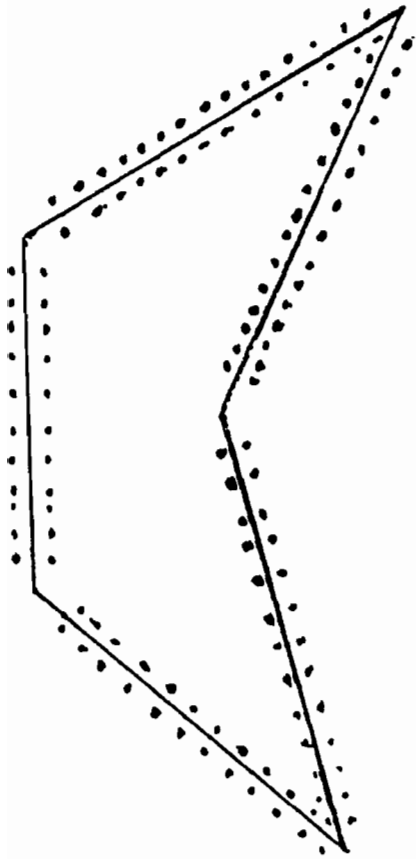


Figure 4-11: Final Step in Finding Voronoi Points for Map

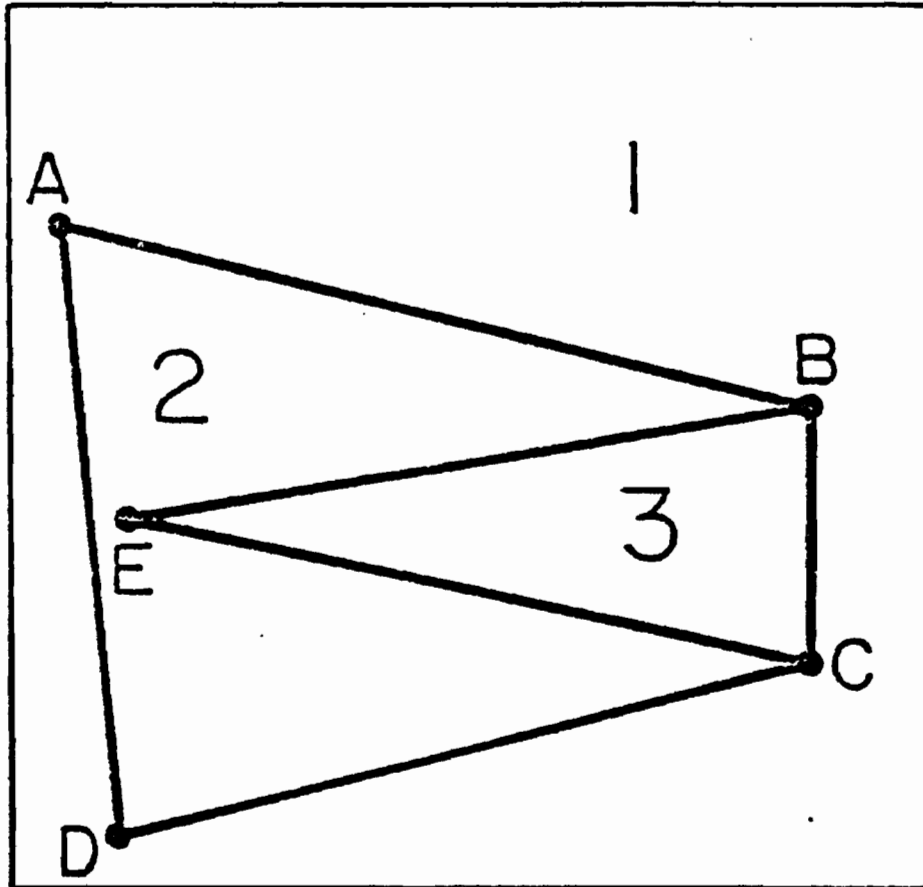


Figure 4-12: Sample polygonal map

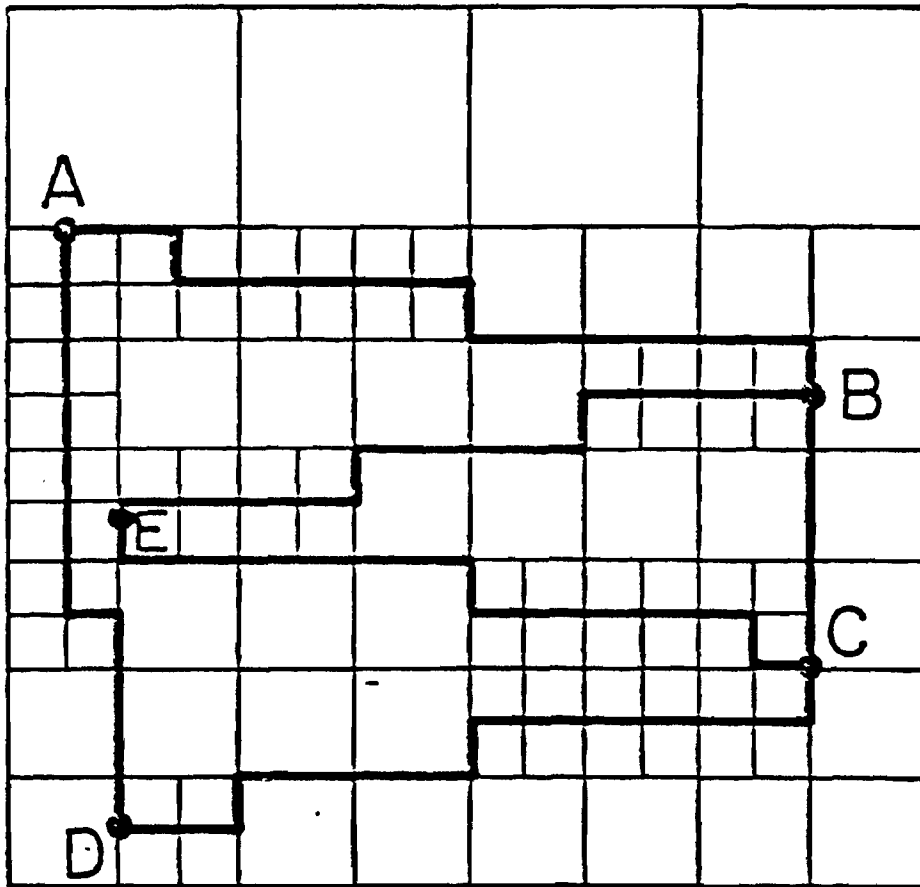


Figure 4-13: Line quadtree corresponding to Figure 4-12

