MONOTONIC TREE AND ITS APPLICATION TO MULTIMEDIA INFORMATION RETRIEVAL

By

Yuqing Song

July 18, 2002

A DISSERTATION SUBMITTED TO THE

FACULTY OF THE GRADUATE SCHOOL OF STATE

UNIVERSITY OF NEW YORK AT BUFFALO

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Acknowledgments

I would like to express my appreciation to Professor Aidong Zhang for her advice and encouragement in all these years.

Many thanks to Professor Xin He and Professor Venugopal Govindaraju for their insightful advice and comments on my research work.

I would also like to thank Professor Gultekin Ozsoyoglu for his review and comments.

Abstract

Contour trees have been used in geographic information systems (GIS) and computer imaging to display scalar data. Contours are only defined for continuous functions. For discrete data, a continuous function is first defined as an interpolation of the data. Then a contour tree is defined on this continuous function. In this dissertation, we first introduce a new concept termed monotonic line, which models contour lines of discrete functions. All monotonic lines in a discrete function form a tree, called monotonic tree. As compared with contour trees, monotonic trees avoid the step of interpolation, thus can be computed and manipulated more efficiently. In addition, when used in image processing, monotonic trees retrieve similar structures as contour trees do while reserving the discreteness of image data. In computer imaging, the discreteness of image data is one main factor which makes image processing and understanding so difficult. The discreteness of image data itself is a research topic.

Monotonic trees are used as a hierarchical representation of image structures in content-based multimedia retrieval. Although a variety of techniques have been developed for content-based image retrieval (CBIR), automatic image retrieval by semantics still remains a challenging problem due to the difficulty in object recognition and image understanding. In this dissertation, we present a novel approach to support semantics-based image retrieval on the basis of monotonic trees. The structural elements of an image are modeled as branches (or subtrees) of the monotonic tree. These elements are classified and clustered on the basis of such properties as color, spatial location, coarseness, and shape. Each cluster corresponds to some semantic feature. Following these steps, images can be automatically annotated with category keywords. So high-level (semantics-based) querying and browsing of images can be supported. This scheme is applied to retrieve scenery features from images and locate smooth background in images. Comparisons of experimental results of this approach with conventional techniques using low-level features demonstrate the effectiveness of our approach. In future work, the monotonic tree model will be extended to general semantic categories on both images and videos.

This dissertation has two main contributions. The first contribution is the mathematical theory of monotonic tree, which is the first theory addressing definitions, properties, and computations of contour structures directly on discrete functions. The second contribution is the application of monotonic tree model to semantics-based image retrieval. The success of this model on scenery features and smooth background implies its potential in analyzing

general semantics of both images and videos.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Contour trees have been used in geographic information systems (GIS) and 3D imaging to display scalar data. Contours are only defined for continuous functions. For discrete data, a continuous function is first defined as an interpolation of the data. Then a contour tree is defined on this continuous function. In image processing, the discreteness of image data is one main factor which makes image processing and understanding so difficult. The discreteness of image data itself is a research topic. So we introduced a new concept termed monotonic tree [84], which retrieves similar structures as contour trees do while reserving the discreteness of image data.

Monotonic trees are used as a hierarchical representation of image structures in content-based multimedia retrieval. Although a variety of techniques have been developed for content-based image retrieval (CBIR), automatic image retrieval by semantics still remains a challenging problem due to the difficulty in object recognition and image understanding. In this dissertation, we present a novel approach to support semantics-based image retrieval on the basis of monotonic trees. This approach has been applied to retrieve scenery features from images [83, 81, 82] and locate smooth background in images [85].

1.1 Contour Tree

The concepts of contour trees have been developed by Boyell and Ruston [5] in 1963, Freeman and Morse [19] in 1967, and by many people [60, 75, 102, 95, 6] since then. In geographic information systems (GIS), contour trees are used to display scalar data defined over the plane, or the three-dimensional space. For example, the elevation in the landscape can be modeled by scalar data over the plane and a contour (also called an isoline) is a line where the elevation function assumes the same value. Contour trees are also used in medical imaging to show the scanned data.

In general, for any continuous function f, a *contour* is defined to be a connected component of a level set of f, where the level set of a specified value v is the set $\{x|f(x) = v\}$. For general continuous functions on $\mathbb{R}^d$, contours may be d-dimensional, $(d-1)$-dimensional, or lower dimensional, which makes it complicated to uniformly represent and process contours. So it's required by many contour tree algorithms that the input function is a Morse function [45], whose contours are either hypersurfaces of $(d-1)$-dimension, or isolated points. Informally, a contour tree is a graph that traces the evolution of individual contours, recording the creation, destruction, splitting, and merging of contours [95, 6]. Contours are only defined for continuous functions. For a set of discrete data, a continuous function is first defined as an interpolation of the data. Then the contour tree is defined on this continuous function.

There are two main reasons why we need a new concept to model contour structures directly on discrete functions, besides the fact that the interpolation process adds overhead to the computation time:

- In image processing, the discreteness of image data is one main factor which makes image processing and understanding so difficult. The discreteness of image data itself is a research topic. So a new concept is needed to model the same or similar structures as contour trees do while reserving the discreteness of image data.

- For contour tree algorithms, the input is a simplicial mesh and the function f is a linear interpolation over the mesh. To ensure that the function f is a Morse function, many contour tree algorithms require that the data values at the vertices of the mesh are unique. However, the data produced in real life may not meet this requirement. The only way to handle this problem is to modify the original data slightly, so that the data values at the vertices become unique. This process will add overhead to the computation of the contour tree and add extra contours to the generated tree, making it not accurately reflect the topology of the contour structures of the original data.

1.2 Monotonic Tree

We observe that for any 2D Morse function, a curve is a normal contour with value v iff it's a boundary of the set $\{x \in \mathbb{R}^2 | f(x) > v\}$. This is not true for non-Morse functions. However, the equivalent condition is more general, and can be used to define contours of discontinuous or discrete functions. For a discrete function f and some value v, a boundary b of the set $\{x | f(x) > v\}$ is always closed and has such a property that the function is monotonic from one side of b to the other side, i.e., either of the following is true:

(1) the function f assumes values higher than v at pixels adjacent to the interior side of b, and assumes values no more than v at pixels adjacent to the exterior side of b. In this case, b is called an outward-falling *monotonic contour* or *monotonic line*;

(2) the function assumes values no more than v at pixels adjacent to the interior side of b, and assumes values higher than v at pixels adjacent to the exterior side of b. In this

case, b is called an outward-climbing *monotonic contour* or *monotonic line*.

Figure 1.2.1(a) shows an example of an outward-falling monotonic line. The two types of monotonic lines correspond to positive and negative contour lines in [48]. However, our definition works for discontinuous or discrete functions. To make the boundary of the domain a monotonic line, we extend the input function to the whole plane such that the extended function assumes $-\infty$ out of the domain. It can be proved that monotonic lines don't cross each other, i.e., if $l_1 = \partial X$, $l_2 = \partial Y$ are two monotonic lines, where X, Y are two simply connected regions, then $X \subseteq Y$, $Y \subseteq X$, or $X \bigcap Y = \emptyset$. Based on this property, we can define a parent-child relationship: monotonic line l_1 is the parent of monotonic line l_2, if l_2 is directly enclosed by l_1. Under this parent-child relationship, all monotonic lines of a discrete function form a rooted tree, called *monotonic tree*. For example, all monotonic lines in Figure 1.2.1(b) form a monotonic tree shown in Figure 1.2.1(c).

A monotonic tree can be reduced. A maximal sequence of uniquely enclosing monotonic lines is called a *monotonic slope*. All monotonic slopes form a *topological monotonic tree*. See Figure 1.2.1(d). The concepts of monotonic tree and topological monotonic tree are only defined for two-dimensional discrete functions. In future work, they will be extended to all dimensions.

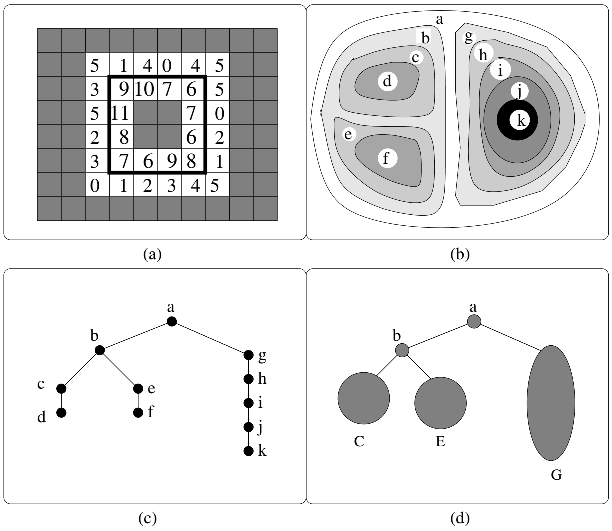


Figure 1.2.1: (a) An outward-falling monotonic line (the solid line in the figure), (b) a set of monotonic lines, (c) the monotonic tree, (d) the topological monotonic tree.

1.3 Semantics-based Image Retrieval

With the advance of multimedia technology and growth of image collections, there is an urgent need to build efficient and effective image retrieval systems. Content-based image retrieval (CBIR) offers a promising technology to address this need, and a variety of CBIR techniques have been developed. Image retrieval can be based on low-level visual features (such as color [79, 51], texture [41, 72], and shape [71, 94, 68]), high-level semantics [18, 97], or both [107]. Various image-querying systems, including QBIC [17], VisualSeek [79], PhotoBook [53], Netra [39], and Virage [3], have been built, using low-level features for general or specific image-retrieval tasks.

However, retrieving images via low-level features has proven unsatisfactory. Effective and precise image retrieval by semantics still remains an open problem due to the difficulty in object recognition and image understanding. Theories and approaches for querying and browsing images based on image semantics become increasingly critical as web-based information retrieval grows in popularity. In recognizing the existing problems in the CBIR field, we believe that research efforts are needed to bridge the gap between the high-level semantics of interest to users and the low-level features which can be efficiently extracted.

The semantics of images can be classified into two levels:

- *Local semantic level*: Local semantic features describe the presence of individual objects in images. Two examples of queries by local semantic features are "find pictures with a bridge" (object of a given type) and "find pictures with sky and trees" (combination of objects).

- *Thematic level (or global semantic level)*: Thematic features describe the global meanings or topics of images. Two examples of queries at this level are "find pictures of a Chinese garden" and "find pictures of an earthquake."

The thematic features of an image are based on all objects in the image along with their spatial relationships. High-level reasoning is needed to derive the global meaning of all objects in the scene and to determine the topic of the image.

In this dissertation, we will present a novel approach to support semantics-based image retrieval. This approach is based on the monotonic tree model. The structural elements of an image are modeled as branches (or subtrees) of the monotonic tree of the image. These structural elements are classified and clustered on the basis of such properties as color, spatial location, coarseness, and shape. Each cluster corresponds to some semantic feature. Following these steps, images can be automatically annotated with category keywords, thus facilitating high-level (semantics-based) image querying and browsing.

1.4 Image Background and Scenery Features

The approach described above has been applied to retrieve scenery features from images [83, 81, 82] and locate smooth background in images [85]. In outdoor pictures, scenery features such as sky, trees, ground, and water are common types of background objects. Locating and interpreting background objects are essential to object recognition and image understanding. The reasons are given below.

Among all objects in a natural image, it's quite common that a foreground object (or a group of foreground objects) indicates the theme of the image, while the background objects provide contextual and complementary information. Needless to say, successfully recognizing background objects in an image is a necessary step in determining the theme of the image. In addition, the contextual information provided by background objects helps and in many cases is necessary in recognizing foreground objects in two aspects:

- it can narrow down the possible types of foreground objects and make foreground object recognition more efficient; and
- it can be used to resolve the ambiguity in recognizing foreground objects.

Thus finding effective ways to recognize common background objects will be a milestone in solving the problem of image understanding. Natural images fall into two categories: outdoor and indoor images. These two categories have totally different background object types. In outdoor pictures, scenery features such as sky, trees, ground, and water are common types of background objects. Scenery images often fall into a small number of typical patterns, such as those shown in Figure 1.4.1. These typical patterns greatly simplify scene interpretation for scenery images.

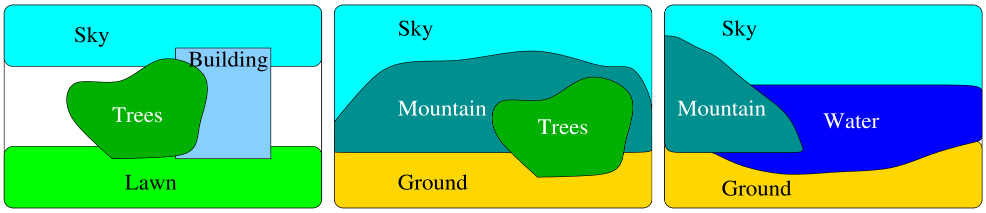


Figure 1.4.1: Some patterns of scenery images.

1.5 Organization of This Dissertation

The rest of this dissertation is organized as follows. Chapter 2 gives a brief literature review on contour trees, and then introduces definitions, computations, and applications of contour trees. Chapter 3 summarizes existing approaches for image retrieval and points out their problems. Chapters 4 to 7 present the monotonic tree theory. Chapter 4 defines the basic concepts of the theory: *monotonic line*, *monotonic tree*, and *topological monotonic tree*. Chapter 5 presents several theorems, describing basic properties of monotonic trees. In Chapter 6, a top-down algorithm is presented for computing monotonic trees. The algorithm takes time linear to image sizes. Chapter 7 introduces the concept *reduced monotonic tree*, which has less nodes than the monotonic tree or topological monotonic tree. Chapter 8 gives an overview to the *SceneryAnalyzer* system, which is a semantics-based image retrieval system based on monotonic trees. Chapter 9 describes the feature extraction component of the *SceneryAnalyzer*. Chapter 10 is devoted to the experiments. Comprehensive experiments were conducted to demonstrate the effectiveness of the monotonic tree model on image retrieval. In Chapter 11, contributions of this dissertation are summarized and future work is discussed.

Chapter 2

Introduction to Contour Tree

Contours have been used in meteorology, geographic information systems (GIS), and medical imaging to display scalar data. For a two-dimensional continuous function, a contour (or an isoline) is a curve where the function assumes a constant value. For example, in the weather map shown in Figure 2.0.1, the isolines in white are used to display the pressure function. Informally, a contour tree is a graph that traces the evolution of individual contours, recording the creation, destruction, splitting, and merging of contours [95, 6]. In this chapter, we will first give a brief literature review, and then introduce definitions, computations, and applications of contour trees.

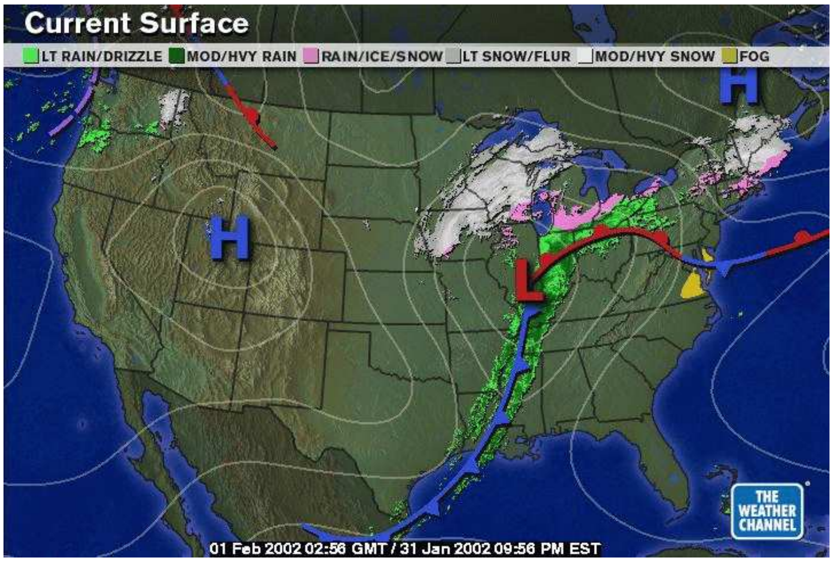


Figure 2.0.1: A weather map.

2.1 Literature Review

The concepts of contour trees have been developed by Boyell and Ruston [5] in 1963, Freeman and Morse [19] in 1967, and by many people [60, 75, 102, 95, 6] since then. Boyell and Reston [5] first introduced a graph representation of adjacency relationships among the contour lines of a two-dimensional function, where the contours are mapped as edges and interstitial regions as nodes. It provided a visual tool to perceive the relationship among contour lines. Freeman and Morse [19] used a slightly different way to represent the topology of contour maps, where contour lines are represented as nodes and inter-contour regions as edges. These two representation schemes are equivalent.

Contour trees were first used in three and higher dimensional data sets only recently. In the approach proposed by M. van Kreveld et al. [102], continuous functions are represented by two or higher dimensional meshes. The contours are isosurfaces of a specified value. To find such contours, a seed set can be used for the starting points from which the traversal of the contours can start. Simple algorithms were given to compute contour trees in regular and irregular meshes that requires $O(n \log n)$ time in 2D for meshes with n elements, and $O(N^2)$ time in higher dimensions where N is the number of simplices. Tarasov and Vyalyi [95] gave an $O(N \log N)$ algorithm to compute contour trees for 3D data sets. Carr et al. [6] improved Tarasov and Vyalyi's method and obtained an $O(n \log n + N\alpha(N))$ algorithm for constructing the augmented contour tree of a real-valued function interpolated over a simplicial mesh in any number of dimensions, where n is the number of vertices in the mesh, N is the number of simplices, and $\alpha(\bullet)$ is the inverse Ackermann function. The augmented contour tree is produced by augmenting the contour tree with the contours passing the vertices of the mesh which are not in the contour tree.

2.2 Definition of Contour Tree

The concepts of contour trees are related to Morse theory [45]. Morse theory studies the changes in the topology of the level sets [1] of a function f as the parameter is varied. These topological changes occur at the critical points of f, where the derivative vanishes. A

[1] For function f, the level set of a specified value v is the set $\{x | f(x) = v\}$.

function is called a *Morse function* if its critical points are isolated.

In general, for any continuous function f, a *contour* is defined to be a connected component of a level set of f. For general continuous functions on $\mathbb{R}^d$, contours may be d-dimensional, $(d-1)$-dimensional, or lower dimensional. For Morse functions, contours are either hypersurfaces of $(d-1)$-dimension, or isolated points. These isolated points are local minima or maxima.

For contour tree algorithms, the input is a simplicial mesh and the function f is a linear interpolation over the mesh. To ensure that the function f is a Morse function, many algorithms require that the data values at the vertices of the mesh are unique. In fact, the uniqueness of the data values at the vertices not only ensures that f is a Morse function but also that the critical points of f occur at the vertices of the mesh [4]. This make it possible to compute contour trees using a combinatorial approach. The contour tree of a simplicial mesh is defined this way: each contour that passes a critical point is a node; and for each region bounded by two contours, an arc is added to connect the corresponding nodes. In some papers [102, 6], the contour tree is extended or augmented to include all contours of the vertices of the mesh.

An example is given in Figure 2.2.1, which is taken from [102]. In this example, the input is a 2D triangular mesh, which is shown by the solid lines of the left graph in Figure 2.2.1. The contours of all critical vertices are shown in dashed lines of the left graph. These contours subdivide the domain into regions, where each region is bounded by two contours. To construct the contour tree, each contour in the subdivision corresponds to a node in the graph, and two nodes are connected (the direction of the arc is from higher value to lower value) if there is a region bounded by their corresponding contours.

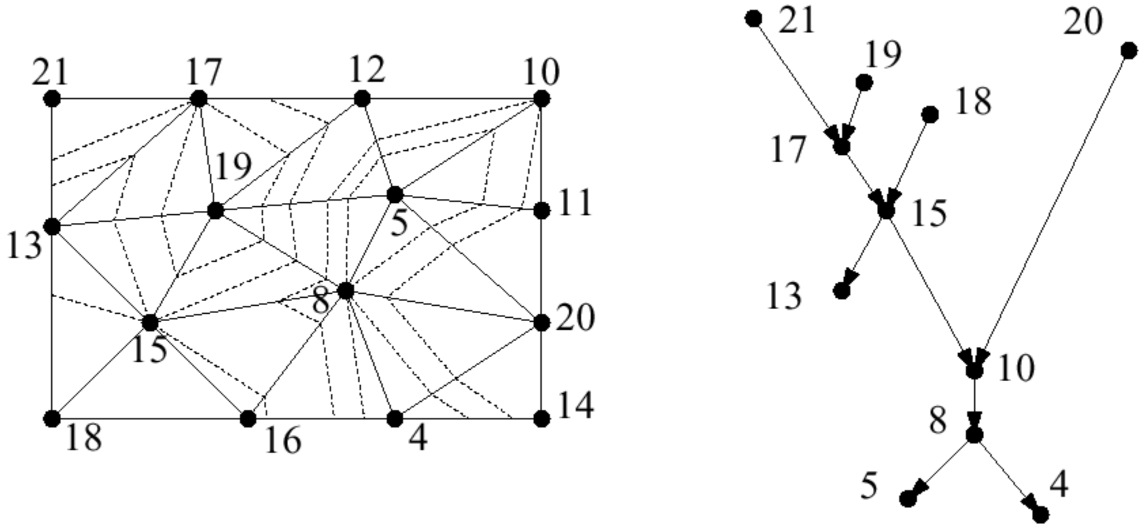


Figure 2.2.1: An example of 2D contour tree.

