Quadtree Representation & Compression of Spatial Data

by Xiang Yin

Computer Science

Submitted in Partial Fulfilment

of the Requirements for the Degree of

Masters of Science

Faculty of Mathematics and Science, Brock University

St. Catharines , Ontario

©August 2009

Approved for the Committee:

Dr. Ivo Düntsch

Dr. Ke Qiu

Dr. Michael Winter

Department of Computer Science

Supervisor Dr. Ivo Düntsch

Acknowledgements

I am deeply indebted to my supervisor, Dr. Ivo Düntsch, for his support, patience and encouragement throughout my graduate studies. With his enthusiasm, guidance and great effort to explain things clearly, I devoted to the research field that I am interested in and overcame lots of difficulties.

My thanks also go to the members of my committee, Dr. Ke Qiu and Dr. Michael Winter for their valuable comments and reasonable suggestions.

My friends and colleagues gave various suggestions on my work, which led to many interesting and inspired discussions relating to my research. In particular, I would like to thank Zhaowei Wang, Junqiang Li and Yifeng Li for their helping with various ideas of coding implementation.

Most importantly, I wish to thank my parents, Xuelong Yin and Rongyi Guo for their encouragement, love and support. And I'd also like to thank my wife, Jing Sun, for her understanding and support during the past few years.

Finally, I thank the Department of Computer Science for financial support and providing me with a great working environment.

This document was prepared using TeXnicCenter.

Abstract

Spatial data representation and compression has become a focus issue in computer graphics and image processing applications. Quadrees, as one of hierarchical data structures, basing on the principle of recursive decomposition of space, always offer a compact and efficient representation of an image. For a given image, the choice of quadtree root node plays an important role in its quadtree representation and final data compression. The goal of this thesis is to present a heuristic algorithm for finding a root node of a region quadtree, which is able to reduce the number of leaf nodes when compared with the standard quadtree decomposition. The empirical results indicate that, this proposed algorithm has quadtree representation and data compression improvement when in comparison with the traditional method.

Contents

List of Committee	ii
Acknowledgements	iii
Abstract	iv
List of Tables	viii
List of Figures	ix
1 Introduction	1
1.1 Problem Statement	1
1.2 Organization of the Thesis	3
2 Literature Review	4
2.1 Overview of Quadtree	4
2.2 Common Types of Quadtree	5
2.2.1 Region Quadtree	5
2.2.2 Point Quadtree	7
2.2.3 Edge Quadtree	8

2.3	Features of Quadtree	10
3	Review of Quadtree Representation and Compression	12
3.1	Alternative Ways for Quadtree Representation	12
3.1.1	Pointer-based Quadtree Representation	12
3.1.2	Pointerless Quadtree Representation	14
3.1.2.1	SONTYPE4	15
3.1.2.2	DF-expression	17
3.2	Quadtree Approximation Methods	18
3.2.1	Concept of Approximation	18
3.2.2	Approximation Spaces	19
3.2.3	Approximation Applied in Quadtrees	20
3.2.3.1	Hierarchical Approximation Method	21
3.2.3.2	Forest-based Approximation Method	25
3.3	Compression Methods for Spatial Data	28
3.3.1	Overview of Data Compression	28
3.3.2	Lossless Compression and Lossy Compression	29
3.3.3	Quadtree-based Image Compression	30
4	Algorithm Design	32
4.1	Problem Analysis	32
4.2	Influential Factors of Quadtree Cost	33
4.3	Significance of Quadtree Root Node Choice	35
4.4	Evaluation Standards of Root Node Selection	39
4.5	Algorithm Design for Root Node Choice	41

5	Implementation and Analysis of CORN Algorithm	44
5.1	Techniques for CORN Algorithm Implementation	44
5.1.1	Maximum Connected-Region Search Strategy	44
5.1.2	Maximal Square Search Strategy	49
5.1.2.1	Algorithm Design for Finding Maximal Square	49
5.1.2.2	Further Study for Maximal Square Search	50
5.1.3	Neighbors Search Strategy	54
5.2	Example Illustration	56
6	Empirical Results	63
7	Conclusion	75
	Bibliography	80

List of Tables

5.1	Maximum squares	54
5.2	Maximum squares in bounding box of I	58
6.1	Leaf nodes reduction	71
6.2	Data compression rate comparison	74

List of Figures

2.1	A region I of cells and its decomposition	6
2.2	Quadtree representation of I	6
2.3	Point quadtree in the world map	8
2.4	Edge quadtree for collection of line segment objects	9
3.1	Quadtree representation for SONTYPE4 method	16
3.2	A binary image and its quadtree representation for DF-expression method .	17
3.3	Quadtree decomposition of image I	22
3.4	Quadtree representation of image I	22
3.5	IB(2) and OB(2) for image I	23
3.6	IB(1) and OB(1) for image I	23
3.7	An example for showing the black forest	26
4.1	The cost of quadtree influenced by grid size	34
4.2	The cost of quadtree influenced by image position	34
4.3	A region I and its bounding box	35
4.4	1st decomposition of I	36
4.5	1st quadtree representation of I	36

4.6	2nd decomposition of I	38
4.7	2nd quadtree representation of I	38
4.8	Two Blocks in contact	40
5.1	A binary image and its maximal connected regions	45
5.2	The directions of searching black neighbors	47
5.3	Two special cases in maximum connected-region search	48
5.4	The image after merging and updating labels	48
5.5	Bounding box of image for special case	51
5.6	Maximum square for b_1, b_2, b_3	52
5.7	The special case 1 in maximum square search process	53
5.8	The special case 2 in maximum square search process	53
5.9	Example for neighbors search technique	56
5.10	A region and its binary array	57
5.11	Bounding box of I	57
5.12	Maximum square containing b_1	59
5.13	Maximum square containing b_2	59
5.14	Maximum squares containing b_3 and b_4	61
5.15	Maximum squares containing b_5 and b_6	61
5.16	Intersection point of S_1	62
5.17	Final image decomposition and quadtree representation of I	62
6.1	Image decomposition of Example 1	64
6.2	Quadtree representation of Example 1	64
6.3	Rough set approximation	65

6.4	Image decomposition of Example 3 by traditional method	66
6.5	Quadtree representation of Example 3 by traditional method	66
6.6	Image decomposition of Example 3 by CORN algorithm	67
6.7	Quadtree representation of Example 3 by CORN algorithm	67
6.8	An image containing a disconnected region	68
6.9	Image decomposition of Example 4 by traditional method	69
6.10	Quadtree representation of Example 4 by traditional method	69
6.11	Image decomposition of Example 4 by CORN algorithm	70
6.12	Quadtree representation of Example 4 by CORN algorithm	70
6.13	Image decomposition of Example 5	72
6.14	Quadtree representation of Example 5	72
6.15	Two binary images in real world	73
7.1	Quadtree representation optimization	77

Chapter 1

Introduction

1.1 Problem Statement

As ever-increasing requirement of information storage, spatial data representation and compression have become one of the most popular and important issues in computer graphics and image processing applications. How to represent the whole information contained in a given image by using less time and less space is concerned by many researchers. Nowadays, in order to reduce the storage requirements of (a sequence of) images, the design of efficient data representations has been studied extensively.

Hierarchical data structures [32] are becoming increasingly important techniques in the representation of spatial data, which mainly focus on the data at various levels-of-detail (LOD). And this focusing brings in an efficient representation and an improved execution time [31], also is especially useful for performing set operations.

Quadtrees, introduced in the early 1970s [10], as one of such data structures based on the principle of recursive decomposition of space, have since become a major representa-

tion method for spatial data.

Quadtrees are of great interest, in general, because they enable to solve problems in a manner that focuses the work on the areas where the spatial data is of the largest density. For many situations, the amount of work is proportional to the number of aggregated blocks (e.g., nodes in the tree-structure) rather than to the actual number of pixels in the original image. In such way, they have the potential of reducing the requirement of storage space and improving the efficiency of execution time.

In many real world applications, by using quadtree to implement the representation and compression of spatial data, most of researchers have applied their central work to the three aspects: build up the quadtree data structure, encode all the tree nodes and compress the coding of nodes .

In the present thesis we shall investigate how to provide a better compression result for the given images. Historical research has told us that the compression effect has been dominated by encoding, approximation and compression methods selected [30]. Undeniably, they are all important determining factors. Nevertheless, these factors are based on a common precondition, that is, we have built up the quadtree structure. In essence, previous work has not accounted for two essential questions: (1) According to the given image, how do we build up the corresponding quadtree structure? and (2) Whether this quadtree structure is an optimal representation for the given image? Indeed, the problem with choosing quadtree root node for a given image is still an open problem with no general solution, so that, the goal of my thesis will aim to extend traditional considerations into this new ground.

1.2 Organization of the Thesis

The remainder of this thesis is organized as follows:

Chapters 2 and 3 deal with the relative concepts and previous work of the quadtree representation and compression. We are primarily concerned with an introduction and review of quadtree in Chapter 2. In particular, Chapter 3 will show a brief overview of current popular methods about quadtree nodes encoding, quadtree approximation and data compression.

Chapters 4 and 5 constitute the heart of the thesis. In Chapter 4, a more extensive analysis of the significance of quadtree root node choice will be shown at first. Then, special attention is made in regards to the explanation of our CORN (Choosing an Optimal Root Node) algorithm. Chapter 5 contains the introductions of all the techniques used to implement our algorithm, as well as makes certain rules to choose the maximum square when encountering some special cases. More importantly, we will use several examples to describe each step of the algorithm in detail.

Chapter 6 presents experimental results from the use of our CORN algorithm and an interpretation of their significance.