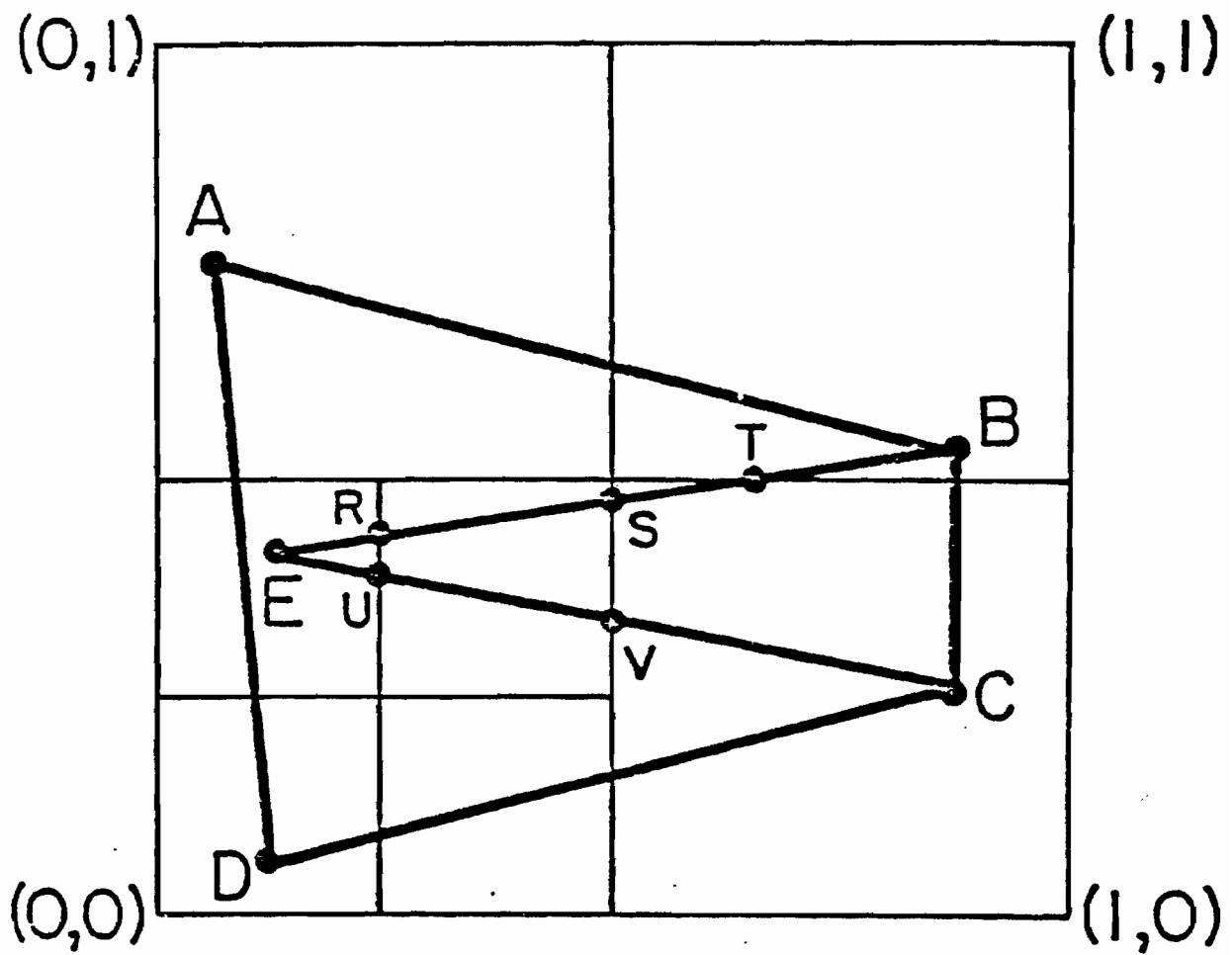


Figure 4-14: PR quadtree corresponding to Figure 4-12 when the line segments of the map are ignored. When the line segments are included, the PM_3 quadtree corresponding to Figure 4-12