2.3 Computation of Contour Tree

A straight-forward way to compute the contour tree of a simplicial mesh is to test locally whether a vertex is a saddle. The contour passing a saddle vertex divides the neighborhood of the saddle into three or more regions. The contour of a saddle can be identified by a traversal starting with the saddle vertex. In this way the contours of critical points (local minima, maxima, and saddle points) can be identified. These contours subdivide the domain into regions. The adjacency relationships among the contours can be found by checking these regions. The time complexity of this approach is $\Omega(n^2)$ where n is the number of vertices, because the total combinatorial complexity of all contours through saddles may be quadratic [102].

M. van Kreveld et al. [102] introduced the first efficient algorithm for generating contour trees. The contour tree of a mesh is generated by a sweep of function value from high to low. During the sweep, the contours of these values are traced. The contours can change structurally only at the critical points. Whenever a critical point is encountered, its type (a local minimum, maximum, or saddle point) is determined and the contour is created, destroyed, split, or merged according to the type of the critical point. The contour tree grows during the sweep, and is constructed when the sweep is over. The time complexity of this algorithm is $O(n \log n)$ for 2D meshes with n elements, and $O(N^2)$ time in higher dimensions where N is the number of simplices.

Tarasov and Vyalyi [95] extended the above algorithm and obtained an $O(N \log N)$ algorithm for 3D data sets. The contour tree is constructed by three sweeps. During the sweeps, the contours are traced. In the first sweep, joins of contours are identified. In the second sweep, the splits are identified. The last sweep combines the results of the first two sweeps.

The algorithm by Carr et al. [6] begins with Tarasov and Vyalyi's idea of three sweeps through the data, but makes the following simplifications and improvements. The first two sweeps do not maintain level sets, but construct “join” and “split” trees, which store partial topological information about the data. Then a simple merging procedure is applied to obtain the contour tree. The resulting algorithm extends to all dimensions and runs in time $O(n \log n + N\alpha(N))$ for data sets in any number of dimensions.

2.4 Application of Contour Tree

Contours and contour trees have been used in GIS and computer cartography [19, 42, 74, 30, 92, 34], image coding [100, 99], medical imaging [37, 11, 25, 35], X-ray crystallography [21, 10, 7], and fluid dynamics [35].

Liu and Ramirez [34] presented a method for vectorizing and labeling very large raster hypsographic map images. By extending the contour tree to a contour graph, open contour lines and carrying contour lines can be handled. Contour graph is constructed and processed efficiently by introducing index contour graph. By using a sequential extraction algorithm for vectorization, attributes for contour lines, such as line width, and elevation tags, are extracted efficiently. Mark [42] applied the contour tree concept in computer cartography, particular in contour identification and surface generalization. He pointed out the potential of the graph structure in rebuilding the topology of contour lines.

Turner [100] presented a process for converting a raster image into a discrete set of plateaus or contours. A contour coder was developed which fully describes these contours via their boundary descriptions in a compact and computationally efficient manner, and is the basis for an image compression method. A contour is fully defined with three distinct pieces, or streams, of information: start location of the contour, which is the top left hand corner of the contour; height value of the boundary; and the boundary description of the contour. Once the contours have been isolated, using contourization, they can then be separated into the three streams of information stated above. Preceding the streams of information is a header containing the horizontal and vertical resolution of the complete image. After the header, the three streams of contour information are encoded using three stream encoders. The decoding procedure is the reverse of the encoding procedure.

In disciplines such as medical imaging, X-ray crystallography, and fluid dynamics, scientific simulations produce data by a sampling process in the form of a scalar function defined at discrete points in 3D space. Many techniques for visualizing such data exist, including volume rendering [43, 73] and isosurface extraction [37, 11, 25]. Volume rendering is a technique for displaying an entire data volume such that the volume consists of semi-transparent material and it's directly projected onto the screen using ray casting. In isosurface based methods, data are visualized by drawing level sets or isosurfaces. Contour trees and augmented contour trees can help in choosing threshold values for interactive exploration of such data. As a data analysis tool, contour trees trace the evolution of isosurfaces as the threshold parameter changes and can be used to compute seed sets for all contours, to select the threshold values, and to support interactive visualization.

Chapter 3

Introduction to Image Retrieval

As the information age is becoming a full reality, it brings with it a rapidly growing demand for technologies to store, retrieve, and manipulate huge amount of data, especially multimedia data. Today, digital image collections are increasingly pervasive in all aspects of life. To make full use of them, we need effective and efficient ways to browse and retrieve images. Since 1970s, image retrieval has been an active research area, attracting researchers from two major research communities, database management and computer vision. Many techniques have been developed. In this chapter, we will give a brief introduction to existing approaches for image retrieval and point out their problems. Comprehensive surveys can be found in [8, 56, 63].

3.1 Text-based Retrieval

Long before images could be digitized, image collections were maintained by librarians, curators, and archivists through text descriptors or classification codes. For the management of an image collection, a common technique used then was: each image is assigned with descriptive metadata in the form of keywords, subject headings, or classification codes when it is first added to the collection; these descriptors are used as retrieval keys at search time. The text-based methods are still used by many systems. There are three schemes for image annotation and indexing: uncontrolled-vocabulary, controlled-vocabulary, and systematic classification [101].

3.1.1 Uncontrolled Vocabulary

In the uncontrolled-vocabulary scheme, each image is annotated with keywords or text describing its content, caption, or background information such as time, place, or photographer. This scheme is called *uncontrolled-vocabulary* because the keywords or text to describe images are not limited in this scheme. J. van den Berg [101] classified the use of uncontrolled vocabulary into two main methods: the use of free text, and of additional keywords. When using free-text method, human catalogers annotate images in natural languages; when using additional-keyword method, images are associated with a limited selection of keywords picked by catalogers.

3.1.2 Controlled Vocabulary

In the controlled-vocabulary scheme, each image is characterized and indexed by terms from a specified or authorized list, called a controlled vocabulary or thesaurus. The Art and Architecture Thesaurus (AAT) [1] is an example of image indexing systems using this scheme. It was developed as part of the Getty Trust's Art History Information Program (AHIP) in the United States and is now maintained by the Vocabulary Program of the Getty Research Institute in Los Angeles. The AAT is a structured vocabulary containing around 125,000 terms and other information. It currently provides access to thirty-three hierarchical categories of image description using seven broad facets: Associated Concepts, Physical Attributes, Styles and Periods, Agents, Activities, Materials, and Objects. The thesaurus keeps growing. An on-line browser for the AAT is available at "http://www.getty.edu/research/tools/vocabulary/aat/index.html".

[1] The following information about AAT is from "http://www.getty.edu/research/tools/vocabulary/aat/about.html".

3.1.3 Systematic Classification

The essence of systematic classification is that rather than keywords or text descriptors, classification codes are used to describe image content, providing a greater degree of language independence and showing concept hierarchies more clearly. ICONCLASS, [2] an iconographic classification system, is an example of this genre. ICONCLASS is a subject specific international classification system for iconographic research and the documentation of images. It was originally devised in 1950s by Henri van de Waal, Professor of Art History at the University of Leiden, and completed by his staff. ICONCLASS contains at present about 28,000 definitions of objects, persons, events, situations and abstract ideas that can be the subject of an image. ICONCLASS organizes iconography into ten main divisions in which the definitions are ordered hierarchically. The system is on-line available at “http://www.iconclass.nl/”.

3.1.4 Problems of Text-based Retrieval

Due to lack of techniques for automatic image annotation, text-based approaches suffer from two main drawbacks, especially when the size of image collections is very large. First, it’s laborious and tedious to manually annotate images in large collections. Second, human annotation is subjective and can’t fully characterize the rich content of images. To

[2] The following information about ICONCLASS is from “http://www.iconclass.nl/”.

overcome these problems and improve retrieval performance, content-based retrieval was proposed in the early 1990s.

3.2 Content-based Retrieval

In contrast to earlier text-based approaches for image indexing and retrieval, recent efforts focus on the research of retrieving images based on their contents. Content-based image retrieval (CBIR) is a technology for the retrieval of images on the basis of features automatically extracted from the images themselves. A variety of techniques have been developed. In particular, content-based image retrieval using low-level features (such as color [90, 79, 51], texture [41, 76, 72, 89, 40], and shape [91, 44, 23, 46, 47, 24, 16, 28, 103, 104, 71, 94, 68]) extracted from the images has been well studied. Various image querying systems including QBIC [17], VisualSeek [79], PhotoBook [53] and Virage [3] have been built based on the low-level features for general or specific image retrieval tasks.

A typical CBIR system assumes a large image database. Visual features such as color, texture, and shape are extracted and stored for the images in the database. To facilitate fast search, effective indexing techniques such as dimension reduction and multi-dimensional indexing techniques are often used. To retrieve images, the user submits a query through the query interface. The images in the database will then be compared with the query via their feature vectors by some similarity measurements, and the qualifying images are returned to the user. The output images are usually sorted by the likelihood of answering the query correctly. The user may then refine the query by providing feedback information.

3.2.1 Content-based Techniques

Feature extraction is the basis of content-based image retrieval. Next, visual features, such as color, texture, and shape, and corresponding extraction techniques will be reviewed. For other techniques such as those on high-dimensional indexing and image segmentation, readers may refer to [63].

- **Color** is one of the most widely used visual features in content-based image retrieval. It is relatively robust and simple to represent and handle. Swain and Ballard [90] are the pioneers using color to manipulate the contents of images. They developed the concepts of color histogram and the distance measures such as histogram intersection and histogram back-projection for similar image matching. They proposed a color indexing method that compares the histogram of an image with those of an object stored in a database in black white, red green and blue yellow spaces. Since then, color histogram has been widely used for CBIR. Besides the color histogram, several other color feature representations have been applied in image retrieval, including color moments [88] and color sets [77].

- **Texture** is loosely defined by Tamura et al. [93] as "what constitutes a macroscopic region. Its structure is simply attributed to the repetitive patterns in which elements or primitives are arranged according to a placement rule". Tuceryan and Jain [98] classified common texture analysis techniques into four categories: statistical, geometrical, model based, and signal processing models. Tamura et al. [93] modeled visual texture properties which are important in psychology studies. The six visual texture properties were coarseness, contrast, directionality, linelikeness, regularity, and roughness. In the early 1990s, many researchers began to study the use of the wavelet transform in texture representation. Famous wavelet functions, such as Haar, Daubechies, and Gabor, are used to design filters for different applications [78, 41, 69]. Random field modeling is a recent model. Under this model, a textural image is considered as a 2D random field, and is decomposed into three mutually orthogonal random fields: a harmonic field, an evanescent field, and an indeterministic field [33]. These fields represent periodicity, directionality, and randomness, respectively.

- **Shape** representation can be classified into two categories: boundary-based and region-based. The former uses only the outer boundary of the shape while the latter uses the entire region. Shape representation is required by some applications to be invariant to translation, rotation, and scaling. Queries based on shapes are generally made by selecting an example image or by having the user sketch a shape. Shapes are matched on the basis of shape invariants, which can be either spatial domain invariants or frequency domain invariants. Spatial domain invariants include geometric [64] and algebraic invariants [96]. Frequency domain invariants include Fourier descriptors [55, 66] and wavelet descriptors [9]. A very good survey of shape analysis techniques can be found in [36].

- **Color layout** is a combination of color and spatial features. Indexing using colour histograms has significant limitations due to lack of spatial information [87]. To overcome this problem, new methods were developed to integrate spatial information with color histograms. Faloutsos et al. [16] introduced an approach to divide the whole image into subblocks and extract color features from each of the subblocks. Smith and Chang [77] applied back-projection on binary color sets to extract color regions which carry the contents of an image, where a binary color set is a selection of a subset of colors. Pass and Zabih [50] presented a histogram refinement method using the colour coherence vectors (CCV), which partitions pixels based on spatial coherence. Image retrieval systems such as QBIC [49], VisualSEEk [79], and Netra [38] have also tried to use color layout features for improving the performance of image retrieval.

3.2.2 Problems of Content-based Retrieval

Although shape, color, and texture are undoubtedly important visual features for image representation, humans tend to use high-level concepts in everyday life. It's still an open problem to automatically retrieve general semantic information from images/videos. A quick review of computer vision research over the last few years points to this difficulty, though encouraging progress has been made in constrained applications, such as recognition of human faces and finger prints. It's been recognized that there is a gap between the low-level features that can be computationally acquired and the high-level semantics that is needed by users for various tasks. Some recent efforts have been made to narrow down this semantic gap.

3.3 Bridging the Semantic Gap

There are several directions for researches to narrow down the semantic gap, including text from context [67], integrating textual and visual features [14, 70, 20], relevance feedback [62, 65], and semantics from low level features [26, 29].

3.3.1 Text Retrieval

In the work by Sable et al. [67], textural information is extracted from associated text sources such as captions and surrounding text in the document in which the image is embedded. The textual information from embedding documents or other sources is integrated by some researchers [14, 70, 20] with the visual features extracted from images to improve retrieval performance. The biggest hurdle facing these approaches is that many images are without accompanying text. Their applications are limited to text-rich environments, such as the world-wide web.

3.3.2 Relevance Feedback

In an image retrieval system with a user-friendly query interface, the user is often allowed to provide his/her evaluation of a current retrieval result to the system. The feedback information is used to tune computational similarity measurements to improve retrieval performance. The first difficulty with relevance feedback methods is that very few users are willing to go through more than four iterations of feedback in hopes of retrieving the best results [27]. So feedback methods require many iterations are not practically useful. The second difficulty comes after the first: with a few iterations of feedback, the system can only get a relatively small number of training samples during the user interaction. The number of samples is often smaller than the dimension of the feature space, which makes it a very challenging problem, if solvable, to find the target images by a learning process with the information fed-back by the user.

3.3.3 Semantics Retrieval

Klauck [26] proposed a graph grammar based method to group low-level image features into meaningful image objects. Firstly, the image is segmented and the neighborhood relations among the segments are computed. Secondly, the object recognition is realized by graph operations triggered by a graph grammar. The graph grammar presents the compiled taxonomy, which reflects the domain knowledge. This grammar consists of three layers: segments as terminals (first level), primitive objects as nonterminals (second level), and complex object as goals (third level). Terminal nodes are represented by the inputs of the color, texture, and contour modules. The goals are recognized image objects and various scene descriptions. The ImageMiner system [29] aims to extend similar techniques into the video domain. All such techniques are based on the idea of locating and recognizing objects by grouping low-level features layer by layer up to get high-level semantics.

There are two main difficulties with these techniques. The first is that these techniques require reliable low-level features (such as color, texture, and segments), which can't be provided by the existing technology yet. For any given low-level feature, there exist many representations. Each of these representations has its advantages and disadvantages. The second difficulty is that the same object may have different appearance in an image if the lighting condition is changed, the image is taken from a different perspective, or the object is occluded by another object in the image. Unless these issues are properly addressed, these semantic image retrieval techniques can only have limited achievements. As Eakins and Graham pointed out in [15], even though research into semantic image retrieval techniques is beginning to gather momentum, particularly in restricted domains, it will take a considerable time before such research supersedes manual indexing and finds its way into commercially-available products.

In this dissertation, a novel approach will be presented to support semantics-based image retrieval on the basis of monotonic trees. As a hierarchical representation for images, the monotonic tree model can uniformly handle low-level features such as color, texture, and shape. So it's a powerful tool to deal with the first difficulty of existing semantic image retrieval techniques, as we mentioned above. In future work, the model will be integrated with existing computer vision and machine learning techniques to extract general semantic information from images and videos.

Chapter 4

Definition of Monotonic Tree

In this chapter, we define the basic concepts of our theory: *monotonic line*, *monotonic tree*, and *topological monotonic tree*. These concepts are defined on the hexagonal grid, which avoids the contiguity paradoxes of the square grid [52].

For the hexagonal grid, we use notations in [22] with modifications. We introduce the basic concepts of the hexagonal grid, including pixel, edgel, region, border, etc. Our emphasis is on the boundaries of regions. We prove that a region on the hexagonal grid is simply-connected if and only if its border is just a connected boundary, which is a boundless list. This gives us an efficient way to traverse and represent the boundary of a simply-connected region. Based on these concepts, we define the *monotonic line* and prove that monotonic lines don't cross each other. Then we define the *monotonic tree* and *topological monotonic tree*.

4.1 Preliminary on Hexagonal Grid

In our theoretical discussion, we choose the hexagonal grid for the digital plane. We use notations in [22] with modifications. The theory here is self-complete and readers don't need to refer to elsewhere to understand it. We first explain the advantages of the hexagonal grid over the square grid and introduce an efficient way to transform between these two grids. We then define the basic concepts of the hexagonal grid. These basic concepts include pixel, edgel, region, border, etc. We also discuss the boundaries [1] of regions. We model the counter-clockwise direction along the border of a region by a "next" function. We also define the simple connection on the hexagonal grid. Then we prove that a region on the hexagonal grid is simply-connected if and only if its border is just a connected boundary, which is a boundless list. This gives us a beautiful way to traverse and represent the border of a simply-connected region.

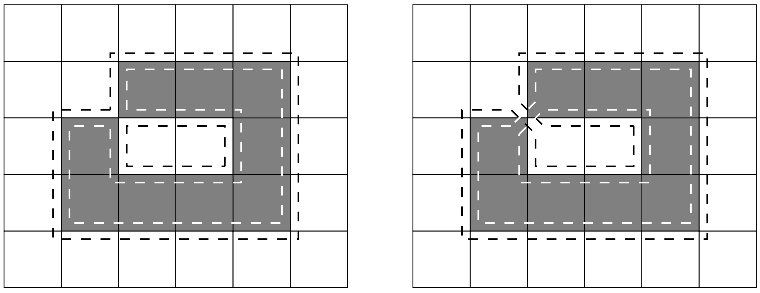
[1] Boundaries and borders are often used in literature for the same thing. In this dissertation, they are different. The border of a region is the set of edgels shared by this region and its complement; a boundary (or connected boundary) is a connected component of the border.

4.1.1 Hexagonal Grid Versus Square Grid

Before the basic concepts of the hexagonal grid are introduced, we explain why we choose the hexagonal grid, instead of the square grid, for the definitions of monotonic trees, and give an easy way to transform between these two grids.

The hexagonal grid has such a property that the border of any region on the grid is isomorphic to the border of its complement.[2] The situation is different on the square grid. For example, on the square grid with 4-adjacency, the set of shaded pixels in Figure 4.1.1(a) has a connected border (shown by the white dashed line), while the border (shown by the black dashed lines) of its complement has two connected components. The same problem occurs if the square grid is associated with 8-adjacency. See Figure 4.1.1(b). One possible solution to the border paradox is to treat objects using 4-adjacency and background using 8-adjacency (or vice versa). However, in case multiple objects exist simultaneously on the square grid, we have difficulty choosing adjacencies for these objects. The border paradox of the square grid is the primary reason why we avoid its use for our theoretical discussions about monotonic trees. The border paradox is one case of the contiguity paradoxes of the square grid [52]. More advantages of the hexagonal grid over the square one can be found in [80].

[2] This property is implied by Lemma 4.1.3 (1) and (2).



(a) (b)

Figure 4.1.1: Border paradox of the square grid with (a) 4-adjacency, or (b) 8-adjacency.

Since images are usually represented in the square grid, we need an efficient way to transform images between the square and hexagonal grids. The basic idea of the transformation is very simple. Each image can be considered as a step function on the Euclidean plane which is constant over the squares of the pixels. Each hexagonal pixel is a hexagon in the plane. We compute the values of the hexagonal pixels by averaging the step function over the corresponding hexagons. The same idea can be applied to transform any discrete data between any two grids. Because this transformation is just an averaging operation at the pixel level, it doesn't change the images visually, though it may blur sharp boundaries slightly. It's also very efficient, taking time linear to the image size.

Formally, let $I_s = (f_s, \Omega_s)$ be a gray image on the square grid $\mathbb{Z}^2$, where Ω_s is the image domain, which is a rectangle on the square grid, and f_s is a function from Ω_s to $\{0, 1, ..., depth-1\}$. The depth for gray images is usually 256. Each pixel $p = (i, j)$ in Ω_s represents a square $[i-\frac{1}{2}, i+\frac{1}{2}] \times [j-\frac{1}{2}, j+\frac{1}{2}]$ on the Euclidean plane $\mathbb{R}^2$. Denote the square as $Square(p)$. Each pixel $q = (x, y)$ on the hexagonal grid represents a hexagon on $\mathbb{R}^2$

whose vertices are $(x+\frac{1}{2}, y+\frac{\sqrt{3}}{6})$, $(x, y+\frac{\sqrt{3}}{3})$, $(x-\frac{1}{2}, y+\frac{\sqrt{3}}{6})$, $(x-\frac{1}{2}, y-\frac{\sqrt{3}}{6})$, $(x, y-\frac{\sqrt{3}}{3})$, and $(x+\frac{1}{2}, y-\frac{\sqrt{3}}{6})$. See Figure 4.1.2 (b). Denote the hexagon as $Hexagon(q)$. We define Ω_h by

$$\Omega_h = \{q \in \mathbf{V}_h | \Sigma_{p\in\Omega_s} Area(Square(p) \bigcap Hexagon(q)) > \frac{Area(Hexagon(q))}{2}\},$$

where $\mathbf{V}_h$ is the set of pixels on the hexagonal grid. We define the function f_h on Ω_h by

$$f_h(q) = \frac{\Sigma_{p\in\Omega_s} f_s(p) \times Area(Square(p) \bigcap Hexagon(q))}{\Sigma_{p\in\Omega_s} Area(Square(p) \bigcap Hexagon(q))},$$

where $Area(\bullet)$ denotes the area of a region on $\mathbb{R}^2$. It's easy to verify that for any $q \in \Omega_h$, $f_h(q) \in [0, depth-1]$. The value of $f_h(q)$ is truncated to an integer. So f_h is a function from Ω_h to $\{0, 1, ..., depth-1\}$. Then $I_h = (f_h, \Omega_h)$ is the transformed image on the hexagonal grid. See Figure 4.1.2. It's easy to see that Ω_h is a simply connected region, i.e., a connected region with no holes inside.