Chapter 7 concludes our thesis with a set of open questions for future research.

Chapter 2

Literature Review

2.1 Overview of Quadtree

For brevity, quadtree is a class of hierarchical data structures which contains two types of nodes: non-leaf node and leaf node, or you can call them internal node and external node. Owing to its "Divide and Conquer" strategy [2], it is most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions.

Nowadays, in the fields of computer graphics and image processing, a quadtree is always regarded as a way of encoding data that enables to reduce storage requirements by avoiding sub-dividing same areas rather than storing values for each pixel. At this point, quadtree is considered as a spatial index which recursively decomposes an image into square cells of different sizes until each cell has a same value.

According to Samet, the types of quadtrees can be classified by following three principles:

“(1) the type of data that they are used to represent, (2) the principle guiding

the decomposition process, and (3) the resolution (variable or not).” [29]

Currently, quadrees can be classified according to the type of data they represent, including areas, points, lines and curves. More generally, quadrees have been categorized by whether the shape of the tree is independent of the order data is processed. Some common types of quadrees in this kind of classification are: region quadtree, point quadtree, and edge quadtree. In the following section, we will outline these three types of quadtree.

2.2 Common Types of Quadtree

2.2.1 Region Quadtree

The scenario for a region quadtree is as follows: We are given a window consisting of black cells situated in a fixed image area A of dimension $2^m \times 2^m$. A is recursively partitioned into equal sized quadrants until each quadrant consists entirely of unicolored cells. The process can be represented by a tree each non-leaf node of which has four children, corresponding to the four quadrants NW, NE, SW, and SE. Each descendant of a node g represents a quadrant in the plane whose origin is g and which is bounded by the quadrant boundaries of the previous step.

Consider, in a simplest example, the region I placed into an area of dimension $2^3 \times 2^3$, shown in Figure 2.1 [34]. The NW quadrant is entirely made up of white cells, thus leading to a white leaf node which terminates this branch. The NE quadrant contains cells of different color; therefore an intermediate (grey) node is placed into the center of the NE quadrant, which is divided into four quadrants. Each quadrant is homogeneous in color, and thus, four leaf nodes are produced, two of which are white and two of which are black.

The process continues until the whole image I is decomposed into square regions. The complete quadtree for the image of Figure 2.1 is shown in Figure 2.2.

Figure 2.1: A region I of cells and its decomposition

Figure 2.2: Quadtree representation of I

2.2.2 Point Quadtree

The point quadtree [10] is an adaptation of a multidimensional binary search tree used to store point data where the underlying space is decomposed into four quadrants as the points are inserted. It shares the features of all quadtrees but a true tree as the center of a sub-division is always on a point. The shape of resulting tree depends on the order data is processed.

A major difference between the point quadtree and the region quadtree is that the former is concerned with the successive subdivision of image array into four quadrants with equal-size, while the four quadrants decomposed by the point quadtree are usually with different-size. By virtue of this property, the great advantage of the point quadtree is its high efficiency in comparing two dimensional ordered data points. For a comprehensive analysis of the search operations using the point quadtree see [5, 22].

A node expressed in a point quadtree is similar to it in a binary tree. In other words, a node in the point quadtree usually consists of two parts : 4 pointers (the pointers to its four sons) and point data (usually expressed as x, y coordinates and its corresponding value: for example, in the world map, it could be the city's name).

Additionally, point quadtrees are extremely efficient when applied to do search operation in a given region. A striking example is the one that requests the determination of all records within a specified distance of a given record. As an example, suppose in the following Figure 2.3, we wish to find all cities within seven units of a data point with coordinates (70, 60). In such a case, there is no need to search the NW, SE, and SW quadrants of the root node (i.e., Chicago with coordinates (35, 40)). Thus, we can decrease the search region to the NE quadrant of the tree rooted at Chicago. Similarly, there is no need to

search the NW, NE and SW quadrants of the tree rooted at Toronto (i.e., coordinates (60, 75)). Therefore, it is easy to find that the efficiency of the point quadtree lies in restricting the search in the minimum scope, so that many records will not need to be examined.

Figure 2.3: Point quadtree in the world map

2.2.3 Edge Quadtree

The edge quadtree [33] is based on the observation that the number of squares in the decomposition can be decreased by terminating the subdivision whenever each square contains no information or nothing more than a single curve which can be approximated by a single straight line. For example, Figure 2.4 shows an edge quadtree for a collection of line segment objects. Applying this process leads to quadtrees in which long edges are repre-

2.3 Features of Quadtree

So far, as all we have mentioned in this chapter, it is not hard to see that the quadtree has a number of useful features:

- By virtue of its hierarchical nature, quadtree especially facilitates the performance of set operations such as the union (i.e., overlap) and intersection of several images. For an overview of greater details, we invite the reader to consult [16, 17].
- Quadtree provides an efficient data structure for computer graphics. In particular, it is easy to transform a detailed shape description into its original image form.
- The prime motivation for the development of the quadtree structure is the desire to reduce the amount of space necessary to store data through the use of aggregation of homogeneous blocks. More generally, one of striking products of this aggregation is to decrease the executing time of a number operations, such as connected component labeling [28] and component counting [8].

Undoubtedly, the quadtree representation also has some hardly avoidable problems: The largest drawback of the quadtree representation is the sensitivity of its storage requirements to its position. Owing to its changeable root node positions, they may lead to many different kinds of tree structures. Thus, it makes so difficult for us to find an uniform algorithm for building up quadtree structure with the minimal space requirements. Indeed, this problem is especially obvious when the number and size of images are large.

Therefore, our motivation in the present thesis, that is, to design a new algorithm which enables us to make an optimal choice for the root node of quadtree. More importantly, by this algorithm, we will get a fixed quadtree structure and minimize the number of quadtree

leaf nodes, so that it will finally result in a better quadtree representation and (lossless) data compression of the original image.

Chapter 3

Review of Quadtree Representation and Compression

3.1 Alternative Ways for Quadtree Representation

In order to represent quadtrees, there are always two major approaches: pointer-based quadtree representation and pointerless quadtree representation.

3.1.1 Pointer-based Quadtree Representation

In general, the pointer-based quadtree representation [6] is one of the most natural ways to represent a quadtree structure. In this kind of method, every node in the quadtree will be represented as a record with pointers to its four sons. In particular, sometimes, in order to achieve some special operations, an extra pointer from the node to its father will also be included.

In the remainder of this part, we will further introduce two common nodes description

methods of the pointer-based quadtree representation:

- Each node is stored as a record with six fields: The first four fields contain its four sons, labeled NW, NE, SW, and SE; the fifth field is the pointer to its father; the sixth field would be the node type. For example, it may depict the block contents of the image which the node represents, that is, black (the point contains data), white (null), and grey (non-leaf point).
- Each node is stored as a record with eight fields: The first four fields contain pointers to the node's four sons; the fifth field is for the node type; the sixth field is the description of the node. For example, in the city map, it could be the name of a road. The last two fields are X coordinate and Y coordinate of the node, which can be used to get the relationship between each two nodes in the quadtree.

However, for further analysis, we can observe that the pointer-based quadtree representation exhibits some inevitable problems when considering space requirements for recording the pointers and internal nodes. For example, suppose we get a corresponding quadtree representation of a given image, and B and W indicate the number of black and white leaf nodes in the quadtree, respectively. In this way, $(B + W - 1)/3$ additional nodes are necessary for the internal (grey) nodes, and each node also requires additional space for the pointers to its sons, even sometimes, another pointer to its father. As a result, these additional spending would lead to an intolerable problem when dealing with images that are very complex and have large size. In other words, in many real world applications, the images may be so large that the space requirements of their quadtree representations will exceed the amount of memory that is available.

Consequently, nowadays, there are a considerable amount of attention concentrated on another quadtree representation method, that is, pointerless quadtree representation. Next, we would like to emphasize on this kind of representation method, which will be used to represent the spatial data in this thesis.

3.1.2 Pointerless Quadtree Representation

In contrast with the pointer-based quadtree representation, pointerless version of quadtree and its variants have been known for many years [12, 31]. The benefit of this kind of quadtree representation is to define each node of the tree as a unique index. By virtue of the regularity of the subdivision, it is possible for us to compute the location code of each node in the tree entirely in local memory rather than accessing the data structure in global memory. In other words, once the location code is known, the actual node containing the point can be accessed through a small number of accesses to global memory (e.g., by hashing). Much work has been done on employing this idea for representing the spatial data, and we refer the reader to [4].

Furthermore, the pointless quadtree representation can be grouped into two categories: The first regards each image as a collection of leaf nodes; and the second represents the image in the form of a traversal of the nodes of its quadtree. Some general nodes encoding methods based on above two pointerless quadtree representation categories will be briefly discussed in the following description.

3.1.2.1 SONTYPE4

In the spirit of the first type of pointerless quadtree representation, each leaf node is represented by a sequence of directional codes which correspond to an encoding of path from the root node of the tree to the leaf node itself. In essence, in order to reduce space requirement, we only need to concern with the directional codes of all the black nodes since a binary image is represented and only grey nodes are internal nodes, which means, all the white nodes can be dominated by the black nodes.

Even though this method was not published until 1982 [12]. The core idea of SONTYPE4 had already been mentioned by Klinger and Dyer [20] as early as 1970's.