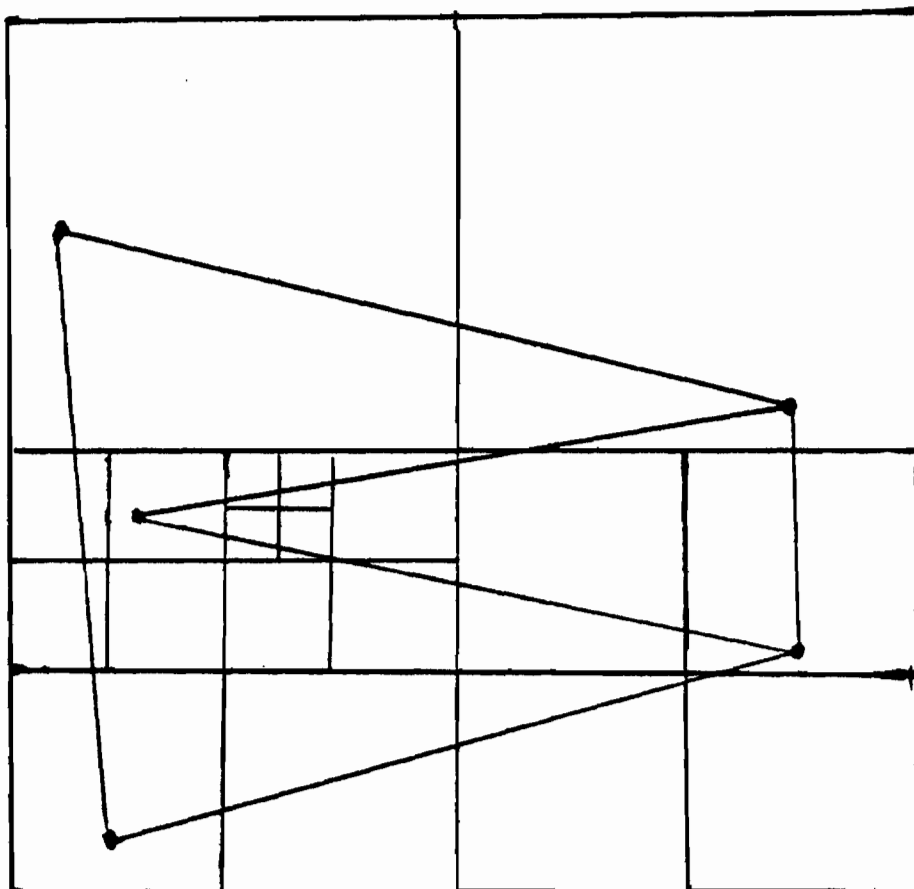


Figure 4-15: The PM₁ quadtree corresponding to Figure 4-12

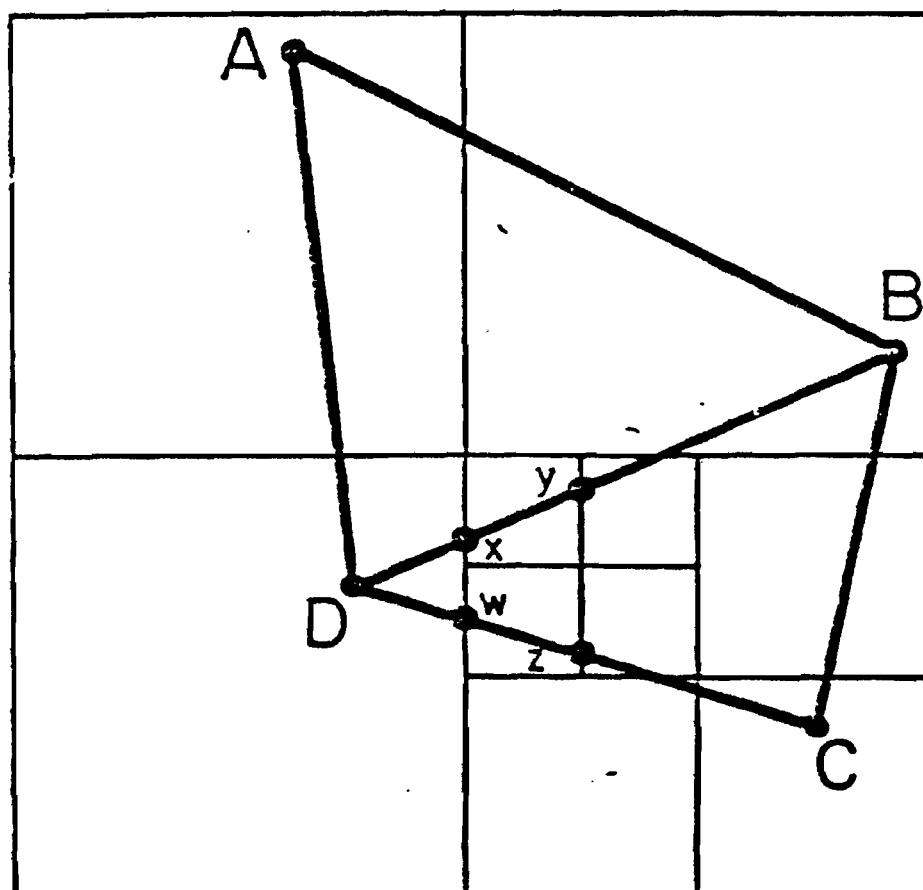


Figure 4-16: Example illustrating $D_3 > D_2'$ when C_3 is used