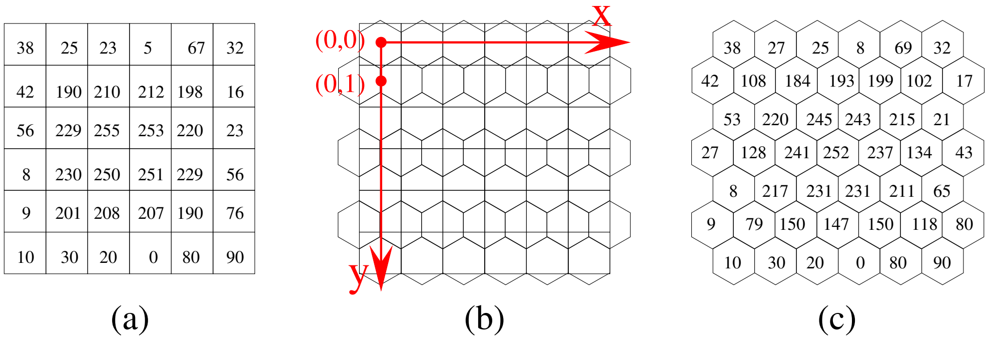


Figure 4.1.2: (a) $I_s = (f_s, \Omega_s)$ on the square grid, where the number in each square indicates the value of f_s at that pixel; (b) Ω_s and Ω_h on the Euclidean plane; and (c) $I_h = (f_h, \Omega_h)$ on the hexagonal grid.

4.1.2 Basic Concepts

In this sub-section, we introduce the basic concepts of the hexagonal grid. The digital plane with the hexagonal grid is modeled by two sets, the pixel set and the edge-adjacency set. The pixel set is the set of hexagons that tile the plane. The edge-adjacency set is a symmetric binary relation on the pixel set, which defines the neighboring relationship of the pixels. Each element of the edge-adjacency set is called an edgel (short for edge element).

A digital curve is modeled as a set of edgels. For an edgel (p, q) in a digital curve, its dual edgel (q, p) need not be in the curve. This gives us a way to model the orientation of a curve so that its “inside” and “outside” can be distinguished. The immediate interior of a digital curve is the set of pixels adjacent to the curve from inside; the immediate exterior is the set of pixels adjacent to the curve from outside.

A set of hexagonal pixels is called a region or digital region. The border of a region is defined as the set of edgels whose immediate interior pixels are in the region and immediate exterior pixels are not. A connected region is a region such that any two pixels in the region are connected by a path in the region. A component (or connected component) of a region is a maximal connected subset. At the end of this sub-section, some properties of connected components are given in two lemmas.

The formal definitions and discussions of these concepts are given below. We first define the hexagonal grid.

Definition 4.1.1 *The digital plane with the hexagonal grid is a pair* $(\boldsymbol{V}_h, \Pi_h)$*, where* $\boldsymbol{V}_h$ *is the set of pixels:*

$$\boldsymbol{V}_h = \{h_1(1,0) + h_2(-0.5, \sqrt{0.75}) | h_1, h_2 \in \mathbb{Z}\}, \tag{4.1.1}$$

and Π_h *is the edge-adjacency shown in Figure 4.1.3(a).* Π_h *is a symmetric binary relation on* $\boldsymbol{V}_h$ *such that* $\boldsymbol{V}_h$ *is* Π_h*-connected. Formally,*

$$\Pi_h = \{(p,q) | p, q \in \boldsymbol{V}_h, and \ \parallel p - q \parallel = 1\}. \tag{4.1.2}$$

Elements of $\boldsymbol{V}_h$ *are called* ***pixels****. Elements of* Π_h *are called* ***edgels*** *(short for edge elements). For any edgel* (c,d)*,* (d,c) *is called its* ***dual edgel****, denoted as* $(c,d)^{-1}$.

We now model a digital curve as a set of edgels.

Definition 4.1.2 *Any nonempty subset of* Π_h *is called an* ***edgel set*** [3] *of* $(\boldsymbol{V}_h, \Pi_h)$*. For any edgel set* S*, its* ***dual edgel set*** *is defined as* $S^{-1} = \{s^{-1} | s \in S\}$.

Note that dual edgels (c,d) and (d,c) are considered two different edgels. So if (c,d) is in an edgel set, (d,c) need not be. This is an essential feature of Herman's theory [22]. This

[3] In Herman's book [22], $(\mathbf{V}, \Pi)$ can be any grid of any dimension. Thus any subset of Π is called a surface there. A surface of a two-dimensional grid models a discrete curve.

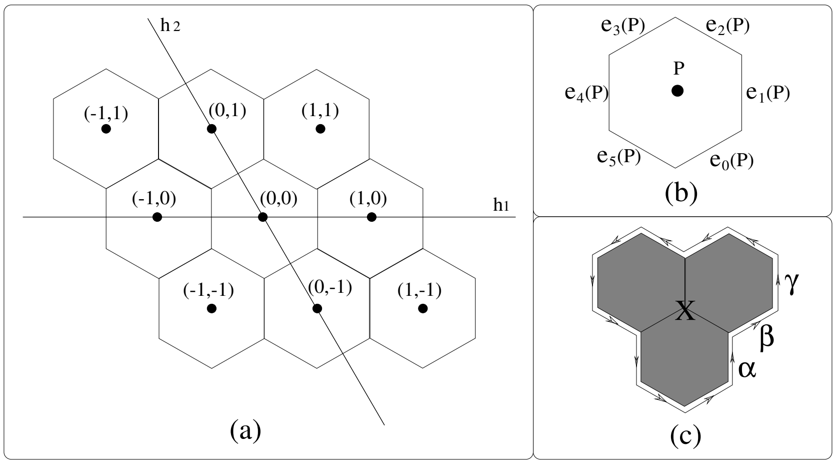


Figure 4.1.3: (a) Hexagonal grid, (b) six edgels of pixel P, and (c) the connected boundary of a digital region.

theory is supposed to deal with edgels oriented so that their "insides" and "outsides" can be distinguished.

Definition 4.1.3 *For any edgel (c, d), c is called its immediate interior pixel, denoted as $IIP(c, d)$, and d is called its immediate exterior pixel, denoted as $IEP(c, d)$.*

For any edgel set S in $(\boldsymbol{V}_h, \Pi_h)$, its ***immediate interior*** *II(S),* ***immediate exterior*** *IE(S), and* ***immediate neighborhood*** *IN(S) are defined as follows:*

(4.1.3) $$II(S) = \{c|\exists(c, d) \in S \; for \; some \; d \in \boldsymbol{V}_h\},$$

(4.1.4) $$IE(S) = \{d|\exists(c, d) \in S \; for \; some \; c \in \boldsymbol{V}_h\}, and$$

(4.1.5) $$IN(S) = II(S) \bigcup IE(S).$$

We then give several definitions about regions.

Definition 4.1.4 *Any subset of* $\boldsymbol{V}_h$ *is called a* ***region*** *or a* ***digital region*** *of* $(\boldsymbol{V}_h, \Pi_h)$*. For any region* X*, its* ***border*** *is defined as*

$$\partial X = \{(p, q) \in \Pi_h | p \in X, and\ q \notin X\}. \tag{4.1.6}$$

Definition 4.1.5 *Two pixels* p *and* q *are called* ***adjacent*** *if* $p = q$ *or* $(p, q) \in \Pi_h$*. Two regions* X *and* Y *are called* ***adjacent*** *if there exist two adjacent pixels* p *and* q *which are in* X *and* Y *respectively.*

Definition 4.1.6 *A* ***path*** *on the hexagonal grid is a sequence of pixels* $\{p_i\}_{i=1}^n \subset \boldsymbol{V}_h$ *such that* p_i *and* p_{i+1} *are adjacent for all* $i = 1, 2, ..., n-1$*. Two pixels* p *and* q *are called "****being connected by a path*** P*" if one of them is the first pixel of the path and the other is the last pixel.*

Definition 4.1.7 *A region* X *is called a* ***connected region*** *if* $(X, \Pi_h|_X)$ *is a connected graph, i.e., for any* $p, q \in X$*, there exists a path in* X *connecting* p *and* q*.*

Definition 4.1.8 *A* ***component*** *(or* ***connected component****) of a region* X *is a maximal connected subset of* X*. A connected component is always non-empty.*

Finally, we give two lemmas about connected components. The first lemma states that for any region X, any connected component of its complement is surrounded [4] by X. The

[4] By "X surrounding Y", we mean that Y is adjacent only to pixels in X.

second lemma states that a connected component of a subset X of a region Y is also a connected component of the superset Y if the component is not adjacent to $Y - X$.

Lemma 4.1.1 ***Lemma One about Connected Components***

For any non-empty region $X \subset \mathbf{V}_h$, let Y be any connected component of $\mathbf{V}_h - X$. Then

(1) Y is adjacent to X,

(2) $II(\partial Y) \subseteq IE(\partial X)$, and

(3) $IE(\partial Y) \subseteq II(\partial X)$.

Proof of (1). Suppose otherwise. Then $IE(\partial Y) \bigcap X = \emptyset$. So $IE(\partial Y) \subseteq \mathbf{V}_h - Y$. Let w be any element in $IE(\partial Y)$. So $w \in \mathbf{V}_h - Y$. Because Y is a connected component of $\mathbf{V}_h - X$ and $w \in \mathbf{V}_h - X$ is adjacent to Y, $w \in Y$. It's contradictory to our assumption that $w \in IE(\partial Y)$.

Proof of (2). Suppose otherwise. So there exists $y \in II(\partial Y) - IE(\partial X)$. Thus there exists $z \in IE(\partial Y)$ such that z is adjacent to y and $z \notin X$. Thus we have $z \in \mathbf{V}_h - X$ and z is adjacent to Y. Because Y is a connected component of $\mathbf{V}_h - X$, $z \in Y$. It's contradictory to our assumption that $z \in IE(\partial Y)$.

Proof of (3). Suppose otherwise: there exists $z \in IE(\partial Y) - II(\partial X)$. So there exists $y \in Y$ such that y and z are adjacent.

We claim that $z \notin X$. Suppose otherwise: z is in X. Because z is adjacent to y and $y \notin X$, $z \in II(\partial X)$. It's contradictory to our assumption, which proves $z \notin X$.

So we have $z \in \mathbf{V}_h - X$ and z is adjacent to Y. So $z \in Y$. It's contradictory to our assumption that $z \in IE(\partial Y)$.

∎

Lemma 4.1.2 ***Lemma Two about Connected Components***

Let X and Y be two regions on the hexagonal grid such that $X \subseteq Y$. Let W be a connected component of X. If W and $Y - X$ are not adjacent, then W is also a connected component of Y.

Proof. Obviously there is a connected component Z of Y such that $W \subseteq Z$. To proof this lemma, it suffices to prove that $W = Z$. Suppose otherwise: $W \neq Z$. Let z be any element of $Z - W$ and w be any element of W. Because both of z and w are in Z, there is a path $P = \{w_0 = z, w_1, ..., w_n = w\}$ in Z connecting z and w. Let w_m be the first element of P which is in W. Obviously $0 < m \leq n$. So we have $w_{m-1} \in Z - W$ and $w_m \in W$. Because W is a connected component of X, $w_{m-1} \notin X$. So W is adjacent to $w_{m-1} \in Y - X$. So W is adjacent to $Y - X$, contradictory to our assumption. This contradiction proves that $W = Z$.

∎

4.1.3 Boundaries of Regions

In this sub-section, we discuss the boundaries of regions. We first model the counter-clockwise direction along the border of a region by a "next" function. The "next" function is a bijective function on the border of the region. The border of a region can be decomposed into connected boundaries and each connected boundary is a list of edgels, connected together by the "next" function.

On the Euclidean plane or sphere, a simply-connected region is a connected region with no holes. A simply-connected region has such a property that its complement is also connected. We use this property to define the simple connection on the hexagonal grid. Then we prove that a region on the hexagonal grid is simply-connected if and only if its border is just a connected boundary. This statement confined to bounded regions is an equivalent of Jordan's Curve Theorem on the hexagonal grid.

The definitions and theorems are discussed formally as follows. We first give a formal way to define boundless lists and circular lists.

Definition 4.1.9 *A **boundless list** is a pair $(X, next)$ such that X is a set and*

(1) next is a bijective function from X to X; and

(2) $\forall x, y \in X$, there exists an integer n such that $x = next^n(y)$.

*A **circular list** is a boundless list $(X, next)$ such that X is a finite set.*

It's easy to see that for a circular list $(X, next)$ and any $x, y \in X$, there exists an integer $n \geq 0$ such that $x = next^n(y)$. By definition, an empty set and a function on the empty set is also a circular list. This gives us a uniform way to deal with the boundaries of simply-connected regions, no matter whether they are empty or not. We now model the counter-clockwise direction along the border of a region by a "next" function. The "next" function is first defined on the border of a pixel.

Definition 4.1.10 *Each pixel P has 6 edgels, named counter-clockwise as $e_i(P)$ for $i = 0, 1, ..., 5$. They correspond to the 6 edges of the hexagon of P. See Figure 4.1.3(b). We define function $next_P$ on $\{e_i(P)\}_{i=0}^{5}$ by*

(4.1.7) $$next_P(e_i(P)) = e_{(i+1)\%6}(P).$$

Definition 4.1.11 *For any region X, we define function $next_X$ on ∂X such that for any $e = (p, q) \in \partial X$,*

(4.1.8) $$next_X(e) = \begin{cases} next_p(e) & \text{if } next_p(e) \in \partial X; \\ (next_q^{-1}(e^{-1}))^{-1} & \text{otherwise.} \end{cases}$$

*The function $next_X$ is called the **next** function on the border of X.*

Function $next_X$ defines the counter-clockwise direction in the border of X. For example, in

Figure 4.1.3(c), $next_X(\alpha) = \beta$ and $next_X(\beta) = \gamma$. On the border of a region, a connected boundary is a list of edgels connected by the next function.

Definition 4.1.12 *For any region $X \subseteq \boldsymbol{V}_h$, a **boundary** (or **connected boundary**) of X is a subset S of its border ∂X such that $next_X(S) = S$ and $(S, next_X|_S)$ is a boundless list.*

About the boundaries of regions, we have two lemmas as follow.

Lemma 4.1.3 ***Lemma One about Boundaries***

For any region $X \subseteq \boldsymbol{V}_h$,

(1) $\partial X = (\partial(\boldsymbol{V}_h - X))^{-1}$;

(2) for any $s \in \partial X$, $next_X(s) = (next^{-1}_{\boldsymbol{V}_h - X}(s^{-1}))^{-1}$;

(3) the border of X can be decomposed into a set of connected boundaries;

(4) if X is bounded (i.e., it's finite), then its connected boundaries are circular lists; and

(5) $\partial X = \emptyset$ iff $X = \boldsymbol{V}_h$ or $X = \emptyset$.

This lemma is obvious. The properties (1) and (2) in this lemma state that the borders of complementing sets are isomorphic. This statement is not true for the square grid.

Lemma 4.1.4 ***Lemma Two about Boundaries***

For any region $X \subseteq \boldsymbol{V}_h$ and any $\alpha \in \partial X$, *(1)* $IEP(Next_X(\alpha))$ *and* $IEP(\alpha)$ *are adjacent;*

(2) $IIP(Next_X(\alpha))$ *and* $IIP(\alpha)$ *are adjacent;*

(3) $IEP(\alpha)$ *and* $IIP(\alpha)$ *are adjacent;*

(4) $IIP(\alpha)$ *and* $IEP(Next_X(\alpha))$ *are adjacent;*

(5) $IEP(\alpha)$ *and* $IIP(Next_X(\alpha))$ *are adjacent;*

(6) $II(\partial X) = IE(\partial(\mathbf{V}_h - X))$*; and*

(7) $IE(\partial X) = II(\partial(\mathbf{V}_h - X))$.

This lemma is obvious. Next we define the simple connection on the hexagonal grid.

Definition 4.1.13 *A region* X *is called* ***simply connected*** *if both* X *and* $\mathbf{V}_h - X$ *are connected.*

The simple connection we defined here is a little bit different from the traditional definition on the Euclidean plane. However, if we add an infinite pixel to the hexagonal grid and make it a digital sphere, then our definition fits the definition of simple connection on Euclidean sphere. About simple connection, we have a theorem: a region is simple connected iff its border equipped with its next function is a boundless list. See Figure 4.1.3(c). The theorem consists of two parts, which are given in the following two lemmas.

Lemma 4.1.5 *For any region* $X \subseteq \mathbf{V}_h$*, if* $(\partial X, next_X)$ *is a boundless list, then* X *is simply connected.*

Proof. If $X = \emptyset$ or $X = \mathbf{V}_h$, then X is obviously simply-connected. So we assume that X is a region such that $X \neq \emptyset$, $X \neq \mathbf{V}_h$, and $(\partial X, next_X)$ is a boundless list. To prove that X is simply connected, it suffices to prove both X and its complement are connected. To prove X is connected, we will prove X has only one connected component. The basic idea to do so is simple. Suppose X has components X_1 and X_2. By Lemma 4.1.1, the two components X_1 and X_2 will be adjacent to the border of X, which is a boundless list. By the definition of components, X_1 and X_2 both cover the immediate interior of ∂X. Thus the two components must be identical. So X has only one component, i.e., it's connected. Similarly, we can prove that $\mathbf{V}_h - X$ is connected. The formal proof is given below.

First we prove that X is connected. Let X_1 and X_2 be two arbitrary connected components of X. To prove that X is connected, it suffices to prove that $X_1 = X_2$. By assumption, $X \neq \mathbf{V}_h$. So is X_1. By Lemma 4.1.3(5), $\partial X_1 \neq \emptyset$. So $II(\partial X_1) \neq \emptyset$. By Lemma 4.1.1(2) and Lemma 4.1.4(6), $II(\partial X_1) \subseteq IE(\partial(\mathbf{V}_h - X)) = II(\partial X)$. Let x be any element in $II(\partial X_1)$. Thus $x \in X_1 \bigcap II(\partial X)$.

Becasue $(\partial X, next_X)$ is a boundless list, by Lemma 4.1.4(2), $II(\partial X)$ is connected. Because X_1 is a connected component of X and intersects with $II(\partial X)$, $II(\partial X)$ is a subset of X_1. Similarly we can prove that $II(\partial X)$ is also a subset of X_2. Because X_1 and X_2 be two connected components of X, $X_1 = X_2$. This proves that X is connected. In the same way, we can prove that $\mathbf{V}_h - X$ is connected. So X is simply connected.

$\blacksquare$

Lemma 4.1.6 *For any simply-connected region $X \subseteq \mathbf{V}_h$, $(\partial X, next_X)$ is a boundless list.*

Proof. If $X = \emptyset$ or $X = \mathbf{V}_h$, then $\partial X = \emptyset$ and by definition, $(\partial X, next_X)$ is a boundless list. So we assume that X is a simply-connected region such that $X \neq \emptyset$ and $X \neq \mathbf{V}_h$. It's easy to check that $next_X$ is a bijective function on ∂X. To prove this lemma, it suffices to prove that for any α and β in ∂X, there exists an integer n such that $\beta = next_X^n(\alpha)$. The basic idea to do so is illustrated in the Figure 4.1.4. Because X and $\mathbf{V}_h - X$ are connected, there exist paths P and Q in X and $\mathbf{V}_h - X$, respectively, connecting α and β. The two paths enclose a region, called Z. Now consider the edgel sequence $\{next_X^i(\alpha)\}_{i=-\infty}^{\infty}$. It's easy to see that α and β are the only entrance and exit for the edgel sequence to get into and out of the region Z. Thus β is in the edgel sequence. So there exists an integer n such that $\beta = next_X^n(\alpha)$. The formal proof is given below.

Let $\alpha = (p_1, q_1)$ and $\beta = (p_2, q_2)$ be two arbitrary elements of ∂X. We can suppose that $\beta \neq next_X^n(\alpha)$ for any integer n with $-3 \leq n \leq 3$, because, otherwise, the lemma is proved. Because X is simply-connected, both X and $\mathbf{V}_h - X$ are connected. Thus there exists a path $P = \{x_i\}_{i=1}^{n_p}$ in X connecting p_1 and p_2. There is also a path $Q = \{y_i\}_{i=1}^{n_q}$ in $\mathbf{V}_h - X$ connecting q_1 and q_2. Each pixel in path P or Q is also a point in the Euclidean plane $\mathbb{R}^2$. Let Z be the polygon in $\mathbb{R}^2$ whose vertices are the pixels in paths P and Q. See

Figure 4.1.4. Each edgel in ∂X corresponds to a line segment in $\mathbb{R}^2$. Let E be the set of edgels in ∂X whose corresponding line segments lie within the polygon Z. E is a finite set. See Figure 4.1.5. It's easy to check that α and β are the only two elements of ∂X whose corresponding line segments intersect the boundary of polygon Z.