The coding sequence in SONTYPE4 is defined as follows: Each node in the quadtree is represented by an n -element sequence $\{q_i\} = \{q_{n-1}, \dots, q_1, q_0\}$ (n is the number of levels from the objective node to the root node of the quadtree) constructed from the digits $\{0, 1, 2, 3, 4\}$. Let $\{x_i\}$ represent the path of nodes from the root node of the quadtree to x_m , that is, the desired node, such that $x_n = \text{root node of the quadtree}$ and $x_i = \text{Father}(x_{i-1})$. Assuming that codes 0, 1, 2, and 3 correspond to quadrants NW, NE, SW, and SE, respectively, and 4 denotes a don't care. In this way, the encoding of the locational code for node x_m is give by q_n where q_i is defined:

$$q_i = \begin{cases} 0 & : i = m \\ 5 \times q_{i-1} + \text{SONTYPE4}(x_i) & : m < i \leq n. \end{cases}$$

As an illustration of this encoding method, the node 7 in Figure 3.1 would be encoded into the sequence of directional codes $\{q_i\} = \{0, 3, 2\}$, that is, $q = 0 \times 5^2 + 3 \times 5^1 + 2 \times 5^0 = 17$.

Furthermore, the SONTYPE4 encoding method has a number of useful properties:

Figure 3.1: Quadtree representation for SONTYPE4 method

- Firstly, it greatly facilitates us to encode a locational code into a sequence of directional codes by using a combination operations of modulo and integer division. In contrast, it also lends itself easily to decode the number in same way that we get the codes.
- Secondly, increasing the resolution of the image does not require extensive recording of the codes for the current codes.
- Finally, it is possible for us to construct a list by sorting the codes of the black nodes in the quadtree representation with an increasing order. In particular, this list would be a variant of a breadth-first traversal of the black portion of the tree, which means, for any $i < j$, black nodes at level j will appear in front of black nodes at level i in the list. And more importantly, as the length of list grows, this breadth-first property

could bring a surprisingly better approximation for the original image.

3.1.2.2 DF-expression

The DF-expression is proposed by Kawaguchi and Endo [19]. For brevity, it is a compacted array that represents an image in the form of a pre-order tree traversal (i.e., depth first) of the nodes in the quadtree. The result of DF-expression method is a sequence consisting of three symbols 'B', 'W', and '(' corresponding to black, white and grey nodes, respectively. As an example, the Figure 3.2 has ((BW(WBWWBWB((WBBBBBW as its DF-expression. To get more insight into the DF-expression method, we suggest that the reader consult the papers [15, 38].

Figure 3.2: A binary image and its quadtree representation for DF-expression method

In addition, the size of the DF-expression is always dependent on the number of nodes in the quadtree representation. In principle, given a $2^n \times 2^n$ binary image, each alphabet of the DF-expression can be represented by using one or two bits for each internal and external node, for example, the symbols '(', 'B' and 'W' are encoded by binary codes 10, 11 and 0, respectively. In this way, the number of bits required for representing Figure 3.2

is equal to 35.

3.2 Quadtree Approximation Methods

3.2.1 Concept of Approximation

An approximation (represented by the symbol $\approx$) is an inexact representation of something that is still close enough to be useful.

In essence, approximation methods may be applied when incomplete or noisy information prevents from getting exact representations. Many problems in real world applications are either too complex to solve, or too difficult to solve accurately. Thus, even when the exact representation is available, an approximation may yield a sufficiently excellent solution by reducing the complexity of the problem significantly (usually in time and space). For instance, suppose we have a picture of a landscape, from the standpoint of our eyes, we only concern with what the whole contents are inside this picture rather than the details of each object. In other words, maybe we will be attracted to the whole structure of a beautiful tree, while we will never take account of how many branches and leaves it contains.

The type of approximation used is mainly dependent on the available information, the degree of accuracy required, the sensitivity of the problem to the data, and the savings (usually in time and effort) that can be achieved. Although approximation is most often applied to numbers, it is also frequently applied to such domains as mathematical functions, visual perceptions, information systems and physical laws. However, in the thesis, we will only aim to provide for the introduction of approximation methods applied in optimizing the quadtree representation.

3.2.2 Approximation Spaces

As an illustration of approximation spaces in information system [7], the granularity of information can be indicated by equivalence relations on a set U of objects, depending on the classes of which objects are discernible. With each equivalence relation θ , we associate a partition P_θ of U by defining that $a, b \in U$ are in the same class of P_θ , if and only if $a\theta b$. The classes of P_θ satisfy the following requirement:

$$\theta a = \{b \in U : a\theta b\}$$

In essence, we also mention the classes of an equivalence relation when we refer to the classes of its associated partition, and call θa *the class of a modulo θ* .

Let U be a set, and θ is an equivalence relation on U . Then, the pair $\langle U, \theta \rangle$ will be called *an approximation space*; and the relation θ is called *an indiscernability relation*, which means the knowledge of the objects in U extends only up to membership in the classes of θ . The idea now aims to approximate our knowledge with respect to a subset X of U modulo the indiscernability relation θ : For any $X \subseteq U$, we define that

$$\underline{X} = \bigcup \{\theta x : \theta x \subseteq X\}$$

is *the lower approximation or positive region of X* , in other words, the complete set of objects which can be positively (i.e., unambiguously) classified as belonging to target set X ; and

$$\overline{X} = \bigcup \{\theta x : x \in X\}$$

is the *upper approximation or possible region* of X , in other words, the complete set of objects that are possibly members of the target set X .

If $X \subseteq U$ is given by a property P and $x \in U$, then

- $x \in \underline{X}$ means that x *certainly has property P* ,
- $x \in \overline{X}$ means that x *possibly has property P* ,
- $x \in U \setminus \overline{X}$ means that x *definitely does not have property P* .

In particular, *the region of uncertainty or boundary region* is defined as: $\overline{X} \setminus \underline{X}$; and $\underline{X} \cup \overline{X}$ is called *the area of certainty*.

Based on above definitions, a pair of form $\langle \overline{X}, \underline{X} \rangle$ is called *a rough set*. At this point, the accuracy of rough set representation of the set X can be given [26] by the following:

$$0 \leq \alpha(X) = \frac{|\underline{X}|}{|\overline{X}|} \leq 1$$

$\alpha(X)$ describes the ratio of the number of objects which can be positively placed in X to the number of objects that can possibly be placed in X . It also provides a measure of how closely the rough set is approximating the target set.

3.2.3 Approximation Applied in Quadrees

By the virtue of its hierarchical structure, in graphics and image processing fields, the quadtree leads itself to serve as an image approximation machine. Similarly, approximation methods are usually regarded as a tool to optimize the quadtree representation. At this point, by truncating quadtree (i.e., ignoring all nodes below a certain level or neglecting some nodes satisfying optimization strategy), a crude approximation is able to be realized.

For further clarification, in the remainder of this section, we will outline two different quadtree approximation methods.

3.2.3.1 Hierarchical Approximation Method

A sequence of inner and outer approximations to an image defined by Ranade, Rosenfeld and Samet [27], is generally used for the hierarchical approximation.

According to their definitions, the inner approximation consists of treating grey nodes as white nodes, whereas the outer approximation treats them as black nodes. Based on these considerations, a more accurate definition is given by Samet [30]:

“Given an image I , the inner approximation, $IB(k)$ is a binary image defined by the black nodes at levels $\geq k$; the outer approximation, $OB(k)$ is a binary image defined by black nodes at levels $\geq k$ and the grey nodes at level k .”

As a simple example, the region I is placed into an area of dimension $2^3 \times 2^3$. The quadtree decomposition and representation of region I are demonstrated by Figure 3.3 and Figure 3.4. According to Samet’s definitions, Figure 3.5 shows $IB(2)$ and $OB(2)$ for Figure 3.4; and Figure 3.6 shows $IB(1)$ and $OB(1)$ for Figure 3.4.

At this point, suppose we use $\subseteq$ and $\supseteq$ to indicate set inclusion in the sense that $A \subseteq B$ and $B \supseteq A$ imply that the space spanned by A is a subset of the space spanned by B . It is easy for us to obtain a conclusion that $IB(n) \subseteq IB(n-1) \subseteq \dots \subseteq IB(0) = I$ and $OB(n) \supseteq OB(n-1) \supseteq \dots \supseteq OB(0) = I$.

In addition, the hierarchical-based quadtree approximation method has also been used in transmission of binary and grey-scale images. In essence, a number of pyramid-based approaches [36] to solve this problem have already been proposed by Sloan and Tanimoto,

Figure 3.3: Quadtree decomposition of image *I*

Figure 3.4: Quadtree representation of image *I*

Figure 3.5: IB(2) and OB(2) for image I

Figure 3.6: IB(1) and OB(1) for image I

in 1979 [35]. They point out that the largest drawback of this kind of quadtree approximation, that is, the redundant information takes up $1/3$ more spatial space. Thus, it leads to no compression. In order to alleviate this problem, they indicate two optimization restrictions as follows:

- For each node in the quadtree representation, a level number and its coordinate are suggested to be included. However, this information is transmitted only if it is distinct from the value of the node's predecessor.
- The receiver is encouraged to deduce the node's value by analyzing its predecessor and three sibling nodes, so that it can lead to decreased space requirement for implementing transmission operation.

Knowlton also addresses a similar problem (transmission of binary and grey-scale images) by using a binary tree version of a quadtree [21]. In his viewpoint, an image is repeatedly split into two halves alternating between horizontal and vertical splits. In particular, he puts forward that all information needed to be transmitted is the composite value for the root node of the quadtree and the successive sets of differentiators. Therefore, the sequence of transmission is a breadth-first traversal of the binary tree with differentiator values. For a comprehensive review of these two methods, we invite the reader to consult [21, 35].

In the remainder of this section, another quadtree approximation method we will discuss in the following part makes no use of above techniques.