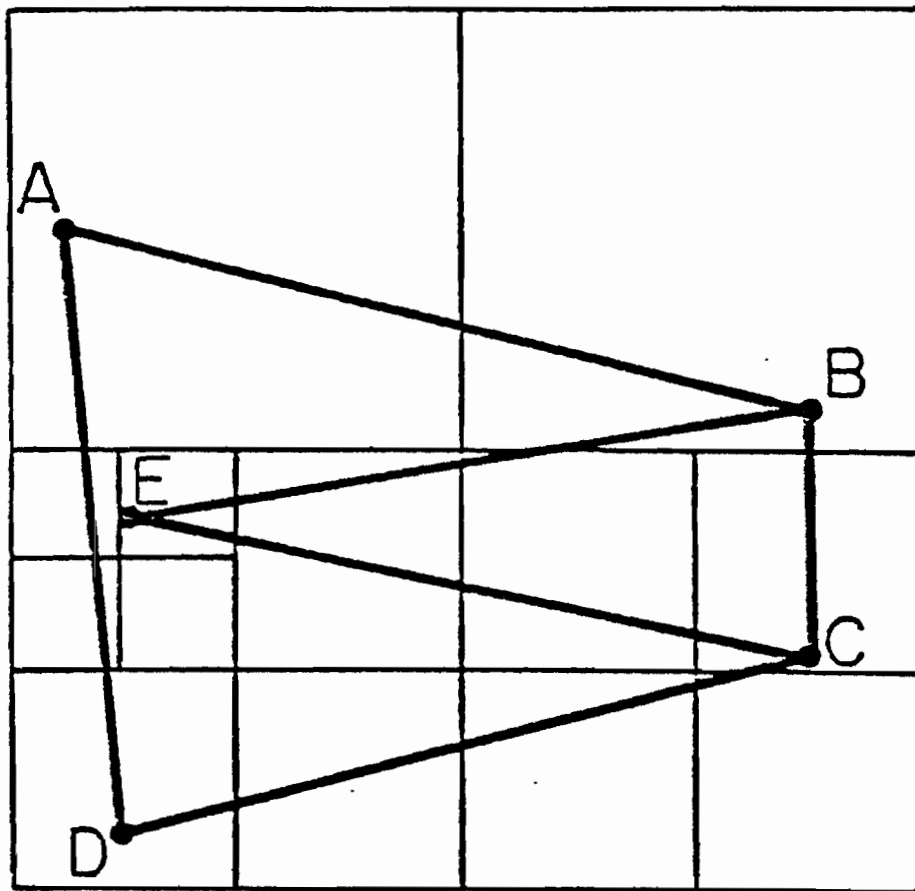


Figure 4-17: The PM₂ quadtree corresponding to Figure 4-12

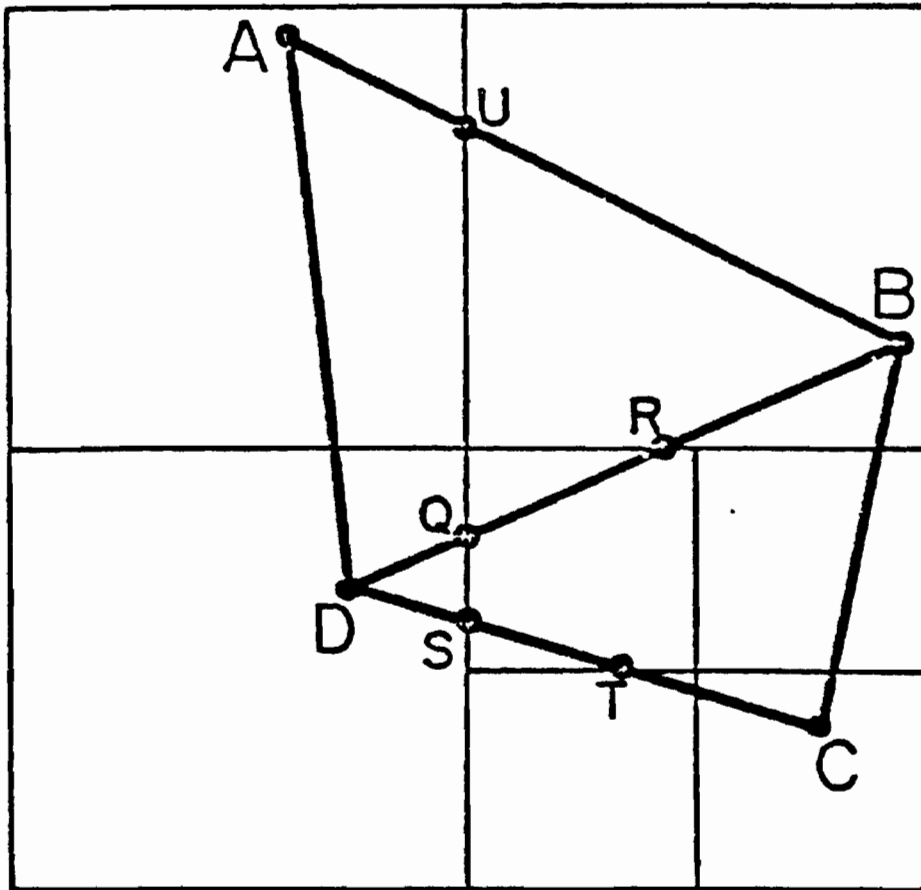


Figure 4-18: Result of using $C3'$, instead of $C3$,
in generating PM quadtree for
Figure 4-16