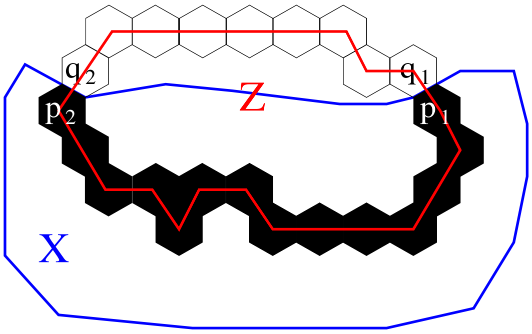


Figure 4.1.4: The polygon Z whose vertices are the pixels in paths P and Q.

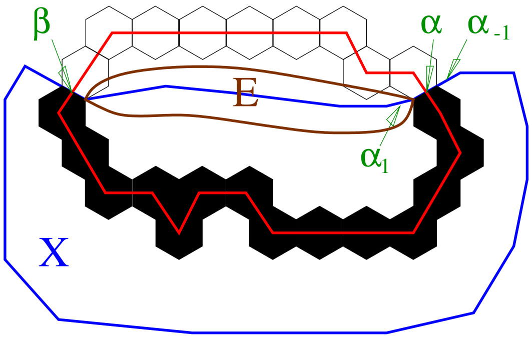


Figure 4.1.5: The set E of edgels in ∂X whose corresponding line segments lie within the polygon Z.

Let $\alpha_{-1} = next_X^{-1}(\alpha)$ and $\alpha_1 = next_X^1(\alpha)$. So either of α_{-1} or α_1 belongs to E, but not both. Without loss of generality, we suppose $\alpha_1 \in E$ and $\alpha_{-1} \notin E$. Now we consider two cases: (1) there exists a positive integer n such that $next_X^n(\alpha) = \alpha_{-1} \notin E$; and (2) there doesn't exist such a positive integer. In the second case, $\{next_X^i(\alpha)\}_{i=1}^{\infty}$ is an infinite sequence. Because E is finite, in case (2), we can also find some positive integer n such that $next_X^n(\alpha) \notin E$. So in both cases, there exists $n > 0$ such that $next_X^n(\alpha) \notin E$.

Now let $next_X^m(\alpha)$ be the first element in the sequence $\{next_X^i(\alpha)\}_{i=1}^n$ such that $next_X^m(\alpha) \notin E$. So $1 < m \leq n$. By our definition of $next_X$, the corresponding line segments of $next_X^m(\alpha)$ and $next_X^{m-1}(\alpha)$ are end-to-end connected. Because the corresponding line segments of $next_X^{m-1}(\alpha)$ lies in polygon Z (implied by $next_X^{m-1}(\alpha) \in E$), the line segments of $next_X^m(\alpha)$ intersects the boundary of Z. So either $next_X^m(\alpha) = \alpha$ or $next_X^m(\alpha) = \beta$. $next_X^m(\alpha) = \alpha$ can't be true because $next_X^{m-1}(\alpha) \in E$ and $next_X^{-1}(\alpha) = \alpha_{-1} \notin E$. So $next_X^m(\alpha) = \beta$.

∎

Theorem 4.1.7 ***Theorem about Simple Connection***

A region $X \subseteq \mathbf{V}_h$ is simply connected iff $(\partial X, next_X)$ is a boundless list.

This theorem follows from Lemmas 4.1.5 and 4.1.6.

Corollary 4.1.8 *For a bounded region X, X is simply connected iff $(\partial X, next_X)$ is a circular list. That is to say, for a bounded region X, $(\partial X, next_X)$ is a circular list iff both X and $\mathbf{V}_h - X$ are connected.*

This corollary is an equivalent of Jordan's Curve Theorem [5] on the hexagonal grid.

4.2 Monotonic Line

In this section, we first model a gray image as a real valued function on a nonempty, bounded, and simply-connected region. Then we define monotonic lines and prove that monotonic lines don't cross each other.

4.2.1 Gray Images

Next, we model gray images on the hexagonal grid. The gray images can be the images from computer imaging, or discrete data from other disciplines.

Definition 4.2.1 *A* ***gray image*** *is a pair* (f, Ω) *such that* $\Omega \subset \mathbf{V}_h$ *is a nonempty, bounded, and simply-connected region, and* f *is a real valued function defined on* Ω. Ω *is the image domain.*

Definition 4.2.2 *For any gray image* (f, Ω), *we extend the function* f *to the whole plane*

[5] The theorem states that any continuous simple closed curve in the plane separates the plane into two disjoint connected regions, the inside and the outside.

by

(4.2.1)
$$f_E(p) = \begin{cases} f(p) & \textit{if } p \in \Omega; \\ -\infty & \textit{otherwise.} \end{cases}$$

*f_E is called the **extended function** of f.*

The reason we extend the function f this way is to ensure that the boundary of the image domain is a monotonic line and becomes the root of the monotonic tree. In fact, instead of choosing $-\infty$, we can choose any value which is less than all values assumed by f, or greater than all values assumed by f.

In the rest of this chapter, we let $I = (f, \Omega)$ be a fixed gray image, and f_E be the extended function of f, unless otherwise stated.

4.2.2 Definition of Monotonic Line

Now we can define the concept "monotonic line".

Definition 4.2.3 *A **monotonic line** (also called a **monoline**) of image I is a boundary ∂X such that $X \subseteq \Omega$ is a simply-connected and nonempty region, and there exists a real number v with the property that either of the following is true:*

(1) $\forall x \in II(\partial X), f_E(x) > v$, and $\forall y \in IE(\partial X), f_E(y) < v$;

(2) $\forall x \in II(\partial X), f_E(x) < v$, and $\forall y \in IE(\partial X), f_E(y) > v$.

*Condition (1) states that function f_E assumes higher pixel values in immediate interior pixels than in immediate exterior pixels; condition (2) states that function f_E assumes lower pixel values in immediate interior pixels than in immediate exterior pixels. If (1) is true, ∂X is called **outward falling**; if (2) is true, ∂X is called **outward climbing**. X and $\boldsymbol{V}_h - X$ are called the **interior** and **exterior** of the monoline ∂X, respectively. We denote the set of all monotonic lines of image I as $MonoLineSet(I)$.*

An example of monotonic lines is shown in Figure 4.2.1. In the rest of the dissertation, we often represent a monotonic line as the boundary of its interior, for example, ∂X.

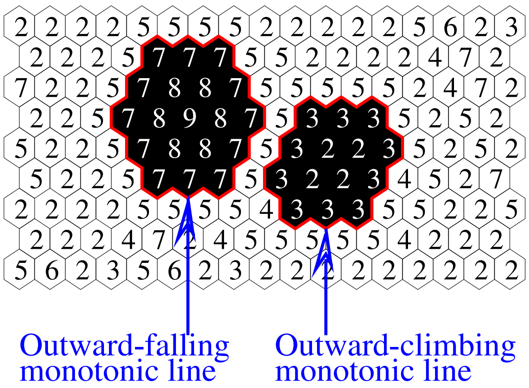


Figure 4.2.1: An example of monotonic lines.

4.2.3 No Crossing Property of Monotonic Lines

Based on the definition of monotonic lines, we can prove that monotonic lines don't cross each other, i.e., for any monotonic lines ∂X and ∂Y, one of the following is true: $X \subseteq Y$,

$Y \subseteq X$, or $X \bigcap Y = \emptyset$. The basic idea to prove this is simple. Suppose we have two monotonic lines ∂X and ∂Y crossing each other. Let a, b, c and d be four pixels around a crossing point. See Figure 4.2.2. By the definition of monotonic lines, we discuss on four cases:

(1) both ∂X and ∂Y are outward falling;

(2) ∂X is outward falling, ∂Y is outward climbing;

(3) ∂X is outward climbing, ∂Y is outward falling; and

(4) Both ∂X and ∂Y are outward climbing.

For each case, we can get a contradiction. For example, in case (1), there exists v_1 such that

$$f_E(a) > v_1, f_E(c) > v_1, \text{ and } f_E(b) < v_1, f_E(d) < v_1;$$

and there exists v_2 such that

$$f_E(c) > v_2, f_E(d) > v_2, \text{ and } f_E(a) < v_2, f_E(b) < v_2.$$

Then we get both $f_E(a) > v_1 > f_E(d)$ and $f_E(d) > v_2 > f_E(a)$, which is a contradiction.

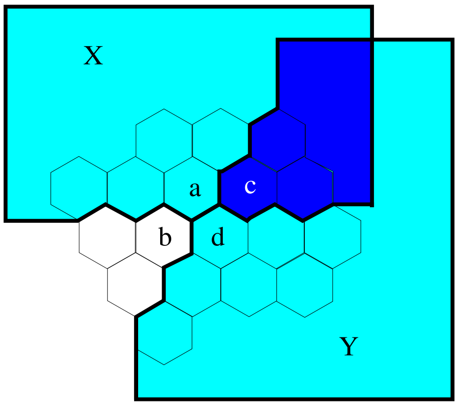


Figure 4.2.2: ∂X crosses ∂Y.

The difficult part is to prove the existence of such pixels as a, b, c, and d. Let's prove several lemmas first.

Lemma 4.2.1 *For any two simply connected regions X and Y on the hexagonal grid such that both $X - Y$ and $Y - X$ are not empty, we suppose $x \in X - Y$ and $y \in Y - X$. Let S_x be the connected component of $X - Y$ which contains x, i.e. $S_x = \{w \in X - Y|$ there exists a path in (X-Y) connecting w and $x\}$. Then S_x is simply connected.*

Proof. Obviously, S_x is connected. Suppose it's not simply connected. So $\mathbf{V}_h - S_x$ is not connected. Obviously, $y \in \mathbf{V}_h - S_x$. Let R be a connected component of $\mathbf{V}_h - S_x$ such that $y \notin R$. Then we can prove the following properties of R:

(1) $R \subseteq X$.

Suppose otherwise. Let $w \in R - X$. So $w \in \mathbf{V}_h - X$. Because X is simply connected, so is $\mathbf{V}_h - X$. Because $y \in \mathbf{V}_h - X$, there exists a path P in $\mathbf{V}_h - X$ which connects w and y. Because $\mathbf{V}_h - X \subseteq \mathbf{V}_h - S_x$, P is also a path in $\mathbf{V}_h - S_x$. Because $w \in R$ and R is a connected component of $\mathbf{V}_h - S_x$, $P \subseteq R$. So $y \in P \subseteq R$, which contradicts with our assumption that $y \notin R$.

(2) R is adjacent to S_x.

This property is implied by Lemma 4.1.1(1).

(3) $R \bigcap Y$ is not empty.

By (2), there is is some $v \in R$ such that v is adjacent to S_x. To prove $R \bigcap Y \neq \emptyset$, it suffices to prove that $v \in R \bigcap Y$, i.e., $v \in Y$. Suppose $v \notin Y$. Then $v \in X - Y$. By our definition of S_x, $v \in S_x$, contradictory to our assumption $v \in R \subseteq \mathbf{V}_h - S_x$. So we get $v \in Y$.

Now let $v \in R \bigcap Y$ be fixed. Because v and y are both in Y, there is a path P in Y which connects v and y. Because $Y \subseteq \mathbf{V}_h - S_x$, P is also a path in $\mathbf{V}_h - S_x$. Because $v \in R$ and R is a connected component of $\mathbf{V}_h - S_x$, $P \subseteq R$. So $y \in P \subseteq R$, which contradicts with our assumption that $y \notin R$.

∎

Lemma 4.2.2 *For any two finite and simply connected regions X and Y on the hexagonal grid, if there exist $x \in X - Y$ and $y \in Y - X$, then there exist $x_1 \in X - Y$ and $z_1 \in \mathbf{V}_h - X - Y$ such that:*

(1) there exists a path in $X - Y$ which connects x and x_1; and

(2) x_1 and z_1 are adjacent.

Proof. Because X and Y are both finite, there exists some $w \in \mathbf{V}_h - X - Y$. Because Y is simply connected, $\mathbf{V}_h - Y$ is connected. Because $w \notin Y, x \notin Y$, and $\mathbf{V}_h - Y$ is connected, there is a path $P = \{w_0 = x, w_1, ..., w_n = w\} \subseteq \mathbf{V}_h - Y$. Let $w_{\tilde{n}}$ be the first element in P such that $w_{\tilde{n}} \notin X$. Let $Q = \{w_0 = x, w_1, ..., w_{\tilde{n}-1}\}$. So we have $Q \subseteq X - Y$ and $w_{\tilde{n}} \in \mathbf{V}_h - X - Y$. Let $x_1 = w_{\tilde{n}-1}$ and $z_1 = w_{\tilde{n}}$. Then we get (1) and (2).

∎

Lemma 4.2.3 *For any monotonic lines ∂X and ∂Y of image I, one of the four sets, $II(\partial X) \bigcap IE(\partial Y)$, $IE(\partial X) \bigcap IE(\partial Y)$, $II(\partial X) \bigcap II(\partial Y)$, or $IE(\partial X) \bigcap II(\partial Y)$ must be empty.*

Proof. Suppose otherwise. Let a, b, c, and d be four pixels such that

$a \in II(\partial X) \bigcap IE(\partial Y)$,

$b \in IE(\partial X) \bigcap IE(\partial Y)$,

$c \in II(\partial X) \bigcap II(\partial Y)$, and

$d \in IE(\partial X) \bigcap II(\partial Y)$.

Pixels a, b, c, and d are corresponding to those in Figure 4.2.2. By the definition of monotonic line, we discuss on four cases:

(1) both ∂X and ∂Y are outward falling;

(2) ∂X is outward falling, ∂Y is outward climbing;

(3) ∂X is outward climbing, ∂Y is outward falling; and

(4) both ∂X and ∂Y are outward climbing.

In each case, we can get a contradiction. Let's take the first case as an example. In the first case, there exist real values v_1 and v_2 such that

$\forall x_{in} \in II(\partial X), f_E(x_{in}) > v_1; \forall x_{out} \in IE(\partial X), f_E(x_{out}) < v_1$; and

$\forall y_{in} \in II(\partial Y), f_E(y_{in}) > v_2; \forall y_{out} \in IE(\partial Y), f_E(y_{out}) < v_2.$

Then we have both $f_E(a) > v_1 > f_E(d)$ and $f_E(d) > v_2 > f_E(a)$, which is a contradiction.

∎

Finally, we get to our "No Crossing Theorem".

Theorem 4.2.4 ***No Crossing Theorem***

$\forall \partial X, \partial Y \in MonoLineSet(I)$, *one of the following is true:*

$$X \subseteq Y,\ Y \subseteq X, \text{ or } X \bigcap Y = \emptyset.$$

Proof. Suppose otherwise: there exist some $x \in X - Y, y \in Y - X$ and $z \in X \bigcap Y$. We need to find a contradiction. The difficult part is to prove the existence of such pixels as a, b, c, and d in Figure 4.2.2. Let $S_x = \{w \in X - Y|$ there exists a path in $(X - Y)$ connecting w and x $\}$, i.e., S_x is the connected component of $X - Y$ such that $x \in S_x$. Let S_y be the connected component of $Y - X$ such that $y \in S_y$. Of course both S_x and S_y are finite and connected. The pixels corresponding a, b, c, and d in Figure 4.2.2 are found along the borders of S_x and S_y.

We first prove some properties of S_x and S_y.

(1) S_x is simply connected.

This comes directly from Lemma 4.2.1.

(2) $IE(\partial S_x) \subseteq (\mathbf{V}_h - X - Y)\bigcup(Y - X)\bigcup(X \bigcap Y)$.

Suppose otherwise. Then there exists some $w \in IE(\partial S_x) - (\mathbf{V}_h - X - Y)\bigcup(Y - X)\bigcup(X \bigcap Y)$. Obviously, $(\mathbf{V}_h - X - Y)\bigcup(Y - X)\bigcup(X \bigcap Y) = \mathbf{V}_h - (X - Y)$. So $w \in X - Y$. Thus by the definition of S_x and the assumption that $w \in IE(\partial S_x)$, we have $w \in S_x$, which is contradictory with the assumption that $w \in IE(\partial S_x)$.

(3) $IE(\partial S_x)\bigcap(X \bigcap Y)$ is not empty.

Because X is simply connected, there is a path $P = \{x_0 = x, x_1, ..., x_n = z\} \subseteq X$. Let $x_{\tilde{n}}$ be the first element in P such that $x_{\tilde{n}} \in X \bigcap Y$. Because $x_0 = x \in X - Y, x_n = z \in X \bigcap Y$, we have $0 < \tilde{n} \leq n$. Let $Q = \{x_0 = x, x_1, ..., x_{\tilde{n}-1}\}$. So we have $Q \subseteq X - Y$. By the definition of S_x, $x_{\tilde{n}-1} \in S_x$. So $x_{\tilde{n}} \in IE(\partial S_x)$. Thus we get $x_{\tilde{n}} \in IE(\partial S_x)\bigcap(X \bigcap Y)$.

(4) $IE(\partial S_x)\bigcap(\mathbf{V}_h - X - Y)$ is not empty.

This is directly from Lemma 4.2.2.

(5.1) $\exists \alpha \in \partial S_x$ such that $IEP(\alpha) \in X \bigcap Y$ and $IEP(Next_{S_x}(\alpha)) \notin X \bigcap Y$.

By (3), there is some $a \in IE(\partial S_x)\bigcap(X \bigcap Y)$. By (4), there is some $b \in IE(\partial S_x) - (X \bigcap Y)$. Let $w_0 \in \partial S_x$ such that $a = IEP(w_0)$. Because S_x is finite and simply connected, $< \partial S_x, Next_{S_x} >$ is a circular list. So there exist $n \in \mathbb{N}$ and

$w_n \in \partial S_x$ such that $w_n = (Next_{S_x})^n(w_0)$ and $b = IEP(w_n)$. Let $P = \{w_i = (Next_{S_x})^i(w_0)\}_{i=0}^n$. Let w_m be the first element in P such that $IEP(w_m) \notin X \bigcap Y$. Obviously $0 < m \leq n$. Let $\alpha = w_{m-1}$. Then we have $IEP(\alpha) \in X \bigcap Y$ and $IEP(Next_{S_x}(\alpha)) \notin X \bigcap Y$.

(5.2) For any α satisfying the condition of (5.1), $IEP(Next_{S_x}(\alpha)) \notin X$.

Suppose otherwise. Then we have $IEP(Next_{S_x}(\alpha)) \in X - Y$. Then by the definition of S_x, we have $IEP(Next_{S_x}(\alpha)) \in S_x$, which is impossible.

(6) $\exists \beta \in \partial S_x$ such that $IEP(\beta) \in \mathbf{V}_h - X - Y$ and $IEP(Next_{S_x}(\beta)) \in Y$.

By (4), there exists $a \in IEP(\partial S_x) \bigcap (\mathbf{V}_h - X - Y)$; and by (3), there exists $b \in IE(\partial S_x) \bigcap (X \bigcup Y)$. Then similar to the proof of (5.1), we can prove that there exists some $\beta \in \partial S_x$ such that $IEP(\beta) \in \mathbf{V}_h - X - Y$ and $IEP(Next_{S_x}(\beta)) \in X \bigcup Y$. To prove (6), it suffices to prove that $IEP(Next_{S_x}(\beta)) \in Y$. Suppose otherwise. Then $IEP(Next_{S_x}(\beta)) \in X - Y$. Then by the definition of S_x, we have that $IEP(Next_{S_x}(\beta)) \in S_x$, which is impossible.

(7) $\exists \gamma \in \partial S_y$ such that $IEP(\gamma) \in \mathbf{V}_h - X - Y$ and $IEP(Next_{S_y}(\gamma)) \in X$.

The proof of (7) is similar to that of (6).

Now we want to apply Lemma 4.2.3. Let α, β, γ be edgels satisfying the conditions of (5.1), (6), and (7), respectively. Next we find the pixels a, b, c, and d.

Let $c = IEP(\alpha)$, $c_1 = IIP(\alpha)$, and $c_2 = IEP(Next_{S_x}(\alpha))$. Then we have $c \in X \bigcap Y$, $c_1 \in S_x \subseteq X - Y$, and c is adjacent to c_1. So $c \in II(\partial Y)$. Similarly we have $c \in X \bigcap Y$, $c_2 \notin X$ (by statement 5.2 above), and c is adjacent to c_2 (by Lemma 4.1.4). So $c \in II(\partial X)$. Thus we have $c \in II(\partial X) \bigcap II(\partial Y)$.

Let $b = IEP(\beta)$, $b_1 = IIP(\beta)$, and $b_2 = IEP(Next_{S_x}(\beta))$. Then we have $b \in \mathbf{V}_h - X - Y$, $b_1 \in S_x \subseteq X$, $b_2 \in Y$, b is adjacent to b_1, and b is adjacent to b_2. So we have $b \in IE(\partial X) \bigcap IE(\partial Y)$.

Let $a = IIP(\beta)$, $a_1 = IEP(\beta)$, and $a_2 = IEP(Next_{S_x}(\beta))$. Then we have $a \in X - Y$, $a_1 \in \mathbf{V}_h - X - Y$, $a_2 \in Y$, a is adjacent to a_1, and a is adjacent to a_2. So $a \in II(\partial X) \bigcap IE(\partial Y)$.