3.2.3.2 Forest-based Approximation Method

A forest of quadrees [18] is a decomposition of a quadtree into a collection of sub-quadrees. Each of subquadtree corresponds to a maximal square, which is identified by refining an internal node to indicate some information about its subtrees. At this point, the forest is a refinement of a quadtree data structure used to develop a sequence of approximations to a binary image, and provide space savings over regular quadrees by concentrating on vital information [13].

Before expanding the discussion of the forest-based approximation method, we need to be aware of some concepts: GB node, GW node, and black forest. A grey node (internal node) is said to be of type GB node if at least two of its sons are black nodes or of type GB nodes. Otherwise, the node is said to be of type GW node.

For example, in Figure 3.4, the nodes E, F, H, I, K, and L are of type GB node and nodes A, B, C, D, G, J, M, and N are of type GW node. Naturally, we can draw a conclusion that each black node or an internal node with a label GB can be regarded as a maximal square.

Based on above concepts, a black forest is defined as follows (We can define a white forest in the same way.):

- It contains the minimal set of maximal squares.
- All the black or GB nodes in each maximal square are not included in any other square.
- The squares in this minimal set would cover the whole black area of the original image.

Therefore, the black forest of Figure 3.4 is $\{A\}$. By giving another example, the black

forest corresponding to Figure 3.7 is {F, 10, 16, 25, 27, M, 38}.

Figure 3.7: An example for showing the black forest

In essence, a forest can also be used to achieve the better approximation for the quadtree representation. If we regard the elements of the forest as black and all remaining nodes as white, we can expand on employing a forest to approximate images. It is so valuable for us to sort the nodes of the forest based on their codes. (e.g., we can use the locational codes given by q_i which we introduced in Section 3.1.2.1 to implement nodes encoding.) For example, in the Figure 3.7, the nodes will appear in the order of 25, 16, F, M, 27, 38, and 10. This order is a partial ordering ($S, \geq$) such that $S_i \geq S_{i+1}$ means the block subsumed by S_i is $\geq$ in size than the block subsumed by S_{i+1} . In this way, for a breadth-first traversal we only need to deal with the nodes in an order that satisfies the above subsumption relation.

Samet gives an explicit explanation for the *FBB*, *FWW*, and *FBW* approximation methods, which reasonably leads to compression in the sense. In other words, they avai-ably reduce the amount of spatial data that is required by encoding the image, as well as the transmission. For an overview of definitions of these methods and the relationship between them, we advise the reader to refer to [30].

Indeed, aside from its superiority with respect to the quality of the resulting approxi-mation, the forest-based approximation contains a number of interesting properties:

- It is obvious that the forest-based approximation method is biased in favor of ap-proximating objects with the shape of a "panhandle", whereas the hierarchical-based approximation method is insensitive to them.
- Owing to the fact that the elements of the forest are encoded by locational codes, such approximation can lead to savings of space requirement whenever the situation satisfies that three out of four sons of any grey node have the same type (i.e., black nodes and GB nodes, respectively, white nodes and GW nodes).
- The total number of nodes in the approximation sequence will never exceed the mini-mum number of black or white nodes in the original quadtree representation. In other words, we can guarantee that this approximation method is always at least as good or better than encoding the quadtree by listing its black nodes (or white nodes). In this way, this kind of approximation will bring in a more efficient compression. Even as larger images are used, the compression effect will become more conspicuous.

3.3 Compression Methods for Spatial Data

3.3.1 Overview of Data Compression

Data compression [23] is the process in order to encode spatial data using fewer bits (or other information-bearing units) than any other un-encoded representation. Some popular compression file formats we are familiar with, such as zip and rar, which, as well as providing the information compression, act as an archiver, storing many source files in a single destination output file.

In general, data compression techniques are commonly used to achieve a low bit rate in the digital representation of signals for efficient processing, transmission, and storage. It is able to reduce the consumption of expensive resources, such as hard disk space and transmission bandwidth. The design of data compression schemes is still an open problem, which involves trade-offs among various factors, including the degree of compression, the amount of distortion introduced (if using a lossy compression scheme), and the computational resources required to compress and decompress the spatial data.

Image compression is one application of spatial data compression on digital images [11]. In principle, its motivation is to reduce redundancy of the image data in order to store or transmit data in an efficient form. In computer graphics and image processing domains, image compression is the process of encoding information using less bits rather than storing values of each pixel in the image.

3.3.2 Lossless Compression and Lossy Compression

For brevity, the core idea of lossless compression algorithm is to exploit statistical redundancy so that it can represent the spatial data more concisely without any mistake. Lossless compression is usually possible because most data in real world has statistical redundancy. Thus, lossless compression algorithms always have two stages in common: The first is a decorrelation stage which exploits redundancy between neighboring samples in a data sequence; and the second stage is an entropy encoder which takes advantage of the decreased variance of the data [1].

In comparison to the lossless compression algorithm, another compression method, called lossy data compression, is available when some loss of data is allowable. In principle, the heart of the lossy data compression method is with respect to how people perceive and deal with the data in question. For example, since the human eye is more sensitive to subtle variations in luminance than in color, image compression could work well in ignoring some of color information.

The lossless compression requires reversibility [25] so that the original data can be reconstructed. It provides a way to obtain the best fidelity for a certain degree of compression. However, the lossy compression accepts some loss of data in order to achieve the higher compression. In some cases, transparent (unnoticeable) compression is desired; while in other cases, fidelity is sacrificed to reduce space requirement for storing spatial data. Thus, during the image processing in the real world, we often use them in combination to realize the best data compression.

In practice, lossless data compression algorithms will always fail to compress some files; indeed, any compression algorithm will necessarily fail to compress any data con-

taining no discernible patterns. Similarly, lossy data compression algorithms also come to a point where compressing again does not make any sense (no better compression rate), whatever an extremely lossy algorithm we use. The difference work principle between lossless and lossy compression will be explicitly demonstrated by dealing with the following string: 36.777777777. By using lossless compression method, this string can be compressed as: 36.[9]7, so that the original string is perfectly recreated, just written in a smaller form. In a lossy compression system, we will directly use 37 to express the string. The original data is lost, at the benefit of a smaller size for compression.

3.3.3 Quadtree-based Image Compression

Generally, when compressing a digital image containing a lot of spatial data, a large amount of memory space will be used for storing the image information. More importantly, owing to the limited network bandwidth, it is possible for us to take a long time to transmit all the spatial data. As a result, we are confronted with an open problem to explore a good image compression technique.

As we mentioned in Section 3.2, it may be worthy of developing an image compression method based on the quadtree approximation. By the virtue of the attractiveness of quadtree approximation approach, it is an effective attempt to decrease the amount of data needed to encode the image, as well as, transmit it. Recall that we can represent a quadtree by merely specifying all of the black blocks or all of the white blocks. Depending on the image, it is possible for us to use the color with the smaller cardinality in order to save storage. In essence, Samet [30] shows an excellent image compression according to the FBW approximation method.

Another lossless binary image compression, called QSBIC [37], addresses the same problem, which consists of four stages:

- Processing stage: Scan original image pixel by pixel, and set each pixel color in its corresponding preprocessing image. The goal of this stage is to reduce the entropy of original binary image.
- Quadtree compressing stage: Build up the BFT (breath first traversal) linear quadtree by traveling the preprocessing image in BFT order.
- Run length compressing stage: Use the run length method to encode the tree list and color list of the BFT linear quadtree.
- Statistical model-based compressing stage: Apply the Huffman coding algorithm to further compress the remaining data of the BFT linear quadtree.

The reader who is interested in these compression methods, can get more details in [30] and [37].

Chapter 4

Algorithm Design

In this chapter, we will aim to provide some solutions for two essential questions mentioned at the beginning of my thesis. Furthermore, a brand-new heuristic algorithm will be designed for deciding the root node of a region quadtree, which effectively reduces the amount of leaf nodes in the quadtree representation when compared with the standard quadtree decomposition.

4.1 Problem Analysis

In the “Problem Statement” part of Chapter 1, we have pointed out that the result of compression rate for a given image is not only influenced by encoding, approximation, and compression methods we choose, but also has a compact relation with its quadtree representation according to the original image. Even, when viewing all these factors, in our standpoint, the former is only the external factors, and the latter is an internal factor which has a greater impact on the compression result.

At this point, in the remainder of this chapter, we will aim to answer three main ques-

tions:

- Why does the root node choice of a quadtree have an extremely important meaning for image compression?
- What criteria do we use to evaluate the selections of quadtree root node?
- How do we find a good root node choice for a quadtree?

4.2 Influential Factors of Quadtree Cost

Past experience has taught us that the space cost of quadtrees is measured by the number of their leaf nodes, since the number of nodes in a quadtree is directly proportional to the number of its leaves. This proof procedure has been demonstrated by Horowitz and Sahni [14]. Therefore, it may be noted that minimizing the total number of leaf nodes of a quadtree is equivalent to minimizing its total number of nodes (leaf nodes and non-leaf nodes).

In essence, two factors often greatly influence the total number of leaf nodes of a quadtree: the size of the coordinate grid and the position of the image on the grid. Figure 4.1 clearly demonstrates that reducing the grid size may result in a less cost of quadtree; and Figure 4.2 shows that we may obtain a cheaper quadtree cost by moving the image position.

Furthermore, these two crucial factors are just decided by the root node choice of the quadtree, which is the main reason why the root node choice is of great significance in our consideration.