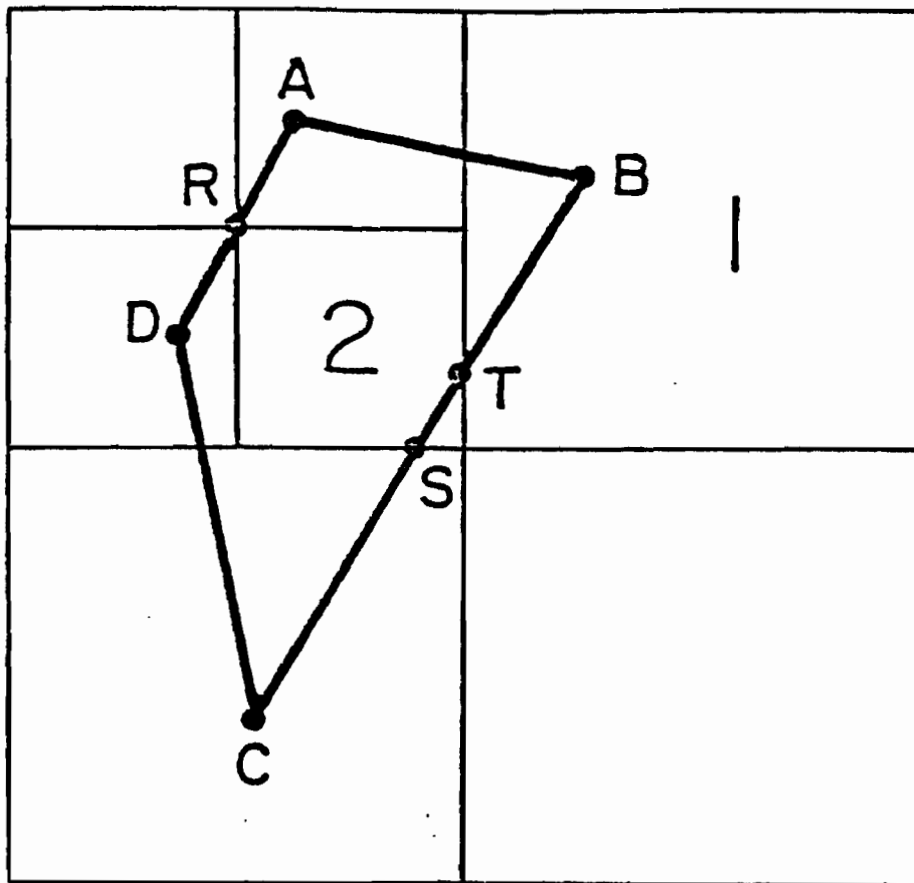


Figure 4-19: Example demonstrating why diagonal neighbors should not be examined prematurely when attempting to perform point-in-polygon determination

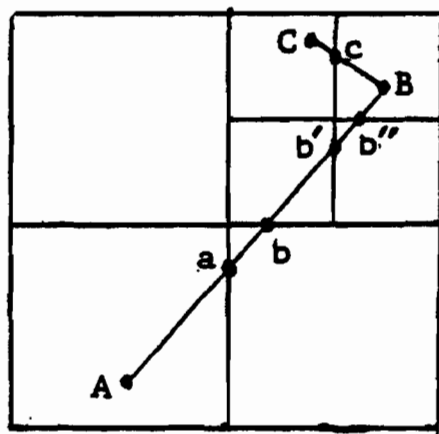
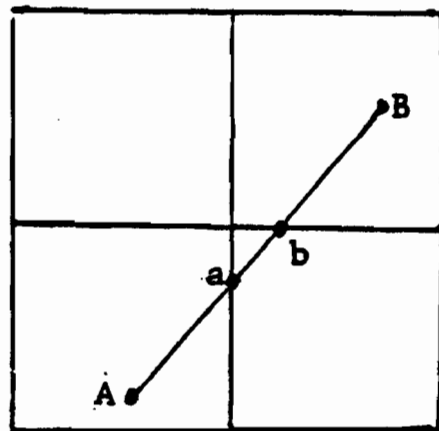


Figure 4-20: Example demonstrating how the inserting of a q-edge can result in the splitting of one that had been inserted earlier

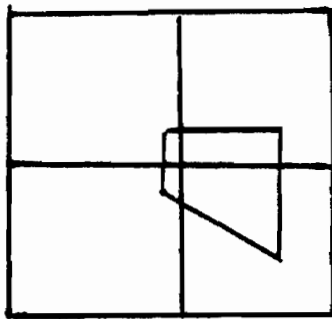
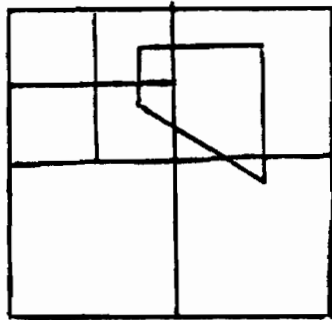


Figure 4-21: Example demonstrating the sensitivity fo the PR quadtree to shifts

CHAPTER 5

CONCLUSIONS

In this dissertation, we have approached the task of analyzing quadtree algorithms on three levels:

1. The analysis of the manipulation of quadtrees irrespective of any interpretation placed on the data stored.
2. The analysis of the manipulation of quadtrees under the assumption that certain trees represent equivalent information.
3. The analysis of the manipulation of quadtrees given a definite interpretation of the data being represented (in our case, the data was interpreted in terms of a geometric model).

The motivating factor behind this research is to lay the groundwork for automating the methods of reasoning about quadtrees presented in this thesis. It is felt that in order to say that a program understands the quadtree data structure, the program would have to be able to respond to questions posed on each of these three levels.

First, we considered the formal analysis of the semantics of the quadtree data structure. The main tool in the development of this semantics was the notion of a theory of neighbor relations. This theory was developed from scratch, illustrating the bare minimum that needs to be manipulated in order to define the neighbor relations between the nodes of the quadtree/pyramid/picture. The particular approach adopted in defining this semantics is also appropriate to a comparison of quadtree algorithms with hierarchical algorithms that are not based on quadtrees, e.g., hextree algorithms [10]. Thus, this presentation of the theory of neighbors is better for the task of analyzing quadtree algorithms than a matrix-based theory.