Let $d = IIP(\gamma)$, $d_1 = IEP(\gamma)$, and $d_2 = IEP(Next_{S_y}(\gamma))$. Then we have $d \in Y - X$, $d_1 \in \mathbf{V}_h - X - Y$, $d_2 \in X$, d is adjacent to d_1, and d is adjacent to d_2. So $d \in IE(\partial X) \bigcap II(\partial Y)$.

Now by Lemma 4.2.3, we get a contradiction, which proves the theorem.

▮

4.3 Monotonic Tree and Topological Monotonic Tree

In this section, we define the concepts of *monotonic tree* and *topological monotonic tree*.

4.3.1 Definition of Monotonic Tree

We first define some relations on $MonoLineSet(I)$.

Definition 4.3.1 *For any monotonic lines $\partial X, \partial Y \in MonoLineSet(I)$,*

- *∂X **encloses** ∂Y, denoted as $Enclose(\partial X, \partial Y)$, if $X \supset Y$;*
- *∂X **directly encloses** ∂Y, denoted as $DirectEnclose(\partial X, \partial Y)$, if $X \supset Y$ and there is no $\partial Z \in MonoLineSet(I)$ such that $X \supset Z \supset Y$;*
- *∂X **uniquely directly encloses** ∂Y, denoted as $UniqueDirectEnclose(\partial X, \partial Y)$, if ∂X directly encloses ∂Y, and ∂Y is the only monotonic line which is directly enclosed by ∂X.*

We can prove that *directly-enclosing* is a parent-child relation on the set of monotonic lines. The proof is given in the following three lemmas.

Lemma 4.3.1 *For any $\partial C \in MonoLineSet(I)$, if $\partial C \neq \partial\Omega$, then there exists $\partial P \in MonoLineSet(I)$ such that $DirectEnclose(\partial P, \partial C)$.*

Proof. Let $AncestorSet(\partial C)$ be the set of monotonic lines enclosing ∂C. Obviously, $\partial\Omega \in AncestorSet(\partial C)$. Let ∂P be some element in $AncestorSet(\partial C)$ such that it doesn't enclose any other element of $AncestorSet(\partial C)$. By definition, ∂P directly encloses ∂C.

∎

Lemma 4.3.2 *For any $\partial P_1, \partial P_2, \partial C \in MonoLineSet(I)$, if DirectEnclose $(\partial P_1, \partial C)$ and DirectEnclose$(\partial P_2, \partial C)$, then $\partial P_1 = \partial P_2$.*

Proof. By the definition of $DirectEnclose$, $C \subset P_1$ and $C \subset P_2$. So $C \subseteq P_1 \bigcap P_2$. By Theorem 4.2.4, either $P_1 \subseteq P_2$ or $P_2 \subseteq P_1$. Without loss of generality, suppose $P_1 \subseteq P_2$. So we have $C \subset P_1 \subseteq P_2$. Because DirectEnclose$(\partial P_2, \partial C)$, $P_1 = P_2$. So $\partial P_1 = \partial P_2$.

∎

Lemma 4.3.3 *For any $l \in MonoLineSet(I)$, there exists no sequence $\{l_0 = l, l_1, l_2, ..., l_n = l\}$ such that l_i directly encloses l_{i+1} for all $i = 0, 1, ..., n-1$.*

The above lemma is obvious. By the lemmas 4.3.1 to 4.3.3, the relation $DirectEnclose$ is a parent-child relation on the set of monotonic lines in gray image I. So we get the following theorem, based on which we can define the monotonic tree.

Theorem 4.3.4 *$(MonoLineSet(I), DirectEnclose)$ is a rooted tree, and $\partial\Omega$ is its root.*

Definition 4.3.2 *The tree $(MonoLineSet(I), DirectEnclose)$ is called the **monotonic tree** or **monotree** of image I, and denoted as $MonoTree(I)$.*

4.3.2 Definition of Topological Monotonic Tree

Next we define monotonic slopes and topological monotonic trees.

Definition 4.3.3 *A **monotonic slope** $\mathcal{S}$ is a maximal sequence of monotonic lines $\mathcal{S} = \{l_i\}_{i=1}^{n}$ with $n \geq 1$ such that*

(1) $\forall i = 1, 2, ..., n-1$, $UniqueDirectEnclose(l_i, l_{i+1})$; and

(2) either all l_i are outward-falling, or all l_i are outward-climbing.

*$\mathcal{S}$ is called **outward-falling/outward-climbing** if all l_i are outward-falling/outward-climbing. The first monotonic line l_1 is called the **enclosing line** of the slope $\mathcal{S}$. The set of all monotonic slopes is denoted as $MonoSlopeSet(I)$.*

We can also define some relations on $MonoSlopeSet(I)$.

Definition 4.3.4 *For any $s_a, s_b \in MonoSlopeSet(I)$, we define*

(1) $Enclose(s_a, s_b)$ *if* $\exists l_a \in s_a, \exists l_b \in s_b, Enclose(l_a, l_b)$*;*

(2) $DirectEnclose(s_a, s_b)$ *if* $\exists l_a \in s_a, \exists l_b \in s_b, DirectEnclose(l_a, l_b)$.

The relation $DirectEnclose$ is a parent-child relation on the set of all monotonic slopes. Based on this property, we can define the topological monotonic tree.

Theorem 4.3.5 $(MonoSlopeSet(I), DirectEnclose)$ *is a rooted tree, and the monotonic slope which contains $\partial\Omega$ is the root.*

Definition 4.3.5 *The tree $(MonoSlopeSet(I), DirectEnclose)$ is called the **topological monotonic tree** of I, and denoted as $TopoMonoTree(I)$.*

Chapter 5

Properties of Monotonic Tree

In this chapter, we present some theorems about monotonic trees. The theorems, together with the "No Crossing Theorem" (Theorem 4.2.4), describe the basic properties of monotonic trees, which are used in next chapter to design an efficient algorithm for computing monotonic trees.

5.1 Lemmas

Before we prove the theorems, we need four important lemmas about simply-connected regions:

- Lemma 5.1.1 **Boundary Lemma One for Simply-Connected Regions**.

  This lemma states that a simply-connected subset of a simply-connected region is identical to the superset if it covers the immediate interior of the superset and the superset is not equal to the digital plane. See Figure 5.1.1 (a). A corollary to this lemma (Corollary 5.1.2) states that a non-empty simply-connected subset of a simply-connected region is identical to the superset if its immediate exterior is covered by the immediate exterior of the superset.

- Lemma 5.1.3 **Boundary Lemma Two for Simply-Connected Regions**.

  This lemma states that if a set of disjoint simply-connected subsets $\{Y_k \subseteq X\}_{k=1}^{n}$ covers the immediate interior of their simply-connected superset X, then there exist two different subsets Y_i and Y_j such that the boundaries of X, Y_i, and Y_j are adjacent to each other. See Figure 5.1.1 (b). A corollary to this lemma (Corollary 5.1.4) states that if a set of disjoint simply-connected regions $\{Y_k\}_{k=1}^{n}$ covers the boundary of a simply-connected region Z from outside, then there are two different sets Y_i and Y_j such that Z, Y_i, and Y_j are adjacent to each other. See Figure 5.1.1 (c).

- Lemma 5.1.5 **Complement Lemma One**.

  This lemma states that for any connected region X, any connected component of its complement is simply-connected and surrounded [1] by X. See Figure 5.1.2 (a).

- Lemma 5.1.6 **Complement Lemma Two**.

[1] By "X surrounding Y", we mean Y is adjacent only to pixels in X.

This lemma states that for any connected subset X of a simply-connected region Z, any connected component of $Z - X$ is simply-connected and surrounded by X if it doesn't touch the boundary of Z; otherwise, the component is disjoint with the minimal simply-connected region which covers X. See Figure 5.1.2 (b).

These lemmas are presented and proved as follows.

Lemma 5.1.1 ***Boundary Lemma One for Simply-Connected Regions***

For any simply-connected regions X and Y on the hexagonal grid, if $Y \subseteq X \neq \mathbf{V}_h$ and $II(\partial X) \subseteq Y$, then $X = Y$. See Figure 5.1.1 (a).

Proof. Suppose otherwise. Let $x \in X - Y$ and $w \in \mathbf{V}_h - X$. Because Y is simply-connected and both of x and w are not in Y, there is a path $P = \{w_0 = x, w_1, ..., w_n = w\}$ in $\mathbf{V}_h - Y$ connecting x and w. Suppose w_m is the first element in P such that $w_m \notin X$. Because $x \in X$ and $w \notin X$, $0 < m \leq n$. Thus we have $w_{m-1} \in X$. So $w_{m-1} \in II(\partial X) \subseteq Y$. So $w_{m-1} \in Y$. It contradicts with our assumption that P is in $\mathbf{V}_h - Y$.

∎

Corollary 5.1.2 ***Boundary Corollary One for Simply-Connected Regions***

For any simply-connected regions X and Y on the hexagonal grid, if $\emptyset \neq Y \subseteq X$ and $IE(\partial Y) \subseteq IE(\partial X)$, then $X = Y$.

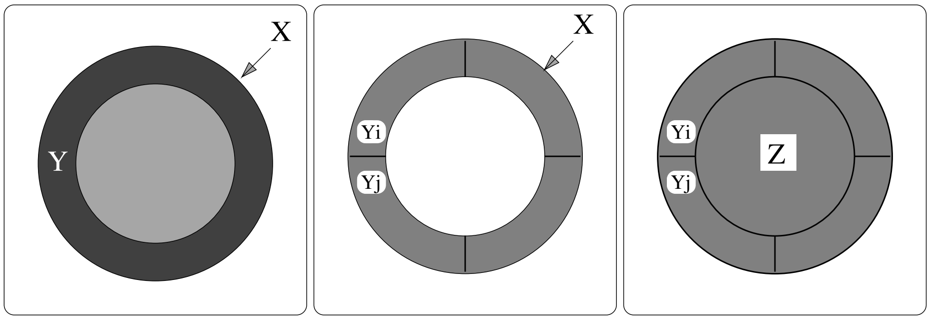


Figure 5.1.1: (a) A proper subset Y in its simply-connected super set X with same boundary is not simple connected. (b) If a set of disjoint simply-connected subsets $\{Y_k\}_{k=1}^n$ covers the immediate interior of their simply-connected superset X, then there are two different subsets Y_i and Y_j such that ∂X, ∂Y_i, and ∂Y_j are adjacent to each other. (c) If a set of disjoint simply-connected sets $\{Y_k\}_{k=1}^n$ covers the boundary of a simply-connected set Z from outside, then there are two different sets Y_i and Y_j such that Z, Y_i, and Y_j are adjacent to each other.

Proof. Let $X_1 = \mathbf{V}_h - X$ and $Y_1 = \mathbf{V}_h - Y$. So $X_1 \subseteq Y_1$. Obviously, $II(\partial Y_1) = IE(\partial Y)$ and $II(\partial X_1) = IE(\partial X)$. Because $IE(\partial Y) \subseteq IE(\partial X)$, $II(\partial Y_1) \subseteq II(\partial X_1)$. So by Lemma 5.1.1, $Y_1 = X_1$, i.e., $X = Y$.

■

Lemma 5.1.3 ***Boundary Lemma Two for Simply-Connected Regions***

For any integer $n > 1$ and any simply-connected regions X, Y_1, Y_2, ..., and Y_n on the hexagonal grid, if $X \neq \mathbf{V}_h$ and

(1) each Y_i is a non-empty subset of X;

(2) $\forall i \neq j, Y_i \bigcap Y_j = \emptyset$; and

(3) $II(\partial X) \subseteq \bigcup_{i=1}^n Y_i$,

then there exist $Y_i \neq Y_j$ for some $i = 1, 2, ...,$ or n and $j = 1, 2, ...,$ or n such that *(i)* $(\partial Y_i)\bigcap(\partial Y_j)^{-1} \neq \emptyset$*;*

(ii) $(\partial Y_i)\bigcap(\partial X) \neq \emptyset$*; and*

(iii) $(\partial Y_j)\bigcap(\partial X) \neq \emptyset$*.*

See Figure 5.1.1 (b).

Proof. First we claim that for any $i = 1, 2, ...,$ or n, $II(\partial X) - Y_i \neq \emptyset$. Otherwise, by Lemma 5.1.1, we have $X = Y_i$, contradictory to our assumption.

Let α be any element of ∂X. We suppose $IIP(\alpha) \in Y_i$ for some $i = 1, 2, ...,$ or n. From the above claim we know that there is some $\beta \in \partial X$ such that $IIP(\beta) \notin Y_i$. Because X is simply connected, $< \partial X, Next_X >$ is a boundless list. So β can be represented as $(Next_X)^m(\alpha)$ for some $m \in \mathbb{Z}$. We suppose $(Next_X)^k(\alpha)$ is the first element in the sequence $\{(Next_X)^0(\alpha) = \alpha, (Next_X)^1(\alpha), ..., (Next_X)^m(\alpha) = \beta\}$ such that $IIP((Next_X)^k(\alpha)) \notin Y_i$. Let $x = IIP((Next_X)^{k-1}(\alpha))$, $x_E = IEP((Next_X)^{k-1}(\alpha))$, $y = IIP((Next_X)^k(\alpha))$, and $y_E = IEP((Next_X)^k(\alpha))$. So $x \in Y_i$ and $y \in Y_j$ for some $j \neq i$. By Lemma 4.1.4(2), x and y are adjacent. Thus we have:

(i) $(x, y) \in (\partial Y_i)\bigcap(\partial Y_j)^{-1}$;

(ii) $(x, x_E) \in (\partial Y_i)\bigcap(\partial X)$; and

(iii) $(y, y_E) \in (\partial Y_j)\bigcap(\partial X)$.

∎

Corollary 5.1.4 ***Boundary Corollary Two for Simply-Connected Regions***

For any integer $n > 1$ and any simply-connected regions Z, Y_1, Y_2, ..., and Y_n on the hexagonal grid, if $Z \neq \emptyset$ and

(1) each Y_i is a non-empty subset of $\mathbf{V}_h - Z$;

(2) $\forall i \neq j, Y_i \bigcap Y_j = \emptyset$; and

(3) $IE(\partial Z) \subseteq \bigcup_{i=1}^{n} Y_i$,

then $\exists i \neq j$ such that

(i) $(\partial Y_i) \bigcap (\partial Y_j)^{-1} \neq \emptyset$;

(ii) $(\partial Y_i) \bigcap (\partial Z)^{-1} \neq \emptyset$; and

(iii) $(\partial Y_j) \bigcap (\partial Z)^{-1} \neq \emptyset$.

See Figure 5.1.1 (c).

This corollary can be proved by Lemma 5.1.3 straightforwardly if we let $X = \mathbf{V}_h - Z$.

Lemma 5.1.5 ***Complement Lemma One***

For any connected region X on the hexagonal grid, let Y be a connected component of $\mathbf{V}_h - X$. Then

(1) Y is simply connected,

(2) $II(\partial Y) \subseteq IE(\partial X)$, and

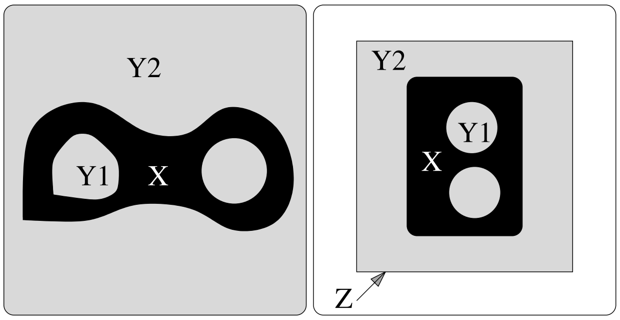
(3) $IE(\partial Y) \subseteq II(\partial X)$.

See Figure 5.1.2 (a).

Proof of (1). Suppose otherwise. Then $\mathbf{V}_h - Y$ is not connected. Let Z_1 be the connected component of $\mathbf{V}_h - Y$ such that $X \subseteq Z_1$ [2], and Z_2 be another connected component of $\mathbf{V}_h - Y$. It's easy to see that $Z_2 \subseteq \mathbf{V}_h - X$. By Lemma 4.1.1(1), Z_2 is adjacent to Y. Because Y is a connected component of $\mathbf{V}_h - X$, Z_2 should be included in Y. It is contradictory to our assumption that Z_2 be a connected component of $\mathbf{V}_h - Y$. This contradiction proves (1).

Proof of (2) and (3). Statements (2) and (3) follow from Lemma 4.1.1(2) and (3).

∎



(a) (b)

Figure 5.1.2: (a) X is the black region, Y_1 and Y_2 are two components of its complement. (b) X is the black region; Z is the larger rectangle; W (not marked in the figure) is the smaller rectangle, which is the minimal simply-connected region covering X; Y_1 and Y_2 are two components of $Z - X$ such that Y_1 doesn't touch the boundary of Z, and Y_2 touches the boundary of Z.

Lemma 5.1.6 ***Complement Lemma Two***

Let Z be a simply connected region on the hexagonal grid, X be a connected subset of Z, and Y be a connected component of $Z - X$. If $II(\partial Y) \bigcap II(\partial Z) = \emptyset$, then

[2] Existence of such a Z_1 is because $X \subseteq \mathbf{V}_h - Y$ and X is connected.

(1) Y *is simply connected,*

(2) $II(\partial Y) \subseteq IE(\partial X)$*, and*

(3) $IE(\partial Y) \subseteq II(\partial X)$*;*

otherwise, there exists a simply connected region $W \subseteq Z$ *such that*

(4) $X \subseteq W$ *and* $W \bigcap Y = \emptyset$*,*

(5) $II(\partial W) \subseteq II(\partial X)$*, and*

(6) $IE(\partial W) \subseteq IE(\partial X)$*.*

See Figure 5.1.2 (b).

The region W in the above lemma is the minimal simply connected region which covers X.

Proof. First we suppose $II(\partial Y) \bigcap II(\partial Z) = \emptyset$. So Y and $\mathbf{V}_h - Z$ are not adjacent. By Lemma 4.1.2, Y is also a connected component of $\mathbf{V}_h - X$. Thus statements (1),(2), and (3) follow from Lemma 5.1.5.

Now suppose $II(\partial Y) \bigcap II(\partial Z) \neq \emptyset$. Thus Y and $\mathbf{V}_h - Z$ are adjacent. Let

$$FirstYSet = \{\tilde{Y} \text{ is a connected component of } Z - X | II(\partial\tilde{Y}) \bigcap II(\partial Z) = \emptyset\};$$

and

$$SecondYSet = \{\tilde{Y} \text{ is a connected component of } Z - X | II(\partial\tilde{Y}) \bigcap II(\partial Z) \neq \emptyset\}.$$

Let $W = X \bigcup (\bigcup_{\tilde{Y} \in FirstYSet} \tilde{Y})$. Obviously, $W \subseteq Z$, $X \subseteq W$ and $W \bigcap Y = \emptyset$. This proves (4).

Next we want to prove W is simply connected. Because each $\tilde{Y} \in FirstYSet$ is connected and adjacent to X (by (2)), W is connected. Obviously,

$$\mathbf{V}_h - W = (\mathbf{V}_h - Z) \bigcup (\bigcup SecondYSet).$$

Because each $\tilde{Y} \in SecondYSet$ is connected and adjacent to $(\mathbf{V}_h - Z)$, $\mathbf{V}_h - W$ is connected. So W is simply connected.

To prove (5), we suppose otherwise: there exists $x \in II(\partial W) - II(\partial X)$. By the definition of W,

$$II(\partial W) \subseteq II(\partial X) \bigcup (\bigcup_{\tilde{Y} \in FirstYSet} II(\partial \tilde{Y})).$$

So $x \in II(\partial \tilde{Y_1})$ for some $\tilde{Y_1} \in FirstYSet$. Let $y \in \mathbf{V}_h - W$ be a pixel such that x and y are adjacent. Because $\tilde{Y_1}$ is not adjacent to $\mathbf{V}_h - Z$, $y \notin \mathbf{V}_h - Z$. So $y \in \tilde{Y_2}$ for some $\tilde{Y_2} \in SecondYSet$. Thus $\tilde{Y_1}$ and $\tilde{Y_2}$ are adjacent. It's contradictory to our assumption that $\tilde{Y_1}$ and $\tilde{Y_2}$ are two connected components of $Z - X$. This proves (5).

Now we has to prove (6). Let $y \in IE(\partial W)$ be arbitrary. So there exists $x \in II(\partial W)$ such that x and y are adjacent. By (5), $x \in II(\partial X)$. Because $y \notin X$, $y \in IE(\partial X)$. So for each $y \in IE(\partial W)$, $y \in IE(\partial X)$. This proves (6).

∎

5.2 Theorems

The five theorems proved in this chapter are:

- Theorem 5.2.1 **Inside Touching Theorem**.

  This theorem states that if two monotonic lines share a common edgel (which implies that the two lines touch from inside), then either they are both outward falling, or they are both outward climbing. See Figure 5.2.1(a).

- Theorem 5.2.3 **Outside Touching Theorem**.

  This theorem states that if two monotonic lines touch from outside, then one of them is outward falling and the other is outward climbing. See Figure 5.2.1(b).

- Theorem 5.2.6 **Separation Theorem**.