Figure 4.1: The cost of quadtree influenced by grid size

Figure 4.2: The cost of quadtree influenced by image position

4.3 Significance of Quadtree Root Node Choice

As mentioned in last section, it may be argued that in the visual field, the choice of the base frame is somewhat arbitrary and, as a consequence, the root node of the quadtree representation of an image I may be to some extent variable. Indeed, if we aim to minimize the number of black nodes of a quadtree representation of I , then the choice of the root node may influence the size of the quadtree. A striking example is the case, when an optimal position with respect to a central root node is shifted by one pixel, see, e.g. the discussion in [9]. We shall illustrate this with another example [29]. An image and its bounding box are shown in Figure 4.3.

Figure 4.3: A region I and its bounding box

Following [29], we place the image into an 8×8 grid as shown in Figure 4.4; the corresponding quadtree can be found in Figure 4.5. It contains 43 leaf nodes, 20 of which are black.

If we have the freedom to choose where the image I is placed into an area A , then a

Figure 4.4: 1st decomposition of I

Figure 4.5: 1st quadtree representation of I

smaller number of leaf nodes can be obtained. Figure 4.6 shows a different placement, and Figure 4.7 depicts the corresponding quadtree. This second quadtree contains only 34 leaf nodes, 14 of which are black.

Above analysis clearly demonstrates the significance of quadtree root node choice. Obviously, these results indicate that a good choice of the quadtree root node will bring in a great improvement for the quadtree representation, as well as the final image compression. In particular, when dealing with the image in a large size, this benefit would be much more conspicuous. However, in the meantime, a new and more severe problem is generated, which involves the difficulty in evaluating among the different quadtree root node choices according to the same image. Thus, these observations now lead to the following question:

- Suppose we are given an image of black cells with bounding box dimension $n \times m$. Which position of the root node will minimize the number of black leaf nodes in the quadtree decomposition of A ? In other words, how do we make a choice among a number of candidate quadtree root nodes?

For further discussion, we would like to emphasize that the goal of this thesis is to investigate how the spatial data contained in a given image could be efficiently represented in the quadtree structure by using as little data space as possible. Since the difference in the storage requirements is so significant, at times it may be deemed worthwhile to minimize the space requirement of the image stored in the quadtree, especially when the size of images is large.

Given an image, the quadtree that contains the least number of leaf nodes among all the possible quadtrees that represent the same image is called the normalized quadtree [3] of the image. Therefore, our final motivation is changed to design some algorithm to choose

1	2	3	4	5	6	7
8	9	10	11	12	13	
14	15	16		17		18
19	20					
21	22	23		24		
25	26					
27		28	29	30		
			31	32		
		33	34			

Figure 4.6: 2nd decomposition of I

Figure 4.7: 2nd quadtree representation of I

a suitable root node under certain specific criteria, so that it will lead to an unique quadtree structure and finally build up a normalized quadtree representation within most of cases. In what follows, we shall exhibit these evaluation standards for the quadtree root node choice.

4.4 Evaluation Standards of Root Node Selection

For brevity, we call a (closed) square of size $2^k \times 2^k$ containing only black cells a *block*.

We say that two blocks s, t are *in contact* if $s \cap t \neq \emptyset$, i.e.,

- s and t contain a common cell (Figure 4.8(a)), or
- s and t contain a common edge of a cell (Figure 4.8(b)), or
- s and t contain a common corner point (Figure 4.8(c)).

A set S ($S = \{s_1, s_2, \dots, s_n\}$) of blocks is called a *contact set*, if $s_1 \cap s_2 \cap \dots \cap s_n \neq \emptyset$, i.e., the blocks in S have at least one common point of contact.

Furthermore, if A is a point (i.e., the intersection of two orthogonal edges), the *neighbor number* n_A of A is the sum of the size of all blocks of which A is a corner point.

In order to make an assessment for each candidate quadtree root node, in the sequel, we would like to present three main criteria in order of priority for selecting a root node A :

1. The choice of A minimizes the size of the image space $2^t \times 2^t$ containing I .

Let $Image_x(Image_y)$ be the length of horizontal (vertical) side of its enclosing rectangle. Defining k such that

$$2^k < \max(Image_x, Image_y) \leq 2^{k+1},$$

Figure 4.8: Two Blocks in contact

thus, the optimal grid size (minimum size) is either $2^{k+1} \times 2^{k+1}$ or $2^{k+2} \times 2^{k+2}$. It is very easy to prove this conclusion: It is clear that the image won't fit in a grid of size less than $2^{k+1} \times 2^{k+1}$; equally, a grid of size greater than $2^{k+2} \times 2^{k+2}$ is also non-optimal because an image of size less than or equal to $2^{k+1} \times 2^{k+1}$ can be completely covered in a quadtree of a $2^{k+2} \times 2^{k+2}$ grid. So, the optimal grid size can only be $2^{k+1} \times 2^{k+1}$ or $2^{k+2} \times 2^{k+2}$. (In some special cases, all the candidate root nodes cannot expand the original image into the size of $2^{k+1} \times 2^{k+1}$, so that we have to expand the original image to the size $2^{k+2} \times 2^{k+2}$.)

In addition, it may be necessarily noted that the extra space spent on the larger image-expansion would far exceed the space saved from reducing some black leaf nodes of the quadtree representation under this situation, especially, with images of a large size.

As analysis above, the primary standard to evaluate the root node choice is whether it

can minimize the size of the extended image. For example, suppose we have a given binary image with a bounding box of dimension 13×14 . In this case, we prefer choosing root node by which the original image is only needed to expand to the size of 16×16 , and excluding all the candidate choices have to expand the image size to 32×32 , even though they maybe contain less black leaf nodes corresponding to their quadtree representations.

2. A block of maximal size will be one leaf node in the finale quadtree representation.

In our viewpoint, the largest black square in the original image is the most important factor to influence the number of black leaf nodes in the final quadtree representation. As a simple example, if there is an image with the size 100×100 pixels (bits), and the largest square of which is 32×32 . In such situation, if after image-expansion, we can only need to use one leaf node to represent this square, namely, 1023 cells space would directly be saved.

3. The size of neighbors of A which are blocks is maximized.

This standard guarantees that we could combine with utmost black blocks in the original image when we build up its corresponding quadtree representation.

4.5 Algorithm Design for Root Node Choice

Minimizing the number of black leaf nodes will result in a better (lossless) compression of the original image, and we will use this number as a metric. Associating with the evaluation standards mentioned above, in this section we will outline a heuristic algorithm which decreases the number of black nodes required to represent an image.

The cells of the bounding box are scanned from left to right, row by row, starting at the upper left hand corner of the bounding box of I . The pseudocode of the CORN¹ algorithm proceeds as follows:

1. repeat

Progress to the next unvisited black cell b_i and record its position.

Find a block s_i of maximal size (i.e., $2^n \times 2^n, n \geq 0$) that

- (a) contains b_i , and
- (b) maximizes the number of unvisited cells.

Change black color of the cells of s_i into, say, red.

until all black cells have been visited.

{At this stage, we have a set $B = \{b_1, \dots, b_n\}$ of points and a set of blocks $S = \{s_1, \dots, s_n\}$.}

2. Find the maximal blocks, say, $s_1, \dots, s_n$ corresponding to each cell b_i ($i = 0, 1, 2, \dots, n$).
3. Find all maximal contact sets $S_1, \dots, S_m$ from S containing at least one of $s_1, \dots, s_k$ and for each S_j the set C_i of points in the intersection.
4. Order the points in $\bigcup \{C_i : 1 \leq i \leq m\}$ by their neighbor numbers in non-increasing order, say $A_1, A_2, \dots, A_k$ such that $n_{A_1} \geq n_{A_2} \geq \dots \geq n_{A_k}$.
5. Let t be the smallest number such that $n, m \leq 2^t$; in other words, a block with side length t is a smallest block that can contain I .

¹Choosing an **Optimal Root Node** - we are aware that the name is somewhat optimistic.

6. Let $i = 0, j = t$.

repeat

repeat Increase i by 1.

Decompose the image with A_i as a root node. If the resulting block has side length 2^j then choose A_i as root node and stop.

until $i = k$.

{At this stage, none of the A_i will allow using 2^j as a side length, so we try the next smallest block.}

Increase j by 1.

until FALSE

Now, we have presented a brand-new algorithm which could find the normalized quadtree of an image in most of situations. However, the problem with its implementation also becomes a more difficult and challenging task. Thus, in the next chapter, much more work will concentrate on techniques used to realize the CORN algorithm.

Chapter 5

Implementation and Analysis of CORN Algorithm

In this chapter, some methods and techniques applied in realizing the CORN algorithm will be introduced first. After that, our interests will move to analyze some special cases in practice, as well as put forward their feasible solutions. Finally, some examples will be provided for us to better understand details of the CORN algorithm.

5.1 Techniques for CORN Algorithm Implementation

5.1.1 Maximum Connected-Region Search Strategy

Maximum connected region search is an important operation in quadtree root node selection process. It is analogous to finding the different connected components of the whole image (i.e., in this theses, the goal is to find the independent black regions).

For example, the image of Figure 5.1(a) has two connected regions. In order to identify

these separate regions by program, we have to rely on some algorithm.

Figure 5.1: A binary image and its maximal connected regions

The traditional method of performing this operation [28] would be a “breadth-first traversal” approach, which scans the image row by row, from left to right, and assigns the same label to the adjacent black cells. During this process pairs of equivalences may be generated, and it will bring two more steps: one to merge the equivalences and the other one to update the labels associated with the various cells to reflect the merger of the equivalences [24].