The theory of neighbors was presented in two stages. First, there was a simple theory that allowed us to prove that two nodes was neighbors. An example of a proof in this system was given in Table 2-3. Although this theory has a certain toy-like

charm, the main results about neighbor relations lie not in this theory, but rather in its metatheory. Thus, in the second stage, we considered metatheorems that posed restrictions on which nodes could be proved to be related. First, we proved that each node had only one neighbor on any given side (Theorem 1 of Section 2.1). Then, we proved (Theorem 3 of Section 2.1) that it was decidable whether or not two nodes were neighbors (and also that given a node, the function that calculates its neighbor is computable). Next, we investigated transformations of subframes that preserved relations (Theorem 4 and Lemma 6 of Section 2.1). Finally, a sequence of lemmas were proved, leading to a theorem (Theorem 9 of Section 2.1) that demonstrated the compactness of our web of neighbors, e.g., one corollary of this theorem is that if a node has a northwestern neighbor, then it also has a northern neighbor.

Having created this theory of neighbors, we turned to the concept of a picture viewed as an association of colors with elements of the neighbor structure. Next, basic picture manipulations (embedding, surrounding, shifting, overlaying) and measures (area and perimeter) were defined. Subsequently, the concept of a pyramid was developed as an extension of the picture coloring technique restricted by certain hierarchical constraints. These definitions culminated in the defining of two types of quadrees (region quadrees and line quadrees) as subsets of the corresponding pyramids. It was shown that the original picture could be reconstructed from either of these quadrees (Theorems 10 and 14 of Section 2.2). The demonstration of this result was less straightforward in the case of the line quadtree and relied heavily on the close relationship between line quadrees and region quadrees noted in Lemma 12 of Section 2.2.

Given the basic definitions for pictures, pyramids, and quadrees, we proceeded to consider their relation in terms of the size of each of these representations, and in terms of the ease of the operation of neighbor finding on each of these representations. Although the sizes of pictures and pyramids are roughly comparable, it was shown that the size of a quadtree, measured as a function of the length of the perimeter of the object stored, could be significantly less than the size of the corresponding pyramid (Theorem 19 of Section 2.3). Although this theorem (as well as Theorem 20 of Section 2.3 on the result of overlaying two quadrees) was originally due to Hunter [12, 13], it

is given a proof herein that has removed the irrelevant geometric assumptions that are common to proofs based on an intuitive understanding of neighbor relations in the quadtree (with the hope that this will make these results easier to adapt when performing the analysis of quadtree variants).

The problem of neighbor finding provides the transition between the section on formalizing quadtree theory and the section on quadtree transformations. First, we are able to use our Neighbor Theory to prove the correctness (Lemma 21 of Section 2.4) of an important neighbor finding algorithm [28]. Such a correctness proof has many uses. Implicit in any formal correctness proof is the potential of some day being able to automatically derive the algorithm. Along with correctness, we also get formal derivations of the basic analysis associated with the algorithm (Theorems 22 and 24 of Section 2.4). However, part of this basic analysis only applies to pyramids and not to quadtrees (here we refer to the aggregate neighbor finding cost being bounded from above by a constant times the number of neighbors sought). That this analysis does not directly extend to quadtrees (as well as a special case where it does extend) is the final topic considered in the section on formal analysis of quadtree algorithms.

The next level of analysis adds an important new concept to our theory. We use the operation of shifting to form a meaningful equivalence class among different quadtrees. Thus, while on the first level, we were analyzing quadtrees in isolation, on the second level, we will be able to consider the ramifications of considering various quadtrees as equivalent. In each equivalence class of quadtrees, there are two important members: 1) the one that minimizes the aggregate cost of neighbor finding and 2) the one (NM) that minimizes the number of nodes in the quadtree. Instead of trying to minimize the aggregate cost of neighbor finding, we look at the related ANF transformation that insures that the aggregate cost of neighbor finding will be bounded from above by a constant when averaged over all the neighbors found.

The ANF transformation is first developed with respect to the chain code representation of a picture to aid in a chain code to quadtree algorithm. Using the ANF transformation (which can be performed in time proportional to the length of the chain code), we derive an algorithm that has a worst case analysis proportional to

previously reported [25] average case analysis for performing this task. We then extend this transformation to directly transform quadtrees (this time with cost linear in the size of the original and resulting quadtrees). This leads us to improve on known results for quadtree to chain code, connected component algorithms, as well as to give alternative algorithms for tasks already known to have a linear cost (e.g., computing perimeters and Euler numbers). Finally, we compare the ANF transformation with the NM transformation, showing that neither transformation subsumes the other (even in the loose sense of being only off by a constant factor).

The third level of analysis is to integrate the quadtree data structure with a problem domain (in our case, storing polygonal maps in the Euclidean plane). First, certain basic correspondences between the Euclidean plane and Neighbor Theory are established. One by-product of this correspondence is a demonstration of the consistency of Neighbor Theory (relative to Euclidean geometry). The task we set ourselves is to minimize the size of the quadtree while maintaining an ease of manipulation. A priori, it is considered that the size of the quadtree that corresponds to a reasonable digitization of a polygonal map is too large for the amount of information it is storing.