  This theorem states that for any two pixels in the image domain, (1) if they are connected by a path where the function assumes a constant value, then no monotonic lines separate them; and (2) if the function assumes different values at the two pixels, then there exist monotonic lines separating them.

- Theorem 5.2.7 **Private Region Theorem**.

  This theorem says that each monotonic line has a nonempty private region (a region enclosed by the line and excluding the regions of the child monotonic lines) and that the function is constant over this region. In addition, the private region is adjacent to the outside of the monotonic line and to each child monotonic line. Then the private value of a monotonic line is defined as the value assumed by the function over the private region of the monotonic line.

- Theorem 5.2.8 **Value Jumping Theorem**.

  This theorem states that (1) there is a value jump (up or down) from a child monotonic line to its parent; (2) if the child monotonic line is outward falling/climbing, then the private value of the parent is less/greater than that of the child.

These theorems are presented and proved as follows. We let $I = (f, \Omega)$ be a fixed gray image on the hexagonal grid, and f_E be the extended function of f, unless otherwise stated.

Theorem 5.2.1 ***Inside Touching Theorem***

For any ∂X and ∂Y in $MonoLineSet(I)$, if

$$(\partial X) \bigcap (\partial Y) \neq \emptyset,$$

then

$$\partial X \textit{ is outward falling} \Leftrightarrow \partial Y \textit{ is outward falling.}$$

Proof. Let $(x, y) \in (\partial X) \bigcap (\partial Y)$. We have: ∂X is outward falling $\Leftrightarrow f_E(y) < f_E(x) \Leftrightarrow$ ∂Y is outward falling, where f_E is the extended function of image I.

∎

Corollary 5.2.2 *For any ∂X and ∂Y in $MonoLineSet(I)$, if*

$$II(\partial X) \bigcap II(\partial Y) \neq \emptyset,$$

then

$$\partial X \textit{ is outward falling} \Leftrightarrow \partial Y \textit{ is outward falling.}$$

Proof. $II(\partial X) \bigcap II(\partial Y) \neq \emptyset$ implies that $X \bigcap Y \neq \emptyset$. By Theorem 4.2.4, either $Y \subseteq X$ or $X \subseteq Y$. Without loss of generality, we suppose $Y \subseteq X$. Let x be any element in $II(\partial X) \bigcap II(\partial Y)$. $x \in II(\partial X)$ implies that there exists some $w \in IE(\partial X)$ such that $(x, w) \in \Pi_h$. $w \in IE(\partial X)$ implies $w \notin X$. So $w \notin Y$. Thus $w \in IE(\partial Y)$. So $(x, w) \in (\partial X) \bigcap (\partial Y)$. Then the corollary follows from Theorem 5.2.1.

∎

The above corollary implies that for a sequence of monotonic lines which intersect from inside, either they are all outward falling, or they are all outward climbing. See Figure 5.2.1(a).

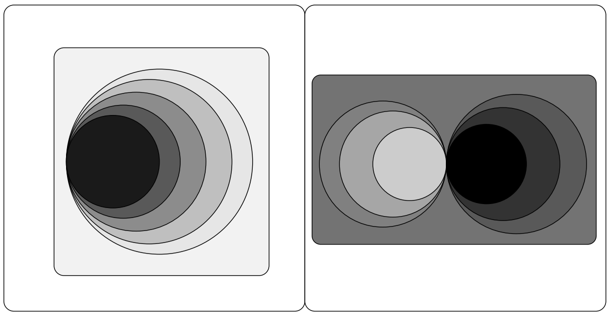
Theorem 5.2.3 ***Outside Touching Theorem***

For any $\partial X, \partial Y \in MonoLineSet(I)$, if

$$(\partial X) \bigcap (\partial Y)^{-1} \neq \emptyset,$$

then

$$\partial X \textit{ is outward falling} \Leftrightarrow \partial Y \textit{ is outward climbing.}$$



(a) (b)

Figure 5.2.1: (a) An inside-touching sequence, and (b) two inside-touching sequences which touch from outside.

Proof. Let $(x, y) \in (\partial X) \bigcap (\partial Y)^{-1}$. So ∂X is outward falling $\Leftrightarrow f_E(y) < f_E(x) \Leftrightarrow \partial Y$ is outward climbing, where f_E is the extended function of image I.

∎

The above theorem implies that for two inside-touching sequences, if they intersect from outside, then the monotonic lines in one sequence are all outward falling, and the monotonic lines in the other sequence are all outward climbing. See Figure 5.2.1(b).

Corollary 5.2.4 *For any ∂X and ∂Y in $MonoLineSet(I)$, if*

$$II(\partial X) \bigcap IE(\partial Y) \neq \emptyset \text{ and } IE(\partial X) \bigcap II(\partial Y) \neq \emptyset,$$

then

$$\partial X \text{ is outward falling} \Leftrightarrow \partial Y \text{ is outward climbing.}$$

Proof. We suppose $x \in II(\partial X) \bigcap IE(\partial Y)$ and $y \in IE(\partial X) \bigcap II(\partial Y)$. So $x \in X - Y$ and $y \in Y - X$. Thus by Theorem 4.2.4, $X \bigcap Y = \emptyset$. Because $x \in IE(\partial Y)$, there exists $w \in II(\partial Y)$ such that $(w, x) \in \partial Y$. $w \in II(\partial Y)$ implies that $w \notin X$. Because $x \in II(\partial X)$ and w is adjacent to x, $w \in IE(\partial X)$. So $(x, w) \in (\partial X) \bigcap (\partial Y)^{-1}$. Then the corollary follows from Theorem 5.2.3.

∎

Similarly, we can prove:

Corollary 5.2.5 *For any $\partial X, \partial Y \in MonoLineSet(I)$, if*

$$II(\partial X) \bigcap IE(\partial Y) \neq \emptyset \text{ and } X \bigcap Y = \emptyset,$$

then

$$\partial X \text{ is outward falling} \Leftrightarrow \partial Y \text{ is outward climbing.}$$

Theorem 5.2.6 ***Separation Theorem***

Let x, y be two pixels in Ω.

(1) If there is a path $P = \{w_1 = x, w_2, ..., w_n = y\}$ in Ω such that f is constant along this path, i.e., $f(w_1) = f(w_2) = ... = f(w_n)$, then $\forall\ \partial X \in MonoLineSet(I), x \in X \Leftrightarrow y \in X$.

(2) If $f(x) \neq f(y)$, then there is some $\partial X \in MonoLineSet(I)$ such that $x \in X \Leftrightarrow y \notin X$.

Proof of (1). Suppose otherwise: there exists $\partial X \in MonoLineSet(I)$ such that either (i) $x \in X$ and $y \notin X$, or (ii) $x \notin X$ and $y \in X$. Without loss of generality, we suppose

(i) is true. Let w_m be the first element in the path P such that w_m is not in X. Obviously $0 < m \leq n$. Then we have $w_{m-1} \in X$, $w_m \notin X$. So $w_{m-1} \in II(\partial X)$ and $w_m \in IE(\partial X)$. Thus either $f(w_{m-1}) > f(w_m)$ (if ∂X is outward falling) or $f(w_{m-1}) < f(w_m)$ (if ∂X is outward climbing). It is contradictory to our assumption that $\forall i = 0, 1, ..., n, f(w_i) = f(x) = f(y)$.

Proof of (2). Without loss of generality, we suppose $f(x) > f(y)$. Let a be a real number such that $f(y) < a < f(x)$ and $\forall z \in \Omega, f(z) \neq a$. Let $A = \{w \in \Omega | f(w) > a\}$ and S_x be the connected component of A such that $x \in S_x$. Thus we have: (i) $\forall w \in S_x$, $f(w) > a$; and (ii) $\forall w \in IE(\partial S_x), f(w) < a$. (i) is obvious. To prove (ii), we suppose otherwise: there exists $w \in IE(\partial S_x)$ such that $f(w) > a$. Thus $w \in A$ and w is adjacent to S_x. By the definition of S_x, $w \in S_x$, which is wrong.

Obviously $y \notin S_x$. Let Y be the connected component of $\Omega - S_x$ such that $y \in Y$. To apply Lemma 5.1.6 (“Complement Lemma Two”), we consider two cases: $II(\partial Y) \bigcap II(\partial \Omega) = \emptyset$ and $II(\partial Y) \bigcap II(\partial \Omega) \neq \emptyset$.

In the first case, we have (iii) Y is simply connected, (iv) $II(\partial Y) \subseteq IE(\partial S_x)$, and (v) $IE(\partial Y) \subseteq II(\partial S_x)$. Statements (i)-(v) imply that $\partial Y \in MonoTree(I)$. By the definition of Y, $x \notin Y$ and $y \in Y$. This proves (2).

In the second case, there exists a simply connected region $W \subseteq \Omega$ such that (vi) $S_x \subseteq W$ and $W \bigcap Y = \emptyset$, (vii) $II(\partial W) \subseteq II(\partial S_x)$, and (viii) $IE(\partial W) \subseteq IE(\partial S_x)$. Statements (i)-(ii) and (vi)-(viii) imply that $\partial W \in MonoTree(I)$, $x \in W$, and $y \notin W$. This proves (2).

∎

Definition 5.2.1 *For any $\partial X \in MonoLineSet(I)$, let $\{\partial Y_i\}_{i=1}^{n}$ ($n \geq 0$) be the set of children of ∂X in the monotonic tree. We define the **private region** of ∂X to be:*

$$PRegion_I(\partial X) = X - \bigcup_{i=1}^{n} Y_i.$$

It's easy to see that the set of all private regions,

$$\{PRegion_I(\partial X) | \partial X \in MonoLineSet(I)\}$$

is a subdivision of the image domain Ω.

Theorem 5.2.7 ***Private Region Theorem***

For any $\partial X \in MonoLineSet(I)$, let $\{\partial Y_i\}_{i=1}^{n}$ ($n \geq 0$) be the set of children of ∂X in the monotonic tree. Then

(1) $PRegion_I(\partial X) \bigcap II(\partial X)$ is not empty;

(2) f is constant over the region $PRegion_I(\partial X)$; and

(3) for each child ∂Y_i, Y_i is adjacent to $PRegion_I(\partial X)$.

Proof of (1). We suppose otherwise. So $II(\partial X) \subseteq \bigcup_{i=1}^{n} Y_i$. If $n = 1$, by Lemma 5.1.1, $X = Y_1$. It's contradictory to our assumption that ∂Y_1 is a child of ∂X. So $n > 1$. By Lemma 5.1.3, there exist $Y_i \neq Y_j$ such that: (i) $(\partial Y_i) \bigcap (\partial Y_j)^{-1} \neq \emptyset$; (ii) $(\partial Y_i) \bigcap (\partial X) \neq \emptyset$; and (iii) $(\partial Y_j) \bigcap (\partial X) \neq \emptyset$.

Statement (ii) implies that ∂Y_i is outward falling $\Leftrightarrow$ ∂X is outward falling (by Theorem 5.2.1). Statement (iii) implies that ∂Y_j is outward falling $\Leftrightarrow$ ∂X is outward falling. So ∂Y_i is outward falling $\Leftrightarrow$ ∂Y_j is outward falling. However, statement (i) implies that ∂Y_i is outward falling $\Leftrightarrow$ ∂Y_j is outward climbing (by Theorem5.2.3). The contradiction proves (1).

Proof of (2). Suppose otherwise: there exist $x, y \in PRegion_I(\partial X)$ such that $f(x) \neq f(y)$. By Theorem 5.2.6, there exists $\partial Y \in MonoTree(I)$ such that either (i) $x \in Y$ and $y \notin Y$; or (ii) $x \notin Y$ and $y \in Y$. Without loss of generality, we suppose (i) is true. By Theorem 4.2.4, either $X \subset Y$ or $Y \subset X$. Because $y \notin Y$, $X \subset Y$ is not true. So $Y \subset X$, which implies that ∂Y is a descendant of ∂X in the monotonic tree. So $Y \subseteq Y_i$ for some $i = 1, 2, ...$ or n. It contradicts with the fact that $x \in Y$ and $x \in PRegion_I(\partial X)$. This proves (2).

Proof of (3). We suppose otherwise. Without loss of generality, we assume that Y_1 is not adjacent to $PRegion_I(\partial X)$. So $IE(\partial Y_1) \subseteq IE(\partial X) \bigcup (\bigcup_{i=2}^{n} II(\partial Y_i)) = II(\partial \mathbf{V}_h - X) \bigcup (\bigcup_{i=2}^{n} II(\partial Y_i)) \subseteq (\mathbf{V}_h - X) \bigcup (\bigcup_{i=2}^{n} Y_i)$. If $n = 1$, by Corollary 5.1.2, we get

$Y_1 = X$, which is wrong. So $n > 1$.

Denote $\mathbf{V}_h - X$ as Y_{n+1}. Now by Corollary 5.1.4, we know that there exist $Y_i \neq Y_j$ for $2 \leq i < j \leq n+1$ such that:

(i) $(\partial Y_j) \bigcap (\partial Y_i)^{-1} \neq \emptyset$;

(ii) $(\partial Y_j) \bigcap (\partial Y_1)^{-1} \neq \emptyset$; and

(iii) $(\partial Y_i) \bigcap (\partial Y_1)^{-1} \neq \emptyset$.

From Theorem 5.2.3 and Statement (iii), we get

(iv) ∂Y_1 is outward falling $\Leftrightarrow$ ∂Y_i is outward climbing.

Now we consider two cases: $j = n+1$ and $j < n+1$.

In the first case, $Y_j = \mathbf{V}_h - X$. By Theorem 5.2.1, statement (ii), and statement (i), we get: ∂Y_1 is outward falling $\overset{by(ii)}{\Longleftrightarrow}$ ∂X is outward falling $\overset{by(i)}{\Longleftrightarrow}$ ∂Y_i is outward falling. It's contradictory to (iv).

In the second case, by Theorem 5.2.3, we get: ∂Y_1 is outward falling $\overset{by(ii)}{\Longleftrightarrow}$ ∂Y_j is outward climbing $\overset{by(i)}{\Longleftrightarrow}$ ∂Y_i is outward falling. It's contradictory to (iv).

∎

The theorem above says that each monotonic line has a nonempty private region and that the function is constant over this region. Based on this theorem, we give the following definition.

Definition 5.2.2 *For any $\partial X \in MonoLineSet(I)$, we define the* ***private value*** *of ∂X to be the constant value assumed by f over $PRegion_I(\partial X)$. The private value of ∂X is denoted as $PValue_I(\partial X)$.*

Theorem 5.2.8 ***Value Jumping Theorem***

For any $\partial P, \partial C \in MonoLineSet(I)$, if ∂P is the parent of ∂C, then

(1) $PValue_I(\partial P) \neq PValue_I(\partial C)$; and

(2) ∂C is outward falling $\Leftrightarrow PValue_I(\partial P) < PValue_I(\partial C)$.

Proof. By Theorem 5.2.7(3), ∂C is adjacent to $PRegion_I(\partial P)$, i.e.,

$$PRegion_I(\partial P) \bigcap IE(\partial C) \neq \emptyset.$$

Let

$y \in PRegion_I(\partial P) \bigcap IE(\partial C)$ and

$x \in PRegion_I(\partial C) \bigcap II(\partial C)$. [3]

By the definition of $PValue_I$, $PValue_I(\partial P) = f(y)$ and $PValue_I(\partial C) = f(x)$. Then by the definition of monotonic line, Def 4.2.3, we have:

∂C is outward falling $\Rightarrow f(x) > f(y)$; and

∂C is outward climbing $\Rightarrow f(x) < f(y)$.

This proves (1) and (2).

∎

[3] By Theorem5.2.7(1), there exists such an x.

Chapter 6

Computation of Monotonic Tree

In this chapter, a top-down algorithm is presented for computing monotonic trees. The algorithm begins with the root monotonic line, browsing the private region of the root to find all child monotonic lines. This process is repeated at each internal node to find its children. When all monotonic lines are found, they are stored in the monotonic tree. Once the monotonic tree is constructed, the topological monotonic tree can be easily obtained by merging the monotonic slopes into new nodes.

The algorithm has the following advantages:

- It takes time linear to the image size. As a comparison, the best contour tree algorithms take $O(n \log n)$ time, where n is the number of pixels or data items.

- It can be implemented in parallel computers. At any time of the computation, the nodes being processed are independent and can be evenly assigned to the processors of a parallel machine for parallel processing.

- It is a top-down algorithm. The global features of an image are represented by large-scale monotonic lines, i.e., the monotonic lines which enclose a large area. When used for retrieving and analyzing global features, this top-down algorithm can stop further processing at those small-scale nodes to save time and space.

We first discuss the representation of monotonic trees and derive the space complexity of the representation. We then introduce our algorithm step by step. In this chapter, we let $I = (f, \Omega)$ be a fixed gray image on the hexagonal grid, and f_E be the extended function of f, unless otherwise stated. We assume that f is a function from Ω to $\{0, 1, 2, ..., d\}$, where d is the depth of the function. In computer imaging, d is usually 256. Discrete data sets from other disciplines can be quantized and then transformed to a function from a region on the hexagonal grid to $\{0, 1, 2, ..., d\}$, where the depth d depends on the precision required.

6.1 Representation of Monotonic Tree

6.1.1 Representation

The monotonic tree of an image is represented as a rooted tree, where each node represents a monotonic line. A node of a monotonic tree, denoted as $MonoNode$, has the following data items:

- type, indicating the monotonic line is outward falling or climbing;
- monoline, which is the monotonic line;
- pvalue, which is the private value of this monotonic line;
- children, which is a set of references to its child nodes.

In representing a node, the monotonic line is the dominant term. There are different ways to represent monotonic lines. The easiest way is to represent a monotonic line by its starting edgel. From its starting edgel, a monotonic line can be easily located in the image using Algorithm 6.2.1 (“Find-Child-MonoLine”), which will be given later in this chapter. This representation must be associated with the image and is not suitable for image coding and feature extraction.

Monotonic lines can also be represented by their features. For a large-scale monotonic line, we can use curve processing techniques to extract its features, such as curvature and angle. Then the monotonic line is represented by its cursive features. Small-scale monotonic lines correspond to texture or noise in the image. We can use texture or noise analysis techniques to get the statistics of shape and placement of these small-scale monotonic lines, which is the texture or noise features of the image. Then the small-scale monotonic lines are represented by these features. A monotonic tree represented this way can be used for image compression and feature extraction. The space complexity of a monotonic tree represented this way depends on the performance of the feature extraction techniques and the scale threshold which is used to separate large-scale and small-scale monotonic lines.

In this chapter, the monotonic tree computing algorithm outputs the monotonic tree such that a monotonic line is represented by the list of edgels in it. The monotonic tree represented this way can be fed to feature extraction techniques to get the features. Next we give an estimation of space complexity for the monotonic tree using this representation.

6.1.2 Space Complexity

In this sub-section, we estimate the space complexity for a monotonic tree where each monotonic line is represented by the list of edgels in it. Because the monotonic line is the dominant term in a node, the space complexity for the monotonic tree of image I is $O(\Sigma_{\partial X \in MonoLineSet(I)} |\partial X|)$, where $\Sigma_{\partial X \in MonoLineSet(I)} |\partial X|$ is the total number of edgels in all monotonic lines.

Next, we will prove that

(6.1.1) $$\Sigma_{\partial X \in MonoLineSet(I)} |\partial X| < TV(I) + 3 \times |\Omega| + d \times |\partial\Omega|,$$

where $| \bullet |$ denotes the number of elements of a set, $TV(I)$ is the total variation of image I, and d is the depth of image I.

For a continuous function h on a domain $\Phi \subseteq \mathbb{R}^2$, its total variation [61] is:

$$TV(h) = \int_\Phi |\nabla h(x, y)| dxdy.$$

For an image $I = (f, \Omega)$ on the hexagonal grid, its total variation is

$$TV(I) = \frac{1}{2} \Sigma_{(p,q) \in \Pi_h \ and \ p \in \Omega, \ q \in \Omega} |f(p) - f(q)|.$$

The coefficient $\frac{1}{2}$ is added because the dual edgels (p, q) and (q, p) are different edgels in Π_h. It's obvious that $TV(I)$ is asymptotically bounded by the image depth times the image size. For a natural image, the ratio of its total variation to its image size is usually in the range $[10, 40]$, depending on the smoothness of the image. So if the image depth is fixed, the total number of edgels in all monotonic lines is linear to the image size. Thus the space complexity of the representation of the contour tree is linear to the image size.

In deriving Formula 6.1.1, we need several theorems for the bounds of numbers of monotonic lines passing a specific edgel. Theorems 6.1.1 and 6.1.2 states that two monotonic lines have different private values if they pass a common edgel with the same or different directions. Theorem 6.1.3 states that if a monotonic line passes a edgel, then its private value is bounded by the values of the immediate interior and exterior pixels of the edgel.

These theorems imply that for an edgel (p, q), the number of monotonic lines passing this edgel is bounded by $|f(p) - f(q)| + 1$. Based on this fact, we can easily get Formula 6.1.1, which is derived in Theorem 6.1.4.