In the remainder of this section, we would like to present another method to achieve the same objective. In describing the procedure we will use Figure 5.1(a) as a running example. In this way, our method will proceed as follows:

1. Set $i = 0, j = 1$.

repeat

- (a) Scan the image row by row, from left to right, until we find the first unvisited black cell c_i . Then, add c_i into an array, say *BlackArray*.
- (b) Find black neighbors (adjacent cells) in East and South sides, as well as NE and SE corners of c_i (show in Figure 5.2).
 {The reason why we search black neighbors in these directions is because we want to guarantee our method will hold in two special cases, see Figure 5.3.}
- (c) Make a judgement for each neighbor of c_i : If its color is black, we add it to the back of *BlackArray*.
- (d) Change black color of c_i into, say, red.

repeat

Find black neighbors of *BlackArray*[j], and add them to the end of *BlackArray*.

Increase j by 1.

until all black cells in *BlackArray* have found their neighbors.

- 2. Assign the label of all the cells in the *BlackArray* as region “A”. {In this example, “A” is shown in Figure 5.1(b).}
- 3. Clean out all the cells in the *BlackArray*.
- 4. Increase i by 1, and set $j = 1$.

until all black cells in the image have been visited.

{After this stage, we have a region set $S = \{A, \dots, Z\}$.}

5. Find equivalence pairs among all the labels.

{In this example, when cell 32 is processed, it has no label, but its South neighbor 40 is in red color and labeled as “B”. Thus, we will label cell 32 as “C”, meanwhile, we will record “B” and ”C” are equivalence pairs. Similarly, when cell 36 is processed, its East block 37 is in red color and labeled as “B”. Therefore, we label cell 36 the same as its East cell 35 which is “D”. At the same time, we record “B” and “D” are equivalence pairs. (The result after this stage is shown in 5.1(b).)}

6. Merge all the equivalence pairs and update the labels.

The final result of this example is shown in Figure 5.4.

Figure 5.2: The directions of searching black neighbors

Figure 5.3: Two special cases in maximum connected-region search

1 _A	2 _A	3 _A	4 _A	5	6	7	8
9 _A	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29 _B	30	31	32 _B
33	34	35 _B	36 _B	37 _B	38 _B	39 _B	40 _B
41	42	43 _B	44	45 _B	46 _B	47 _B	48 _B
49	50	51 _B	52	53 _B	54 _B	55 _B	56 _B
57	58	59 _B	60 _B	61 _B	62 _B	63 _B	64 _B

Figure 5.4: The image after merging and updating labels

5.1.2 Maximal Square Search Strategy

5.1.2.1 Algorithm Design for Finding Maximal Square

In principle, when considering to implement the first stage of CORN algorithm by computer program (i.e., find the maximum square for a given black cell), it seems to become a much more complicated problem. In this part, we will present a tricky method to deal with this problem.

By explaining each step of our method, we will use the image of Figure 5.4 as a running example. At this point, the algorithm for finding the maximal squares is devised as follows:

1. Set $i = 1$.

repeat

- (a) Progress to the next unvisited black cell b_i and record its position.
- (b) Explore and record the utmost size of the square (actually, in practice, we always turn to find the utmost side-length of the square) which contains b_i .

{In this example, suppose we wish to find the maximum square containing black cell 37. Firstly, we need to count the number of continuous black cells in the same row and same column of cell 37. In this case, the number of continuous black cells in the same row as cell 37 is 6 (i.e., cells 35, ..., 40); and the number of continuous black cells in the same column as cell 37 is 5 (i.e., cells 29, 37, 45, 53, 61). Therefore, the utmost side-length of the square containing cell 37 is 4.}

- (c) Check whether this utmost square exists or not. If we fail to find the suitable square, then we halve the utmost side-length and check it again. Repeatedly to do this checking operation until we find one suitable square or the utmost side-length of square is equal to 1 (In the worst case, the maximum square for the current cell is the cell itself, whose side-length is 1).
- (d) Change color of the cells within the square, say s_i , into red. Then, increase i by 1.

until all cells have been visited.

{At this stage, we have a square set $S = \{s_1, \dots, s_n\}$.}

2. Compare with the side-length of all maximum squares, and record the side-length of maximal square. {In this example, the maximal side-length of all squares contained in Figure 5.4 is 4.}

5.1.2.2 Further Study for Maximal Square Search

The algorithm introduced in the last section is available to decide an unique maximal square s_i for each corresponding cell b_i in most of situations, however it has not taken account of a special case:

- For a cell, if there exist more than one square with the utmost side-length. How do we make a choice among these squares?

For further clarification, we will use the image of Figure 5.5 as an example to specify our selecting principle under this special situation.

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49

Figure 5.5: Bounding box of image for special case

Figure 5.6 shows the maximum squares for cells b_1 , b_2 , and b_3 , which in this example correspond to cells 1 (Figure 5.6(a)), 8 (Figure 5.6(b)), and 27 (Figure 5.6(c))¹. Since these squares are all unique and maximum with regard to their corresponding cells, our algorithm works quite well until we try to find the maximum square for cell b_4 (i.e., cell 33).

Figure 5.7 shows the two possible choices for the maximum square including cell b_4 ; the similar situation recurs when progressing to search the maximum square with cell b_5 (In this example, b_5 is cell 36.), see Figure 5.8.

As mentioned in the first stage of the CORN algorithm, there are two principles for choosing a block s_i of maximal size: first, contains b_i ; second, maximizes the number of unvisited cells. In essence, the goal of these standards is in order to cover all the black cells in the image by using the least number of blocks (squares). Thus, now, we would prefer to making a rule for the maximum square choice under this particular case:

¹The maximum square of b_1 , b_2 , and b_3 are shown by yellow block in each image.

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49

(a)

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49

(b)

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49

(c)

Figure 5.6: Maximum square for b_1, b_2, b_3

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49

Figure 5.7: The special case 1 in maximum square search process

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42
43	44	45	46	47	48	49

Figure 5.8: The special case 2 in maximum square search process

Search the square from the lowest right corner of the domain where the maximum square could exist. Then, repeatedly check the square from right to left, from bottom to top, until we find the first suitable square.

According to such rule, we will choose the yellow block shown in the second image of Figure 5.7 as the maximum square for cell b_4 . Similarly, the maximum square with respect to b_5 would be the yellow block in the right image of Figure 5.8. And the resulting squares of this example are shown in Table 5.1; the cell b_i is shown in bold face.

Table 5.1: Maximum squares

$s_1 :$	{1}
$s_2 :$	{8, ..., 11, 15, ..., 18, 22, ..., 25, 29, ..., 32}
$s_3 :$	{27, 28, 34, 35}
$s_4 :$	{33, 34, 40, 41}
$s_5 :$	{22, ..., 25, 29, ..., 32, 36, ..., 39, 43, ..., 46}
$s_6 :$	{40, 41, 47, 48}

5.1.3 Neighbors Search Strategy

The stage 4 of the CORN algorithm has involved the examination of the total number of blocks “adjacent” to each intersection point. We can speak of these adjacent blocks as “neighbors”.

A new neighbors search technique, called NST for short, which will be discussed in this section, is quite different from the traditional “Neighbor-Finding Techniques”[29] (p.

197 - 201) in following three aspects:

- The traditional method is based on the quadtree representation; while NST method is according to the image (Actually, the quadtree has not yet been created at this moment).
- The target of the traditional method is the nodes in the quadtree; while the object of the NST method is the points in the image.
- The effect of the traditional method is used to optimize the quadtree representation; while the function of the NST method is used to evaluate the candidate quadtree root node choices.

Formally, we would like to define the NST method as following formula:

$$NST(P, C) = B.$$

In such formula, “P” is for the point (In this thesis, it is especially for the point of candidate quadtree root node.); “C” is for the corner (i.e., the direction for search) which could be NE, SE, NW, or SW; “B” is the sum of cells contained in the block (the maximal black square) corresponding to the corner “C”.

As a simple example, in Figure 5.9, in order to find the total number of blocks adjacent to point *A*, we need to orderly check the maximal squares (Certainly, these maximal squares must contain point *A*.) in its NE, SE, NW, and SW corners. By using the NST method, $n_A = NST(A, NE) + NST(A, SE) + NST(A, SW) + NST(A, NW) = 1 + 16 + 4 + 4 = 25$; similarly, for point *B*, $n_B = NST(B, NE) + NST(B, SE) + NST(B, SW) + NST(B, NW) = 0 + 64 + 4 + 0 = 68$.

Figure 5.9: Example for neighbors search technique

5.2 Example Illustration

In order to explain each stage of the CORN algorithm in details, in this part, we will use Figure 5.10(a) as a running example.

Suppose that the input consists of a binary array I with a bounding box of dimension $m \times n$; an entry “1” indicates a black cell and an entry “0” indicates a white cell, see Figure 5.10(b). According to the CORN algorithm, the procedure of searching its quadtree root node proceeds as follows:

1. Find the bounding box of I , see Figure 5.1.1.

(a)

0	0	1	1	1	1
0	0	1	1	1	1
0	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	0	0
1	1	1	0	0	0

(b)

Figure 5.10: A region and its binary array

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

Figure 5.11: Bounding box of I

2. Starting at the upper left hand corner of bounding box, scan row by row, from left to right, until finding the first unvisited black cell b_1 (In this example, b_1 is cell 3). Record the position of cell 3.
3. Find and record the block s_1 , which consists of cells $\{3, \dots, 6, 9, \dots, 12, 15, \dots, 18, 21, \dots, 24\}$ (s_1 is the green block shown in Figure 5.12(a)).
4. Change the cells of s_1 into red (see Figure 5.12(b)).
5. Scan the bounding box from cell 4, until finding cell b_2 (cell 14). Then, search and record the block s_2 , which contains cells $\{14, 15, 20, 21\}$ (see Figure 5.13(a)). In the same manner, change its color into red (Figure 5.13(b)).
6. Since $s_1 \cap s_2 \neq \emptyset$, which means these two blocks are in contact. Therefore, put them into the contact set S_1 , namely, now, $S_1 = \{s_1, s_2\}$.
7. Repeat above operations until all cells have been visited. (Figures 5.14 and 5.15 demonstrate the maximum squares for b_3, b_4, b_5, b_6 . And the resulting squares are shown in Table 5.2; the cell b_i is shown in bold face.)