The first attempt at solution does not attempt to integrate the structure of the Euclidean plane directly into the data structure, but rather applies a common subtree elimination (CSE) process to compress the region quadtree representation. The results of this compression are analyzed in terms of properties of the geometric objects that were digitized. The analysis of the size of the resulting structure (referred to as a CSE quadtree) comes in three parts. First, we derive an upper bound for the compression occurring from a CSE manipulation of a randomly constructed quadtree. While this is not a particularly interesting case, it does introduce the line of reasoning that is used in subsequent proofs. Next, it is shown that the storage requirements for a convex object is proportional to its length divided by the logarithm of its length. Finally, analysis of a special case of the representation of straight lines yields an upper bound on the cost of representing a line with rational slope. This upper bound is proportional to the square root of the length of the represented line segment. Although these bounds indicate the potential for significant compression, they are still too loose to represent a

complete understanding of the situation. It is not known if the bounds for rational sloped lines can be improved, nor if the bounds for non-rational sloped lines are any better than the case of representing a convex object. However, we ultimately abandon this approach, not because of the meager guaranteed compression, but rather because of the difficulty of translating the compression in storage into speed improvements of basic algorithms. For the computation of area, it is easy to develop an algorithm that runs in time proportional to the size of the resulting CSE quadtree, but for algorithms that require calculations that are functions of neighbors (e.g., perimeter) or algorithms that require manipulation of more than one CSE quadtree (e.g., intersection) the correlation between the cost of the algorithm and size of a CSE quadtree is not so clear.

The following two approaches consider the task of applying point quadrees (instead of region quadrees) to the task of storing polygonal maps. Immediately, the question arises of what points to store. The task of storing a polygonal map is one of representing regions. We recall that in the Voronoi Diagram Construction problem, points are used to represent regions. Thus, one possibility that comes to mind is to discover a tessellation of the map that corresponds to a Voronoi diagram, and then represent that map by a labelling of the set of points that the Voronoi diagram represents. This second attempt leaves us with the task of solving the exact inverse of the Voronoi Diagram Construction problem. Two items are worthy of note. First, it may be necessary to break some regions into subregions. Second, such an approach transforms the point-in-polygon problem into the nearest-neighbor problem. The status of this approach is still an open problem. On the one hand, we have an algorithm that terminates, but produces too many points. On the other hand, we have an algorithm that may not terminate, but need not always produce such an excessive number of points.

The third and last approach also represents the polygonal map in terms of points, but this time it uses points that are already given as part of the map specification, i.e., the vertex points. The presentation of this solution of the problem of storing polygonal maps shows major interactions between our theory of quadrees and both the properties of Euclidean geometry and other data structures (in this case, balanced binary trees). Thus, as we reach the end of our analysis of quadrees, we also see how rich

the knowledge base that must be brought to bear on these problems. This final structure turns out to be both easy to manipulate and have reasonable compression factors. The analysis presented assumes that the points are stored in a PR quadtree. Even better worst case results would be possible if we had dynamic balancing algorithms for point quadtrees that were comparable to the AVL algorithm on binary trees. Of all the open problems in the theory of quadtree data structures, the search for this balancing algorithm is the one that appears farthest from solution.'

References

1. G. M. Adelson-Velskii and Y. M. Landis. "An algorithm for the organization of information." *Soviet Math. Doklady* 3 (1962), 1259-1262.
2. N. Ahuja. On approaches to polygonal decomposition for hierarchical image representation. *to appear in: Computer Vision, Graphics and Image Processing*, 1983 (see also Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Dallas, TX, 1981, 75-80)
3. C. R. Dyer. "Computing the Euler number of an image from its quadtree." *Computer Graphics and Image Processing* 13 (1980), 270-276.
4. C. R. Dyer, A. Rosenfeld, and H. Samet. "Region representation: boundary codes from quadtrees." *Communications of the ACM* 23 (1980), 171-179.
5. C. R. Dyer. "Space efficiency of region representation by quadtrees." *Computer Graphics and Image Processing* 19 (1982), 335-348.
6. C. M. Eastman. "Representations for space planning." *Communications of the ACM* 13 (1970), 242-250.
7. R. A. Finkel and J. L. Bentley. "Quad trees: a data structure for retrieval on composite keys." *Acta Informatica* 4 (1974), 1-9.
8. H. Freeman. "Computer processing of line-drawing images." *ACM Computing Surveys* 6 (1974), 57-97.
9. I. Gargantini. "An effective way to represent quadtrees." *Communications of the ACM* 25 (1982), 905-910.
10. L. Gibson and D. Lucas. "Vectorization of raster images using hierarchical methods." *Computer Graphics and Image Processing* 20 (1982), 82-89.
11. F. Harary. *Graph Theory*. Addison-Wesley Publishing Company, Reading, MA, 1969.
12. G. M. Hunter. *Efficient computation and data structures for graphics*. Ph.D. Th., Department of Electrical Engineering and Computer Science, Princeton University, 1978.
13. G. M. Hunter and K. Steiglitz. "Operations on images using quadtrees." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1 (1979), 145-153.
14. G. M. Hunter and K. Steiglitz. "Linear transformation of pictures represented by quadtrees." *Computer Graphics and Image Processing* 10 (1979), 289-296.
15. E. Kawaguchi, T. Endo, and J. Matsunaga. "Depth-first expression viewed from digital picture processing." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 5 (1983), 373-384.
16. A. Klinger. Patterns and search statistics. In *Optimizing Methods in Statistics*, J. S. Rustagi, Ed., Academic Press, New York, 1971.
17. A. Klinger and C. R. Dyer. "Experiments in picture representation using regular decomposition." *Computer Graphics and Image Processing* 5 (1976), 68-105.