Theorem 6.1.1 *For any distinct monotonic lines ∂X and ∂Y in $MonoLineSet(I)$, if $(\partial X) \bigcap (\partial Y) \neq \emptyset$, then $PValue_I(\partial X) \neq PValue_I(\partial Y)$.*

Proof. Let $(x, y) \in (\partial X) \bigcap (\partial Y)$. So $x \in X \bigcap Y$. By Theorem 4.2.4, either $X \subset Y$ or $Y \subset X$. Without loss of generality, we suppose $X \subset Y$, which implies that ∂Y is an ancester of ∂X in the monotonic tree. Let $\{\partial X_i\}_{i=0}^{n}$ be a sequence of monotonic lines such that $X_0 = X$, $X_n = Y$, and ∂X_{i+1} is the parent of ∂X_i in the monotonic tree for any $i = 0, 1, ...,$ or $n - 1$. It's obvious that $X_0 \subset X_1 \subset ... \subset X_n$. Thus, for any $i = 0, 1, ...,$ or n, $x \in X_i$ and $y \notin X_i$. So $(x, y) \in \partial X_i$ for any $i = 0, 1, ...,$ or n. By Theorem 5.2.1, either all monotonic lines in $\{\partial X_i\}_{i=0}^{n}$ are outward falling, or all of them are outward climbing. By Theorem 5.2.8, either $PValue_I(\partial X_0) > PValue_I(\partial X_1) > ... > PValue_I(\partial X_n)$ or $PValue_I(\partial X_0) < PValue_I(\partial X_1) < ... < PValue_I(\partial X_n)$. In both cases, $PValue_I(\partial X) = PValue_I(\partial X_0) \neq PValue_I(\partial X_n) = PValue_I(\partial Y)$.

∎

Theorem 6.1.2 *For any monotonic lines ∂X and ∂Y of image I, if $(\partial X) \bigcap (\partial Y)^{-1} \neq \emptyset$, then $PValue_I(\partial X) \neq PValue_I(\partial Y)$.*

Proof. Let $(x, y) \in (\partial X) \bigcap (\partial Y)^{-1}$. So $x \in X \bigcap (\mathbf{V}_h - Y)$ and $y \in Y \bigcap (\mathbf{V}_h - X)$. By Theorem 4.2.4, $X \bigcap Y = \emptyset$. Let ∂Z be a common ancester of ∂X and ∂Y in the monotonic tree. Let $P = \{\partial X_i\}_{i=0}^{n_p}$ be a sequence of monotonic lines such that $X_0 = X$, $X_{n_p} = Z$, and ∂X_{i+1} is the parent of ∂X_i in the monotonic tree for any $i = 0, 1, ...,$ or $n_p - 1$. Let $Q = \{\partial Y_i\}_{i=0}^{n_q}$ be a sequence of monotonic lines such that $Y_0 = Y$, $Y_{n_q} = Z$, and ∂Y_{i+1} is the parent of ∂Y_i in the monotonic tree for any $i = 0, 1, ...,$ or $n_q - 1$. Let X_n be the first element in sequence P such that it's also an element in Q, i.e., X_n is the "latest" common ancester of ∂X and ∂Y. We suppose $X_n = Y_m$ for some $Y_m \in Q$. It's easy to verify that $n > 0$ and $m > 0$. Obviously,

(1) $X = X_0 \subset X_1 \subset ... \subset X_n = Y_m \supset Y_{m-1} \supset ... \supset Y_0 = Y$.

We claim that

(2) $x \in X_{n-1}$ and $y \notin X_{n-1}$.

By assumption, either $X = X_{n-1}$ if $n = 1$; or $X \subset X_{n-1}$ if $n > 1$. So $x \in X \subseteq X_{n-1}$. To prove $y \notin X_{n-1}$, we suppose otherwise. By Theorem 4.2.4, $Y \subseteq X_{n-1}$. So ∂X_{n-1} is a common ancester of both ∂X and ∂Y, contradictory to our assumption that X_n is the "latest" common ancester. The contradiction proves our claim. Similarly we can prove that

(3) $y \in Y_{m-1}$ and $x \notin Y_{m-1}$.

By statements (1)-(3), we have: $x \in X_i$ and $y \notin X_i$ for any $i = 0, 1, ...,$ or $n-1$; and $y \in Y_j$ and $x \notin Y_j$ for any $j = 0, 1, ...,$ or $m-1$. So $(x, y) \in \bigcap_{i=0}^{n-1} \partial X_i$ and $(y, x) \in \bigcap_{j=0}^{m-1} \partial Y_j$. It's obvious that $f(x) \neq f(y)$. We first suppose $f(x) > f(y)$. So by the definition of monotonic lines, for any $i = 0, 1, ...,$ or $n-1$, ∂X_i is outward falling; and for any $j = 0, 1, ...,$ or $m-1$, ∂Y_j is outward climbing. By Theorem 5.2.8, $PValue_I(\partial X_0) > PValue_I(\partial X_1) > ... > PValue_I(\partial X_n) = PValue_I(\partial Y_m) > PValue_I(\partial Y_{m-1}) > ... > PValue_I(\partial Y_0)$. So $PValue_I(\partial X) > PValue_I(\partial Y)$. In case $f(x) < f(y)$, we can prove in a similar way that $PValue_I(\partial X) < PValue_I(\partial Y)$. In both cases $PValue_I(\partial X) \neq PValue_I(\partial Y)$.

∎

Theorem 6.1.3 *For any $\partial X \in MonoLineSet(I)$ and any $(x, y) \in \partial X$, if both x and y are in Ω, then* $\min(f(x), f(y)) \leq PValue_I(\partial X) \leq \max(f(x), f(y))$.

Proof. We first suppose ∂X is outward falling. So $f(x) > f(y)$. By Theorem 5.2.7(1), $PRegion_I(\partial X) \bigcap II(\partial X) \neq \emptyset$. Let $w \in PRegion_I(\partial X) \bigcap II(\partial X)$. By the definition of monotonic lines, $f(w) > f(y)$. So $PValue_I(\partial X) = f(w) > f(y)$.

If $x \in PRegion_I(\partial X)$, then we have $PValue_I(\partial X) = f(x)$. Now we suppose $x \notin PRegion_I(\partial X)$. Let ∂Y be a descendant of ∂X such that $x \in PRegion_I(\partial Y)$. Let $\{\partial X_i\}_{i=0}^n$ be a sequence of monotonic lines such that $X_0 = Y$, $X_n = X$, and ∂X_{i+1} is the parent of ∂X_i in the monotonic tree for any $i = 0, 1, ...,$ or $n-1$. It's obvious that $X_0 \subset X_1 \subset ... \subset X_n$. Thus, for any $i = 0, 1, ...,$ or n, $x \in X_i$ and $y \notin X_i$. So $(x, y) \in \partial X_i$ for any $i = 0, 1, ...,$ or n. By Theorem 5.2.1, all monotonic lines in $\{\partial X_i\}_{i=0}^n$ are outward falling. By Theorem 5.2.8, $PValue_I(\partial X_0) > PValue_I(\partial X_1) > ... > PValue_I(\partial X_n)$. So $PValue_I(\partial X) = PValue_I(\partial X_n) < PValue_I(\partial X_0) = PValue_I(\partial Y) = f(x)$.

So we have $f(y) < PValue_I(\partial X) \leq f(x)$, given that ∂X is outward falling. In case ∂X is outward climbing, we can prove that $f(x) \leq PValue_I(\partial X) < f(y)$.

∎

Theorem 6.1.4 $\Sigma_{\partial X \in MonoLineSet(I)} |\partial X| < TV(I) + 3 \times |\Omega| + d \times |\partial\Omega|$, *where* $|\bullet|$ *denotes the number of elements of a set,* $TV(I)$ *is the total variation of image* I*, and* d *is the depth of image* I*.*

Proof. Let $\pi = \{\{x,y\}|(x,y) \in \Pi_h\}$. Here, $\{x,y\}$ is a set. Thus $\{x,y\}$ and $\{y,x\}$ are considered to be the same element of π. Let $B_\pi = \{\{x,y\} \in \pi|$ only one of x and y is in $\Omega\}$ and $N_\pi = \{\{x,y\} \in \pi|$ both of x and y are in $\Omega\}$. For any $\{x,y\}$ in $B_\pi \bigcup N_\pi$, let $M(\{x,y\})$ be the number of monotonic lines which contain (x,y) or (y,x). It's easy to check that $\Sigma_{\partial X \in MonoLineSet(I)} |\partial X| = \Sigma_{\alpha \in B_\pi} M(\alpha) + \Sigma_{\alpha \in N_\pi} M(\alpha)$, i.e., the total number of edgels in all monotonic lines is equal to the sum of numbers of monotonic lines passing each edgel. Because each hexagonal pixel only has 6 edges, it's easy to check that $|N_\pi| < 3 \times |\Omega|$.

By Theorems 6.1.1, for any $\alpha \in N_\pi$, $M(\alpha) \leq d$, where d is the depth of image I. So

(1) $\Sigma_{\alpha \in B_\pi} M(\alpha) \leq d \times |B_\pi| = d \times |\partial\Omega|$.

By Theorems 6.1.1-6.1.3, for any $\alpha = \{x, y\} \in N_\pi$, $M(\alpha) \leq |f(x) - f(y)| + 1$. Thus we have $\Sigma_{\alpha \in N_\pi} M(\alpha) \leq \Sigma_{\alpha=\{x,y\} \in N_\pi}(|f(x) - f(y)| + 1) = |N_\pi| + \Sigma_{\alpha=\{x,y\} \in N_\pi} |f(x) - f(y)| = |N_\pi| + TV(I) < 3 \times |\Omega| + TV(I)$. That is to say,

(2) $\Sigma_{\alpha \in N_\pi} M(\alpha) < 3 \times |\Omega| + TV(I)$.

By (1) and (2), the theorem is proved.

∎

6.2 Computation of Monotonic Tree

In this section, we present a top-down algorithm for computing monotonic trees. The algorithm begins with the root monotonic line, browsing the private region of the root to find all child monotonic lines. This process is repeated at each internal node to find its children. The algorithm is presented step by step in the following sub-sections.

6.2.1 Finding Private Value

An easy way to find the private value of a monotonic line is given by the following theorem, which states that the private value of a outward-falling/climbing monotonic line is the minimum/maximum of the values of the immediate interior pixels of the monotonic line.

Theorem 6.2.1

(1) For any outward-falling monotonic line ∂X in $MonoLineSet(I)$,

$PValue_I(\partial X) = \min_{x \in II(\partial X)}(f(x))$.

(2) For any outward-climbing monotonic line ∂X in $MonoLineSet(I)$,

$PValue_I(\partial X) = \max_{x \in II(\partial X)}(f(x))$.

Proof. We prove statement (1) first. Let $\partial X \in MonoLineSet(I)$ be any outward-falling monotonic line, and $p \in II(\partial X)$ be a pixel such that $f(p) = \min_{x \in II(\partial X)}(f(x))$. To prove statement (1), it suffices to prove that $p \in PRegion_I(\partial X)$. We suppose otherwise: $p \notin PRegion_I(\partial X)$. Let ∂Y be a descendant of ∂X such that $p \in PRegion_I(\partial Y)$. Let $\{\partial Y_i\}_{i=0}^{n}$ be a sequence of monotonic lines such that $Y_0 = Y$, $Y_n = X$, and ∂Y_{i+1} is the parent of ∂Y_i in the monotonic tree for any $i = 0, 1, ...,$ or $n - 1$. It's obvious that $p \in \bigcap_{i=0}^{n} II(\partial Y_i)$. By Corollary 5.2.2, all ∂Y_is are outward-falling. By Theorem 5.2.8, $PValue_I(\partial Y) = PValue_I(\partial Y_0) > PValue_I(\partial Y_1) > ... > PValue_I(\partial Y_n) = PValue_I(\partial X)$. By Theorem 5.2.7, there exists $q \in PRegion_I \bigcap II(\partial X)$. Thus $f(p) = PValue_I(\partial Y) > PValue_I(\partial X) = f(q)$, which is contradictory with our assumption that $f(p) = \min_{x \in II(\partial X)}(f(x))$. The contradiction proves (1).

In a similar way, we can prove (2).

∎

When proving the above theorem, we also proved the following corollary.

Corollary 6.2.2 *For any monotonic line $\partial X \in MonoLineSet(I)$ and any $q \in II(\partial X)$, if $f(q) = PValue_I(\partial X)$, then $q \in PRegion_I(\partial X)$.*

6.2.2 Finding A Child Monotonic Line

There are two steps to find a child monotonic line for a parent. The first step is to find an edgel in the child; the second step is to traverse the child starting from the edgel. The methods for the two steps are indicated by Theorems 6.2.4 and 6.2.5. Theorem 6.2.4 states that for a monotonic line ∂X and an edgel (x, y), if x is in the private region of the monotonic line, y is in X, and $f(y)$ is not equal to the private value of the monotonic line, then there exists one and only one child monotonic line of ∂X which contains the edgel (y, x). Based on this theorem and Theorem 5.2.7(3) ("Private Region Theorem"), an edgel of a child monotonic line can be found by browsing the private region of its parent. How to browse the private region of a monotonic line will be discussed in next sub-section.

Theorem 6.2.5 states that for an outward-falling/climbing child monotonic line, the values of its immediate exterior pixels are no more/less than the private value of the parent, and the values of its immediate interior pixels are greater/smaller than the private value of the parent. Based on this theorem, Algorithm 6.2.1 ("Find-Child-MonoLine") is presented to traverse and locate a child monotonic line, given the private value of its parent, and the starting edgel.

Lemma 6.2.3 *Let $\partial X \in MonoLineSet(I)$ be any monotonic line with at least two child monotonic lines. Let ∂Y and ∂Z be two distinct children of ∂X in the monotonic tree. Then we have $Y \bigcap Z = \emptyset$.*

This lemma can be easily proved by Theorem 4.2.4 ("No Crossing Theorem") and the definition of monotonic trees.

Theorem 6.2.4 *For any monotonic line $\partial X \in MonoLineSet(I)$ and any edgel $(y, x) \in \Pi_h$, if $x \in PRegion_I(\partial X)$, $y \in X$, and $f(y) \neq PValue_I(\partial X)$, then there exists one and only one child monotonic line ∂Y of ∂X such that $(y, x) \in \partial Y$. In addition, $f(y) > PValue_I(\partial X)$ iff ∂Y is outward falling.*

Proof. Let $\partial X \in MonoLineSet(I)$ be any monotonic line and (y, x) be an edgel such that $x \in PRegion_I(\partial X)$, $y \in X$, and $f(y) \neq PValue_I(\partial X)$.

By Theorem 5.2.6, there exists $\partial Z \in MonoLineSet(I)$ such that either $(y, x) \in \partial Z$ or $(x, y) \in \partial Z$. By Theorem 4.2.4, $Z \subset X$. It's impossible that $(x, y) \in \partial Z$, bacause $(x, y) \in \partial Z \implies x \in Z \implies x \notin PRegion_I(\partial X)$. So $(y, x) \in \partial Z$. Let $\Phi = \{\partial Z \in MonoLineSet(I) | (y, x) \in \partial Z\}$. So Φ is not empty. Let ∂Y be an element of Φ such that $Area(Y) = \max_{\partial Z \in \Phi} Area(Z)$, where $Area(\bullet)$ denotes the area of an region. By Theorem 4.2.4, $Y \subset X$, i.e., ∂Y is a descendant of ∂X.

We claim that ∂Y is a child of ∂X. To prove the claim, we suppose otherwise. So there exists a child ∂Z of ∂X such that $Y \subset Z$. Because $x \in PRegion_I(\partial X)$, $x \notin Z$. Because $Y \subset Z$, $y \in Z$. So $(y, x) \in \partial Z$. Thus $\partial Z \in \Phi$. By our assumption about ∂Y, $Area(Y) \geq Area(Z)$, which is contradictory with our assumption that $Y \subset Z$. This contradiction proved that ∂Y is a child of ∂X.

We claim that ∂Y is the only child of ∂X such that $(y, x) \in \partial Y$. We suppose otherwise: there is another child ∂Z of ∂X such that $(y, x) \in \partial Z$. So we have $y \in Y \bigcap Z$, which is contradictory with Lemma 6.2.3. The contradiction proved that ∂Y is the only child of ∂X such that $(y, x) \in \partial Y$.

By the definition of monotonic lines, $f(y) > f(x) \iff \partial Y$ is outward falling. Because $f(x) = PValue_I(\partial X)$, $f(y) > PValue_I(\partial X) \iff \partial Y$ is outward falling.

∎

Theorem 6.2.5 *For any monotonic line $\partial X \in MonoLineSet(I)$ and any child ∂Y of ∂X,*

(1) if ∂Y is outward falling, then $\forall x \in IE(\partial Y), f_E(x) \leq PValue_I(\partial X)$ and $\forall y \in II(\partial Y), f_E(y) > PValue_I(\partial X)$; and

(2) if ∂Y is outward climbing, then $\forall x \in IE(\partial Y), f_E(x) \geq PValue_I(\partial X)$ and $\forall y \in II(\partial Y), f_E(y) < PValue_I(\partial X)$.

Proof. We prove (1) first. Let $\partial X \in MonoLineSet(I)$ be any monotonic line, and ∂Y be a child of ∂X. We assume that ∂Y is outward falling. By Theorem 5.2.7(3), there exists $w \in IE(\partial Y) \bigcap PRegion_I(\partial X)$. By the definition of monotonic lines, $\forall y \in II(\partial Y), f_E(y) > f_E(w) = PValue_I(\partial X)$.

To prove that $\forall x \in IE(\partial Y), f_E(x) \leq PValue_I(\partial X)$, we suppose otherwise, i.e., there exists $x \in IE(\partial Y)$ such that $f_E(x) > PValue_I(\partial X)$. Let $y \in II(\partial Y)$ be a pixel such that $(y, x) \in \partial Y$. Then we discuss two cases (i) $x \notin X$ and (ii) $x \in X$.

In the first case, because $x \notin X$ and $y \in Y \subset X$, $(y, x) \in \partial X$. By Theorem 5.2.1, ∂X is outward falling. By Theorem 5.2.7, there exists $z \in PRegion_I(\partial X) \bigcap II(\partial X)$. Because $(y, x) \in \partial X$, by the definition of monotonic lines, $f_E(x) < f_E(z) = PValue_I(\partial X)$, which is contradictory to our assumption that $f_E(x) > PValue_I(\partial X)$.

In the second case, $x \in X$. Because $f_E(x) > PValue_I(\partial X)$, $x \notin PRegion_I(\partial X)$. So there exists a child ∂Z of ∂X such that $x \in Z$. By Lemma 6.2.3, $y \notin Z$. So $(x, y) \in \partial Z$. By Theorem 5.2.3, ∂Z is outward climbing. By Theorem 5.2.7(3), there exists $z \in PRegion_I(\partial X) \bigcap IE(\partial Z)$. So by the definition of monotonic lines, $f_E(x) < f_E(z) = PValue_I(\partial X)$, which is contradictory to our assumption that $f_E(x) > PValue_I(\partial X)$.

In both cases, we get a contradiction, which proved (1). In a similar way, we can prove (2).

∎

The above theorem gives us a way to traverse a child monotonic line, which is illustrated in

Algorithm 6.2.1. The input to the algorithm is the image I, the private value v of a parent monotonic line, and a starting edgel α of a child. The output is a circular list of edgels in the child monotonic line. A circular list is a data structure which maintains a list of data items with no beginning and ending. It supports such operations as appending and browsing. The algorithm takes time linear to the length of the child monotonic line.

In the algorithm, v_α is the value of an immediate interior pixel of the child monotonic line. By Theorem 6.2.5, v_α is greater/smaller than v iff the child monotonic line is outward-falling/climbing. Lines 8 to 13 compute the next edgel for the traversal. There are two choices for the edgel next to β in the child monotonic line: $next_p(\beta)$ and $(next_q^{-1}(\beta^{-1}))^{-1}$. Which one to choose depends on whether pixel $IEP(next_p(\beta))$ is in the interior or exterior of the child monotonic line, which can be determined by Theorem 6.2.5. See Figure 6.2.1.

Algorithm 6.2.1 *Find-Child-MonoLine(*I, v, α*)*

1. *childline* $\leftarrow$ *new CircularList*
2. $\beta \leftarrow \alpha$
3. $v_\alpha \leftarrow f(IIP(\alpha))$
4. ***do***
5.   *append* β *to childline*
6. $p \leftarrow IIP(\beta)$
7. $q \leftarrow IEP(\beta)$
8.   ***if*** $v_\alpha > v$ ***then***

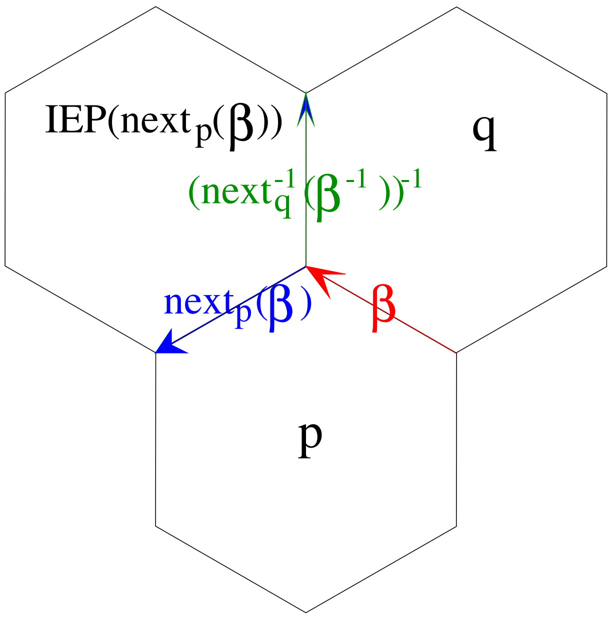


Figure 6.2.1: Finding the next edgel along a child monotonic line.

    if $f_E(IEP(next_p(\beta))) \leq v$ ***then*** $\beta \leftarrow next_p(\beta)$
    else $\beta \leftarrow (next_q^{-1}(\beta^{-1}))^{-1}$
  else
    if $f_E(IEP(next_p(\beta))) \geq v$ ***then*** $\beta \leftarrow next_p(\beta)$
    else $\beta \leftarrow (next_q^{-1}(\beta^{-1}))^{-1}$
while $\beta \neq \alpha$
return childline

6.2.3 Finding All Child Monotonic Lines

Introduction

As we said, child monotonic lines can be found by browsing the private region of its parent. However, the private region of a monotonic line may not be connected, and some of its components may even not be adjacent to the monotonic line. See Figure 6.2.2. In this example, the private region of the root, shown in yellow, has two components, and the one at the center is isolated by the child monotonic lines. This example shows that it's impossible to identify the private region of a monotonic line before its children are located.