Table 5.2: Maximum squares in bounding box of I

$s_1 :$	$\{\mathbf{3}, \dots, 6, 9, \dots, 12, 15, \dots, 18, 21, \dots, 24\}$
$s_2 :$	$\{\mathbf{14}, 15, 20, 21\}$
$s_3 :$	$\{\mathbf{19}, 20, 25, 26\}$
$s_4 :$	$\{26, \mathbf{27}, 32, 33\}$
$s_5 :$	$\{21, 22, 27, \mathbf{28}\}$
$s_6 :$	$\{25, 26, \mathbf{31}, 32\}$

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(a)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(b)

Figure 5.12: Maximum square containing b_1

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(a)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(b)

Figure 5.13: Maximum square containing b_2

8. Find the maximal square among $s_1, \dots, s_6$ by comparing with their size (At this stage, we get s_1 is the maximal square in current image).
9. Find all the contact sets containing s_1 (In this example, there is only one contact set, namely, contact set $S_1 = \{s_1, s_2, s_3, s_4, s_5, s_6\}$).
10. Check the intersection part of S_1 and get the common point of contact squares is the NW corner of cell 27, see Figure 5.16.
11. Use the NST method to count the neighbors number of point A, that is, $n_A = NST(A, NW) + NST(A, NE) + NST(A, SW) + NST(A, SE) = 1 + 16 + 4 + 1 = 22$.
12. Since there is only one candidate root node choice, and point A satisfies three evaluation criteria defined in the Section 4.4, point A is the final quadtree root node choice.

The final image decomposition and its corresponding quadtree representation are shown in Figure 5.17. We can find that the total black nodes number in the final quadtree representation is reduced from original 26 to 8.

In the next chapter, a number of experimentations will be included to demonstrate the effect and value of the CORN algorithm.

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(a)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(b)

Figure 5.14: Maximum squares containing b_3 and b_4

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(a)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

(b)

Figure 5.15: Maximum squares containing b_5 and b_6

Figure 5.16: Intersection point of S_1

Figure 5.17: Final image decomposition and quadtree representation of I

Chapter 6

Empirical Results

In this chapter, some experimentations are employed to exhibit the efficiency of the CORN algorithm. Meanwhile, these empirical results will be compared against the traditional method for demonstrating the significance of our algorithm.

Traditionally, the root node is always placed in the center of the chosen image area or the image to be considered.

In computer graphics field, adherence to the traditional method, Example 1 puts the root node in the center of the black region of Figure 5.10(a). The image decomposition and its corresponding quadtree representation are shown in Figure 6.1 and Figure 6.2. At this point, the CORN algorithm reduces the number of relevant leaf nodes by more than half (see Figure 5.17).

With regard to visual field, the CORN algorithm retains its superiority relative to the traditional method. As an example, in comparison with Figures 4.4 and 4.6 in Chapter 4, the CORN algorithm demonstrates it is an essential tool to offer a good root node choice and bring in a good quadtree representation of the given image data.

Figure 6.1: Image decomposition of Example 1

Figure 6.2: Quadtree representation of Example 1

Furthermore, in data mining domain, granular computing is closely related to the depth of the detail of information with which we are presented, or choose to process. In spatial cognition and image processing, such detail is given by the resolution of an image. The quadtree representation of an image offers a quick look at the image at various stages of granularity, and successive quadtree representations can be used to represent change [39].

Figure 6.3 is a classical example in the rough set model. Figure 6.4 shows the decomposition of the inner approximation of the ellipse (i.e., the green region) in Figure 6.3 by using the traditional method, and Figure 6.5 performs its quadtree representation. However, a better result is obtained with the use of the CORN algorithm instead of the traditional method, the image decomposition and quadtree representation are shown in Figure 6.6 and Figure 6.7.

Figure 6.3: Rough set approximation

In the previous examples, there is only one connected black region in each example. Indeed, we also have a great interest in whether the CORN algorithm could be good at dealing

Figure 6.4: Image decomposition of Example 3 by traditional method

Figure 6.5: Quadtree representation of Example 3 by traditional method

Figure 6.6: Image decomposition of Example 3 by CORN algorithm

Figure 6.7: Quadtree representation of Example 3 by CORN algorithm

with the image containing more than one connected regions. Figure 6.8 offers an image including a disconnected region. Figure 6.9 and Figure 6.10 respectively show the image decomposition and quadtree representation according to the traditional method; while, basing on the CORN algorithm, a much more fantastic result is displayed in Figure 6.11 and Figure 6.12, which reduces nearly three quarters of relevant leaf nodes in comparison with the traditional method.

Figure 6.8: An image containing a disconnected region

The worst case for a quadtree of a given depth in terms of storage requirement occurs when the region corresponds to a “checkerboard” pattern. In other words, each black cell in the original image will be expressed by one leaf node in the final quadtree representation. The yellow region in Figure 6.3 is such an obvious example. In this case, the CORN algorithm and traditional method will handle the image in the same manner, that is, choosing its center point as the quadtree root node. The image composition and quadtree representation

Figure 6.9: Image decomposition of Example 4 by traditional method

Figure 6.10: Quadtree representation of Example 4 by traditional method

Figure 6.11: Image decomposition of Example 4 by CORN algorithm

Figure 6.12: Quadtree representation of Example 4 by CORN algorithm

of this worst case are shown in Figure 6.13 and Figure 6.14, respectively.

According to above experimental results, a brief indication about the reduction performance of the choice of the root node is shown in Table 6.1.

Table 6.1: Leaf nodes reduction

Experiment	Method	Black	White	Total
Example 1	Center (Figure 6.1, 6.2)	20	26	46
	CORN (Figure 5.17)	8	11	19
Example 2	Center (Figure 4.4, 4.5)	20	23	43
	CORN (Figure 4.6, 4.7)	14	20	34
Example 3	Center (Figure 6.4, 6.5)	37	30	67
	CORN (Figure 6.6, 6.7)	19	45	64
Example 4	Center (Figure 6.9, 6.10)	23	35	58
	CORN (Figure 6.11, 6.12)	5	11	16
Example 5	Center (Figure 6.13, 6.14)	39	85	124
	CORN (Figure 6.13, 6.14)	39	85	124

In essence, all the experimental results in Table 6.1 have indicated that the CORN algorithm is always at least as good or better than the traditional method with respect to the image decomposition and quadtree representation, even as well as including the spatial data compression. However, it may be worthy of note that, until now, all the experimental images we choose are confined to a very small size. At this point, this limitation may lead to the following doubt:

- In practical applications, suppose we are given some images with a much larger size.

Figure 6.13: Image decomposition of Example 5

Figure 6.14: Quadtree representation of Example 5

Whether the CORN algorithm could still embody its superiority in relation to the traditional method?

In order to answer this question, in the following, the CORN algorithm will be applied to two binary images drawn from the real world [38], p.230. Each image with size 256×256 is shown in Figure 6.15, where each one requires 65536 bits.

Figure 6.15: Two binary images in real world

By applying the CORN algorithm to two test images, in their final quadtree representations: there are 651 black leaf nodes, 757 white leaf nodes and 469 grey nodes concerned with the image “Taiwan”; and for another image “Cloud”, the number of black, white and grey nodes are 4093, 4116 and 2736, respectively. Then, choosing DF-expression (mentioned in Section 3.1.2.2) as the encoding method, the total bits number required for compression performance is 2997 ($651 \times 2 + 757 \times 1 + 469 \times 2 = 2997$) and 17774 ($4093 \times 2 + 4116 \times 1 + 2736 \times 2 = 17774$), respectively.

In comparison with the results obtained by using the traditional method, which are

shown in p.231 of [38], Table 6.2 lists the memory space needed to store the two binary test images before and after processing by using the traditional method and CORN algorithm, followed by data compression rate C_r , which is defined as follows:

$$C_r = \frac{\text{number of bits needed to store original binary image}}{\text{number of bits needed to store compressed binary image}}$$

Table 6.2: Data compression rate comparison

Binary image	Method	No. of bits needed for original image	No. of bits needed for compressed image	C_r
Taiwan	Traditional method	65536	3058	21.43
	CORN algorithm	65536	2997	21.88
Cloud	Traditional method	65536	19024	3.44
	CORN algorithm	65536	17774	3.69

When viewing the Table 6.2, it is observed that the proposed CORN algorithm has quadtree representation and data compression improvement when compared to the traditional method.

As a consequence, experimental results shown in this chapter reveal that for all the test images, no matter whether their contents are more complicated or simpler, the quadtree representation of the CORN algorithm is unexceptionally better than that of the traditional method provided.

Chapter 7

Conclusion

The spatial data representation and compression by using the quadtree data structure is widely studied and applied in image processing applications. To improve the data compression effect by choosing an optimal quadtree root node is a new research ground in this field. At this point, exploring an efficient way to choose quadtree root node is an open problem endowed with promising future.

In this thesis, we have presented the CORN algorithm for finding the root node of the quadtree representation. The worst-case time complexity of this algorithm is $O((m \times n)^2)$, where $m \times n$ is the bounding box dimension of the given image. Experimental results have demonstrated that, the CORN algorithm is generally better than the traditional method when dealing the quadtree representation with the same image. More importantly, when the image to be represented is large and complex, the CORN algorithm may greatly simplify its quadtree representation and obtain a pretty fantastic data compression rate.