18. D. E. Knuth. *The Art of Computer Programming: Volume 1/ Fundamental Algorithms*. Addison-Wesley Publishing Company, Reading, MA, Second Edition, 1975.
19. D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. *Proceedings of the 8th Annual ACM Symposium on Theory of Computing*, Hershey, PA, 1976, pp. 231-235.
20. M. Li, W. Grosky, and R. Jain. Normalized quadtrees with respect to translation. *Proceedings of the IEEE Conference on Pattern Recognition and Image Processing 81*, Dallas, TX, August, 1981, pp. 60-62.
21. J. J. Martin. Organization of geographical data with quadtrees and least squares approximations. *Proceedings of the IEEE Conference on Pattern Recognition and Image Processing*, Las Vegas, NV, 1982, pp. 458-463.
22. N. J. Nilsson. A mobile automaton: an application of artificial intelligence techniques. *Proceedings International Joint Conference on Artificial Intelligence*, Washington, DC, 1969, pp. 509-520.
23. J. A. Orenstein. "Multidimensional tries used for associative searching." *Information Processing Letters 14* (1982), 150-157.
24. A. Rosenfeld. "Digital straight line segments." *IEEE Transactions on Computers 23* (1974), 1264-1269.
25. H. Samet. "Region representation: quadtrees from boundary codes." *Communications of the ACM 23* (1980), 163-170.
26. H. Samet. "Connected component labeling using quadtrees." *Journal of the ACM 24* (1981), 487-501.
27. H. Samet and R. E. Webber. Line quadtrees: hierarchical data structures for encoding boundaries. *Proceedings of the IEEE Conference on Pattern Recognition and Image Processing 82*, Las Vegas, NV, 1982, pp. 90-92.
28. H. Samet. "Neighbor finding techniques for images represented by quadtrees." *Computer Graphics and Image Processing 18* (1982), 37-57.
29. H. Samet. A top-down quadtree traversal algorithm. Computer Science TR-1237, University of Maryland, 1982.
30. H. Samet. The quadtree and related hierarchical data structures. Computer Science Department, TR 1329, University of Maryland, College Park, MD, 1983.
31. H. Samet and R. E. Webber. Using quadtrees to represent polygonal maps. *Conference on Pattern Recognition and Image Processing 83*, Washington, DC, 1983.
32. M. Shneier. "Two hierarchical linear feature representations: edge pyramids and edge quadtrees." *Computer Graphics and Image Processing 17* (1981), 211-224.
33. I. E. Sutherland, R. F. Sproull, and R. A. Schumacker. "A characterization of ten hidden-surface algorithms." *ACM Computing Surveys 6* (1974), 1-55.
34. G. Toussaint. Pattern recognition and geometric complexity. *Proceedings of Fifth International Conference on Pattern Recognition*, Miami Beach, FL, 1980, pp. 1324-1347.
35. J. E. Warnock. A hidden surface algorithm for computer generated half tone pictures. Computer Science Department, TR 4-15, University of Utah, 1969.

36. Li-De Wu. On the Freeman's conjecture about the chain code of a line. Proceedings 5th International Conference on Pattern Recognition, Miami Beach, FL, 1980, pp. 32-34.

CURRICULUM VITAE

Name: Robert Edward Webber

Permanent address: 2304 Perry Avenue
Edgewood, Maryland 21040

Degree and date to be conferred: Ph.D., 1983.

Date of birth: July 15, 1955.

Place of birth: Danville, Virginia.

Secondary education: Edgewood Senior High School,
Edgewood, Maryland, May 1973.

Collegiate institutions attended	Dates	Degree	Date of Degree
University of Maryland	8/76-8/78	B.S.	May 1978.
University of Maryland	8/78-5/80	M.S.	May 1980.
University of Maryland	5/80-12/83	Ph.D.	December 1983.

Major: Computer Science.

Professional publications:

On formalizing abstract machines, Masters's Thesis, CSC 316,
Computer Science Department, University of Maryland, 1980.

(with A. Rosenfeld, H. Samet, and C. Shaffer) Application of
hierarchical data structures to geographical information systems,
Computer Science TR-1197, University of Maryland, College Park,
MD, June 1982.

(with H. Samet) Line quadtrees: a hierarchical data structure
for encoding boundaries, *Proceedings of the IEEE Conference
on Pattern Recognition and Image Processing*, Las Vegas, NV,
1982, 90-92.

(with H. Samet) Using quadtrees to represent polygonal maps,
Proceedings of Computer Vision and Pattern Recognition 83,
Washington, D.C., June 1983, 127-132.

(with A. Rosenfeld, H. Samet, and C. Shaffer) Application of
hierarchical data structures to geographical information systems

phase II, Computer Science TR-1327, University of Maryland, College Park, MD, June 1983.

(with A. Rosenfeld, H. Samet, and C. Shaffer) Quadtree region representation in cartography: experimental results, *Proceedings of Computer Vision and Pattern Recognition 83*, Washington, D.C., June 1983, 176-177.

(with A. Rosenfeld, H. Samet, and C. Shaffer) A quadtree-based geographical information system, *Proceedings of the Third Scandinavian Conference on Image Analysis*, Copenhagen, July 1983, 231-236.

(with H. Samet) On encoding boundaries with quadtrees, to appear in *IEEE Transactions on Pattern Analysis and Machine Intelligence* (also University of Maryland Computer Science TR-1162).

(with A. Rosenfeld, H. Samet, and C. Shaffer) Quadtree region representation in cartography: experimental results, to appear in *IEEE Transactions on Systems, Man, and Cybernetics*.

Professional positions held:

Teaching Assistant, University of Maryland, College Park, MD.

Instructor, University of Maryland, College Park, MD.

Research Assistant, University of Maryland, College Park, MD.

Assistant Professor, Rutgers University, New Brunswick, NJ.