Our future research will aim to the following two aspects:

- **Quadtree Representation Optimization**

Associating the concepts of GB node and GW node defined in the forest-based approximation method with the thought of hierarchical approximation method, it is possible for us to define a brand-new approximation method to optimize the quadtree representation generated by using the CORN algorithm.

For brevity, by improving the concept of terms GB and GW, a grey node (internal node) is said to be of type GBO (GWO) node if three of its sons are black (white) nodes or of type GBO (GWO) nodes.

In this way, the RSB ¹ approximation method is defined as follows:

1. Find the outer approximation $OB(1)$ of the quadtree representation yielded by using the CORN algorithm. For each grey node in the $OB(1)$, it contains two kind of nodes: the black nodes of its sons and yellow nodes which correspond to the “Maybe” region of rough set model (i.e., the white nodes of its sons in the original quadtree representation). For example, the left image in Figure 7.1 shows the outer approximation $OB(1)$ of Figure 6.6.
2. Check each grey node in the $OB(1)$: If it is a type of GBO(GWO) node, then it will turn to a black(white) node in the optimization of the finale quadtree representation. Otherwise, it will maintain the same format.
3. Merge the leaf nodes to simplify the quadtree structure and update the quadtree representation.

By using the RSB approximation method, there is great probability of realizing an optimization of the quadtree representation. For example, the right image of Figure

¹Rough-Set-Based - we temporarily prefer to calling it this name.

7.1 is one such optimization according to Figure 6.6.

Indeed, the loss or alteration of spatial data is an inevitable question to this method. However, in essence, when the size of image is large enough, there will be no any impact on the image recognition and image processing, such as in computer vision and graphics engine design fields.

Figure 7.1: Quadtree representation optimization

- **Quadtree-Rough-Set Representation**

The quadtree representation offers a hierarchical decomposition of visual information which can be implemented in a system or a robot. More generally, the quadtree serves well as a representation of a hierarchy of approximation spaces in the sense of Pawlak [26]. Due to the fact that the node building process is triggered by the difference of information and the geometrical structure of the plane, we are able to use the quadtree not only as a coding scheme, but as a simple pyramid representation of image data.

Finding the optimal root node is an essential tool to offer a good representation of the given image data.

Due the hierarchical geometrical structure of the quadtree in NW-NE-SW-SE-blocks, we are enabled to analyze the information of the data structure at any stage of the hierarchy. The information presented at any stage of the hierarchy are rough sets: If we code 1 for “black”, 0 for “white” and ? for “branch” (i.e. “grey”) the rough set hierarchy of the first three levels of Figure 4.7 is given by:

```
Level 0:  ?  
Level 1:  ???1  
Level 2:  ???1 ?010 ?10? 1111  
...
```

The quadtree-rough-set representation offers a quick look at the image at an early stage using only a little amount of information. Successive quadtree representations can be used to represent change.

Once again: As the information can be used at any level of the representation, the change information is organized as hierarchical rough sets as well.

Our next tasks will be to optimize successive quadtree representations (SQTR) – in particular, to investigate the properties of the CORN algorithm – and to develop a rough logic for change in SQTR. The tasks will be focused on the detection of global change (e.g. moving environment) and local change (e.g. moving objects) within successive quadtree representations using the hierarchy of rough set representation. Solving these problems will enable us to formalize machine vision algorithms using a suitable (rough) logic based on quadtrees. As patterns of “approaching” or even “frames to contact” are then (roughly)

definable, a first step towards an ecological perception using rough quadtree representations is feasible.

Bibliography

- [1] A. O. Abanmi, S. A. Alshebeili, and T. H. Alamri. Lossless compression of seismic data. *Franklin Institute*, 343(4):340–351, July 2006.
- [2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. The design and analysis of computer algorithms. *Addison-Wesley, Reading, Mass*, 6(2):77–78, 1974.
- [3] C. H. Ang and H. Samet. A fast quadtree normalization algorithm. *Pattern Recognition Letters*, 15:57–63, 1994.
- [4] F. B. Atalay and D. M. Mount. Pointerless implementation of hierarchical simplicial meshes and efficient neighbor finding in arbitrary dimensions. *In Proc. International Meshing Roundtable (IMR)*, pages 15–26, 2004.
- [5] J. L. Bently and D. F. Stanat. Analysis of range searches in quad trees. *Information Processing Letters*, 13(6):170–173, July 1975.
- [6] F. Dehne, A. G. Ferreira, and A. Rau-chaplin. Parallel processing of pointer based quadtrees on hypercube multiprocessors. 1991.
- [7] I. Düntsch and G. Gediga. Uncertainty measures of rough set prediction. *Artificial Intelligence*, pages 109–137, July 1997.

- [8] C. R. Dyer. Computing the Euler number of an image from its quadtree. *Comput. Gr. Image Process*, 13(3):270–276, July 1980.
- [9] C. R. Dyer. The space efficiency of quadtrees. *Computer Graphics and Image Processing*, 19(4):335–348, 1982.
- [10] R. Finkel and J. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.
- [11] P. Fränti. A fast and efficient compression method for binary images. *Image Communication*, 6(1), March 1994.
- [12] I. Gargantini. An effective way to represent quadtrees. *Commun. ACM*, 25(12):905–910, December 1982.
- [13] N. K. Gautier, S. S. Iyengar, N. B. Lakhani, and M. Manohar. Space and time efficiency of the forest-of-quadtrees representation. *Image and Vision Computing*, 3(2):63–70, May 1985.
- [14] E. Horowitz and S. Sahni. Fundamentals of data structures. *Computer Science Press*, 1976.
- [15] C. Y. Huang and K. L. Chung. Transformations between bincodes and the df-expression. *Comput. Gr.*, 19(4):601–610, July 1995.
- [16] G. M. Hunter. *Efficient computation and data structures for graphics*. PhD. Thesis, Princeton University, 1978.
- [17] G. M. Hunter and K. Steiglitz. Operations on images using quad trees. *IEEE Trans. Pattern Anal. Mach. Intell*, pages 145–153, 1979.

- [18] L. Jones and S. S. Iyengar. Representation of regions as a forest of quadtrees. *In Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, Dallas, Tex.*, pages 57–59, 1981.
- [19] E. Kawaguchi and T. Endo. On a method of binary picture representation and its application to data compression. *IEEE Trans. Pattern Anal. Mach. Intell*, 2(1):27–35, January 1980.
- [20] A. Klinger and C. R. Dyer. Experiments on picture representation using regular decomposition. *Comput. Gr. Image Process*, 5(1):68–105, March 1976.
- [21] K. Knowlton. Progressive transmission of grey-scale and binary pictures by simple, efficient, and lossless encoding schemes. *In Proceeding of the IEEE*, 68(7):885–896, July 1980.
- [22] D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9(1):23–29, March 1977.
- [23] D. A. Lelewer and D. S. Hirschberg. Data compression. *ACM Computing Surveys*, 19:261–296, 1987.
- [24] R. Lumia. A new three-dimensional connected components algorithm. *Comput. Vision Gr. Image Process*, 23(2):207–217, August 1983.
- [25] J. Oliver and M. P. Malumbres. From lossy to lossless wavelet image coding in a tree-based encoder with resolution scalability. *Progress in pattern recognition, image analysis and applications, proceedings*, pages 198–207, July 2006.

- [26] Z. Padwlak. Rough sets. *Internat. J. Comput. Inform. Sci.*, 11:341–356, March 1982.
- [27] S. Ranade, A. Rosenfeld, and H. Samet. Shape approximation using quadtrees. *Pattern Recognition*, 15(1):31–40, 1982.
- [28] A. Rosenfeld and J. L. Pfaltz. Sequential operations in digital image processing. *J. ACM*, pages 471–494, October 1966.
- [29] H. Samet. The quadtree and related hierarchical data structure. *Computing Surveys*, 16(2), June 1984.
- [30] H. Samet. Data structures for quadtree approximation and compression. *Image Processing and Computer Vision*, 28(9), 1985.
- [31] H. Samet. Applications of spatial data structures: computer graphics, image processing, and gis. *Addison-Wesley*, 1990.
- [32] H. Samet. Spatial data structures. *Addison-Wesley*, 1995.
- [33] F. A. Schreiber and R. C. Wolfler. Use of neural networks to estimate the number of nodes of an edge quadtree. *Elsevier Science*, 59(2):61–72, March 1997.
- [34] C. A. Shaffer and H. Samet. Optimal quadtree construction algorithms. *Computer Vision, Graphics, and Image Processing*, 37:402–419, 1987.
- [35] K. R. Sloan and S. L. Tanimoto. Progressive refinement of raster images. *IEEE Trans. Comput.*, 28(11):871–874, November 1979.
- [36] S. Tanimoto and T. Pavlidis. A hierarchical data structure for picture processing. *Comput. Gr. Image Process*, 4(2):104–119, June 1975.

- [37] C. L. Wang, S. C. Wu, Y. K. Chan, and R. F. Chang. Quadtree and statistical model-based lossless binary image compression method. *Imaging Science Journal*, 53(2):95–103, January 2005.
- [38] Y. H. Yang, K. L. Chuang, and Y. H. Tsai. A compact improved quadtree representation with image manipulations. *Image and Vision Computing*, pages 223–231, January 1999.
- [39] X. Yin, I. Düntsch, and G. Gediga. Choosing the root node of a quadtree. *IEEE Granular Computing Conference*, August 2009.