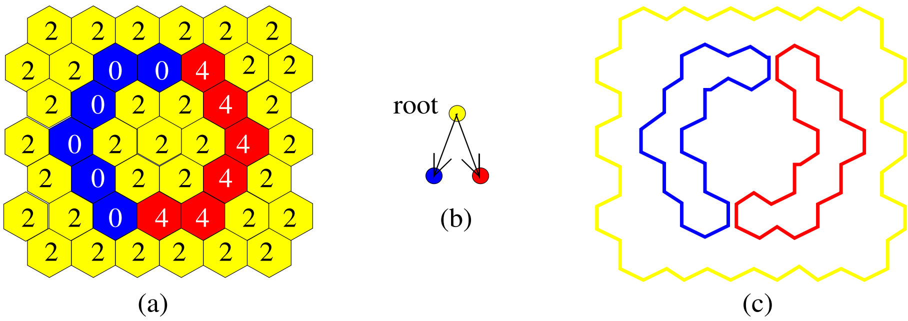


Figure 6.2.2: An example of monotonic trees: (a) the image (pixel values are marked in the pixels), (b) the monotonic tree of the image, (c) the three monotonic lines of the image.

Our solution to the above problem is to treat the child monotonic lines and the pixels in the private region as the vertices of a graph, which are connected by edgels among them. The child monotonic lines are found by a breath-first search of the graph. Formally, we give a definition for the *children-pregion graph*.

Definition 6.2.1 *Let ∂X be the parent monotonic line, and $\{\partial Y_i\}_{i=1}^n$ be the set of its children. The* ***children-pregion graph*** *is represented by its vertex and edge sets: $G =< V, E >$, where*

$$\begin{aligned} V &= PRegion_I(\partial X) \bigcup \{Y_i\}_{i=1}^n;\ and \\ E &= \{(p,q) \in \Pi_h \mid p \in PRegion_I\ and\ q \in PRegion_I\} \bigcup \\ &\quad \{(p,q) \in \Pi_h \mid p \in Y_i\ and\ q \in Y_j,\ for\ i,j \in \{1,2,...,n\}\ with\ i \neq j\} \bigcup \\ &\quad \{(p,q) \in \Pi_h \mid p \in PRegion_I\ and\ q \in Y_i,\ for\ some\ i \in \{1,2,...,n\}\} \bigcup \\ &\quad \{(p,q) \in \Pi_h \mid p \in Y_i\ and\ q \in PRegion_I,\ for\ some\ i \in \{1,2,...,n\}\}. \end{aligned}$$

It's easy to see that the graph is connected.

Our algorithm (given later in Algorithm 6.2.2) begins the search with a pixel in the intersection of the private region and immediate interior of the parent ∂X. By Theorem 5.2.7(1) ("Private Region Theorem"), such a pixel always exists. During the search, for each pixel in the private region of ∂X, we check each of its six neighbors which are within X: if the value of the neighbor is equal to the private value of ∂X, then by Theorem 5.2.6(1) ("Separation Theorem"), the neighbor is also in the private region; otherwise, by Theorem 6.2.4, the neighbor is in the interior of a child monotonic line, which is then located by Algorithm 6.2.1 ("Find-Child-MonoLine"). For each child monotonic line encountered during the search, each of its immediate exterior pixels which are within X is checked: if its value is equal to the private value of ∂X, then by Theorem 6.2.6 (given next), the pixel is in the private region; otherwise, by Theorem 6.2.7 (given later), the pixel is in the interior of another child monotonic line, which is then located by Algorithm 6.2.1. After the breath-first search of the children-pregion graph is done, all child monotonic lines are found.

Theorems 6.2.6 and 6.2.7 are presented next. Following the two theorems is Algorithm 6.2.2 ("Find-All-Child-MonoLines"), which are followed by an explanation. An analysis of the time complexity concludes this sub-section.

Theorems

Theorem 6.2.6 *For any monotonic line $\partial X \in MonoLineSet(I)$ and any child ∂Y of ∂X, $\{x \in IE(\partial Y) | f_E(x) = PValue_I(\partial X)\}$ is a subset of $PRegion_I(\partial X)$.*

Proof. Let $\partial X \in MonoLineSet(I)$ be any monotonic line, and ∂Y be a child of ∂X. Let x be an arbitrary pixel in $IE(\partial Y)$ such that $f_E(x) = PValue_I(\partial X)$. We need to prove that $x \in PRegion_I(\partial X)$. We suppose otherwise, and consider two cases: (1) $x \notin X$ and (2) $x \in X - PRegion_I(\partial X)$.

In the first case, $x \notin X$. Because $x \in IE(\partial Y)$, $x \in IE(\partial X)$. Let w be any element in $PRegion_I(\partial X) \bigcap II(\partial X)$. By the definition of monotonic lines, no matter whether ∂X is outward falling or climbing, $f_E(x) \neq f_E(w) = PValue_I(\partial X)$, which is contradictory to our assumption that $f_E(x) = PValue_I(\partial X)$.

In the second case, $x \in X - PRegion_I(\partial X)$. Let ∂Z be a child of ∂X such that $x \in Z$. Let y be a pixel in $II(\partial Y)$ such that $(y, x) \in \partial Y$. By Lemma 6.2.3, $y \notin Z$. So $x \in II(\partial Z)$. By Theorem 6.2.5, no matter whether ∂Z is outward falling or climbing, $f_E(x) \neq PValue_I(\partial X)$, which is contradictory to our assumption that $f_E(x) = PValue_I(\partial X)$.

In both cases, we get a contradiction, which proved that for any $x \in \{x \in IE(\partial Y) | f_E(x) = PValue_I(\partial X)\}$, $x \in PRegion_I(\partial X)$. So $\{x \in IE(\partial Y) | f_E(x) = PValue_I(\partial X)\}$ is a subset of $PRegion_I(\partial X)$.

∎

Theorem 6.2.7 *Let $\partial X \in MonoLineSet(I)$ be any monotonic line, and ∂Y be a child of ∂X. For any edgel $(y, z) \in \partial Y$, if $z \in X$ and $f_E(z) \neq PValue_I(\partial X)$, then there exists one and only one child monotonic line ∂Z of ∂X such that $(z, y) \in \partial Z$. In addition, $f_E(z) > PValue_I(\partial X)$ iff ∂Z is outward falling.*

Proof. Because $z \in X$ and $f_E(z) \neq PValue_I(\partial X)$, $z \in X - PRegion_I(\partial X)$. By the definition of private regions, there exists one child monotonic line ∂Z of ∂X such that $z \in Z$. By Lemma 6.2.3, $y \notin Z$. So $(z, y) \in \partial Z$.

If there is another child ∂W of ∂X such that $(z, y) \in \partial W$, then $z \in W \bigcap Z$, which is contradictory with Lemma 6.2.3. The contradiction proved that ∂Z is the only child of ∂X such that $(z, y) \in \partial Z$.

Because $z \in II(\partial Z)$, by Theorem 6.2.5, $f_E(z) > PValue_I(\partial X) \iff \partial Z$ is outward falling.

∎

Algorithm

Algorithm 6.2.2 *Find-All-Child-MonoLines(I, v, p, FP)*

1. *childlineset* $\leftarrow$ *new Set*
2. *Q* $\leftarrow$ *new FIFOQueue*
3. ***for** each* $e \in \{e_i(p)\}_{i=0}^{5}$ ***do***
4.   ***if*** $FP(e) = false$ ***then*** *append* e *to* Q
5.   *FP(e)=true*
6. ***while*** $Q \neq \emptyset$ ***do***
7.   *pop head of* Q *to* α
8.   ***if*** *FP(α^{-1})=true* ***then goto*** *Line 6*
9. $q \leftarrow IEP(\alpha)$
10.   ***if*** $f(q) = v$ ***then***
11.   ***for** each* $e \in \{e_i(q)\}_{i=0}^{5}$ ***do***
12.   ***if*** $FP(e) = false$ *and* $FP(e^{-1}) = false$ ***then*** *append* e *to* Q
13.   *FP(e)=true*
14.   ***else***
15.   *childline* $\leftarrow$ *Find-Child-MonoLine(I, v, α^{-1})*
16.   ***for** each* $e \in childline$ ***do***

if $FP(e) = false$ *and* $FP(e^{-1}) = false$ ***then*** *append e to Q*

FP(e)=true

add childline to childlineset

delete Q

return childlineset

The algorithm above takes as input the image I, the private value v of a parent monotonic line, a pixel p in the parent's private region, and a function FP from the set of edgels within the image domain to $\{true, false\}$. The output of the algorithm is a set of child monotonic lines, each of which is represented by a list of edgels in it. The function FP is for indicating whether the edgels are processed or not. Initially, FP is $true$ for all edgels in the parent monotonic line, and $false$ for all edgels within the interior of the parent. When running, the algorithm will assign $true$ to $FP(\alpha)$ after the edgel α being processed. The pixel p is a pixel in the intersection of the private region and immediate interior of the parent. How to find such a pixel will be discussed later.

The algorithm makes a breath-first search over the children-pregion graph. It has two major differences with the popular breath-first search algorithm in Pages 469-472 of [12]. The first difference is that the first-in-first-out queue Q in our algorithm records the edges to new vertices; the second is that, instead of coloring the discovered, discovering, and undiscovered vertices by black, gray, and white, respectively, we mark their edges as processed or unprocessed. The reason for the differences is because the children-pregion graph has heterogeneous vertices, including pixels and regions, which makes it inconvenient to uniformly represent vertices.

The algorithm begins the search with the pixel p, appending to the first-in-first-out queue Q with each edgel $e = (p, q)$ of p such that q is within the interior of the parent monline, which is is indicated by $FP(e) = false$. Initially, $FP(e) = true$ means that e is in the parent monline and that q is in the exterior of the parent monline. Then the six edgels of p will be marked as $true$ in FP. When the algorithm is running, The first-in-first-out queue Q maintains a list of edgels whose immediate exterior pixels will be processed. The edgels in Q represent edges of the children-pregion graph which point to the new vertices waiting to be processed.

In the main loop (Lines 6 to 19), the head of the queue Q is popped to α, whose immediate exterior pixel q will then be checked. If $FP(\alpha^{-1})$ is $true$, then q will be skipped, because it has been processed before. [1] Otherwise, the value of q indicates whether it's in the private region or not. If $f(q)$ equals to the private value v, then q is in the private region and each of six its edgels e will be added to Q if $FP(e)$ and $FP(e^{-1})$ are $false$, where $FP(e) = true$ means e is in the parent monotonic line, and $FP(e^{-1}) = true$ means $IEP(e)$ has been processed. If $f(q)$ doesn't equal to the private value v, then q is in the interior of a child monotonic line, which will be located by Algorithm 6.2.1 ("Find-Child-MonoLine") and

[1] Even though the vertex of the children-pregion graph pointed to by the edge α was not processed when α was added to Q. It may have been processed through another edge pointed to the vertex.

added to the child set $childlineset$. Each edgel e in the child monotonic line is added to Q if $FP(e)$ and $FP(e^{-1})$ are $false$. All edgels being processed in the main loop will be marked as $true$ in FP. When the main loop is over, the child set records the set of all child monotonic lines.

The algorithm can be easily augmented to mark the private regions of monotonic lines, which are sometimes needed by higher-level processing.

Running Time

To analyze the running time, we assume that ∂X is the parent monotonic line, and $\{\partial Y_i\}_{i=1}^n$ is the set of its children. Now consider the set

$$\tilde{E} = (\bigcup_{q \in PRegion_I(\partial X)} \{e_i(q)\}_{i=0}^5) \bigcup (\bigcup_{i=1}^n \partial Y_i).$$

A careful calculation will show that $\tilde{E} = E \bigcup \partial X$, where the set E is given in Definition 6.2.1. Obviously, the number of elements of $\tilde{E}$ is bounded by:

$$|\tilde{E}| \ \leq \ 6 \times |PRegion_I(\partial X)| \ + \ \Sigma_{i=1}^n |\partial Y_i| = O(|PRegion_I(\partial X)| + \Sigma_{i=1}^n |\partial Y_i|),$$

where $| \bullet |$ denotes the cardinality of a set.

In the algorithm, the operations for Lines 1 to 5 and Lines 20 to 21 take $O(1)$ time. Most of the running time is spent for the main loop (Lines 6 to 19). Each time the main loop is executed, the edgel α (popped from the queue Q) is different and is an element of $\tilde{E}$.

So the number of the executions of the main loop is bounded by $|\tilde{E}|$. In one execution of the main loop, Lines 6 to 9 are executed at most once, taking $O(1)$ time. So the total time for executing Lines 6 to 9 is $O(|\tilde{E}|)$. Lines 11 to 13 are for pixels in the private region, taking time $O(|PRegion_I(\partial X)|)$ in total. Lines 15 to 19 are for child monotonic lines, taking time $O(\Sigma_{i=1}^{n}|\partial Y_i|)$ in total. So the running time for the main loop is $O(|\tilde{E}|) + O(|PRegion_I(\partial X)|) + O(\Sigma_{i=1}^{n}|\partial Y_i|) = O(|PRegion_I(\partial X)| + \Sigma_{i=1}^{n}|\partial Y_i|)$. The running time for the algorithm is also $O(|PRegion_I(\partial X)| + \Sigma_{i=1}^{n}|\partial Y_i|)$.

6.2.4 Constructing the Monotonic Tree

Based on Algorithm 6.2.2 ("Find-All-Child-MonoLines"), the monotonic tree can be constructed recursively. For the root monotonic line $\partial\Omega$ of the image I, we create a node for it, find its child monotonic lines, and create nodes for its children. This process is repeated recursively until the nodes are created for all monotonic lines. The pseudo code for the construction is given in Algorithms 6.2.3 and 6.2.4, attached at the end of this sub-section.

Algorithm 6.2.3 takes as input the image I, a monotonic line ML which is a list of edgels in the monotonic line, and the function FP from the set of edgels within the image domain to $\{true, false\}$. It outputs a subtree starting from the monotonic line. The subtree is referred by the node of the monotonic line. In Lines 1 to 2 of the algorithm, a node is created for the monotonic line ML. In Lines 3 to 16, information about the monotonic line ML, such as its private value (represented by variable v in the code) and whether it's outward-falling or outward-climbing, is obtained and stored to the node, on the basis of the definition of monotonic lines and Theorem 6.2.1. In Lines 3 to 16, a pixel in the private region, represented by variable p in the code, is also found, on the basis of Corollary 6.2.2. Then Algorithm 6.2.2 is called to find all child monotonic lines, and subtrees from the children are constructed recursively. When running, Lines 1 to 16 take $O(|ML|)$ time. Line 17 takes $O(|PRegion_I(ML)| + \Sigma_{CL\in ChildSet(ML)}|CL|)$ time, where $ChildSet(ML)$ is the set of children of ML. Lines 18 to 19 take $O(\Sigma_{CL\in DescendentSet(ML)}(|PRegion_I(CL)| + |CL|))$ time, where $DescendentSet(ML)$ is the set of descendents of ML. So the total running time of the algorithm is $O(|ML| + |Interior(ML)| + \Sigma_{CL\in DescendentSet(ML)}|CL|)$, where $Interior(ML)$ is the interior of ML.

Algorithm 6.2.4 computes the monotonic tree for the image I. It first constructs and initializes the function

$$FP : \Pi_\Omega = \bigcup_{p\in\Omega}\{e_i(p)\}_{i=0}^{5} \to \{false, true\},$$

and then calls Algorithm 6.2.3 to construct a tree from the root monotonic line $\partial\Omega$. When the algorithm is running, the function FP is for marking a monotonic line which will be processed and for indicating whether edgels are processed or not. The running time of Algorithm 6.2.4 is $O(|\Omega| + \Sigma_{\partial X\in MonoLineSet(I)}|\partial X|)$. The algorithm is optimal, because any monotree algorithm has to browse the image, taking $O(|\Omega|)$ time, and to find and record all monotonic lines, taking $O(\Sigma_{\partial X\in MonoLineSet(I)}|\partial X|)$ time. The space used in the algorithm can't exceed the time needed. So the space complexity is also $O(|\Omega| + \Sigma_{\partial X \in MonoLineSet(I)} |\partial X|)$. As we discussed before, if the image depth is fixed, then the second term $\Sigma_{\partial X \in MonoLineSet(I)} |\partial X|$ is linear to the image size $|\Omega|$. So the algorithm takes time and space linear to the image size. Once the monotonic tree is constructed, the topological monotonic tree can be easily obtained by merging the monotonic slopes into new nodes. Listed below are Algorithms 6.2.3 and 6.2.4.

Algorithm 6.2.3 *Create-Tree(I, ML, FP)*

$node \leftarrow$ *new MonoNode*
$node$*−>monoline* $\leftarrow$ *ML*
$\alpha \leftarrow$ *head(ML)*
if $f_E(IIP(\alpha)) > f_E(IEP(\alpha))$ ***then***
    $node$*−>type* $\leftarrow$ *outward-falling*
    $v \leftarrow +\infty$
    for *each* $\beta \in ML$ ***do***
        $v \leftarrow \min(v, f_E(IIP(\beta)))$
        if $v = f_E(IIP(\beta))$ ***then*** $p \leftarrow IIP(\beta)$
else
    $node$*−>type* $\leftarrow$ *outward-climbing*
    $v \leftarrow -\infty$
    for *each* $\beta \in ML$ ***do***
        $v \leftarrow \max(v, f_E(IIP(\beta)))$
        if $v = f_E(IIP(\beta))$ ***then*** $p \leftarrow IIP(\beta)$

node−>pvalue $\leftarrow v$

childlineset $\leftarrow$ *Find-All-Child-MonoLines(*I, v, p, FP*)*

for *each* $line \in childlineset$ ***do***

    add Create-Tree($I, line, FP$*) to node−>children*

return node

Algorithm 6.2.4 *MonoTree(*I*)*

FP $\leftarrow$ *new Function from* $\Pi_\Omega = \bigcup_{p\in\Omega}\{e_i(p)\}_{i=0}^{5}$ *to* $\{false, true\}$

for *each* $\alpha \in \Pi_\Omega$ ***do*** $FP(\alpha) \leftarrow false$

for *each* $\alpha \in \partial\Omega$ ***do*** $FP(\alpha) \leftarrow true$

return Create-Tree($I, \partial\Omega, FP$*)*

Chapter 7

Reduced Monotonic Tree

For natural images, their monotonic trees and topological monotonic trees have too many nodes, which causes inconvenience for higher-level processing. In this chapter, we introduce a new concept *reduced monotonic tree*, which has been used to retrieve scenery features from images [83, 81, 82] and locate smooth background in images [85].

7.1 Definition

In the monotonic tree of a gray image, the leaf nodes correspond to the boundaries of the local maximum/minimum regions in the image. Starting from the leaf nodes in the monotonic tree, we find the monotonic slopes which contain these leaf nodes. These monotonic slopes correspond to the leaf nodes in the topological monotonic tree, which are called the *slopes at level one*. By removing these monotonic slopes, we produce a smoothed gray image. We then apply the same process to the smoothed image to generate the *slopes at level two*. The process is repeated recursively to produce a hierarchy of slopes. Together, these slopes form a *reduced monotonic tree*. The parent-child relation in the reduced monotonic tree is defined in a similar manner to that for the topological monotonic tree.

A reduced monotonic tree has such a property that all leaf nodes have the same distance (in terms of the length of a shortest path in the tree) to the root. The distance from the leaf nodes to the root is called the *depth* of the reduced monotonic tree. For a natural image of size around 300×250, the depth of its reduced monotonic tree is usually less than 10 and greater than 6.

7.2 Computation

The reduced monotonic tree of an image can be obtained from the monotonic tree of the image by extracting monotonic slopes from the monotonic tree at different levels. In this section, we introduce a bottom-up algorithm to compute a reduced monotonic tree. The algorithm is based on the definition of reduced monotonic trees. At each level, the algorithm extracts monotonic slopes and smoothes the image. When the image becomes a constant function, all monotonic slopes are removed and the reduced monotonic tree is constructed. The running time at each level is linear to the image size. So the total running time is $O(d \times n)$, where d is the depth of the reduced monotonic tree and n is the image size. Given the fact that the depth of a reduced monotonic trees is often a small number, [1] the total running time is linear to the image size.

Figures 7.2.1 and 7.2.2 illustrate the algorithm for generationing reduced monotonic trees. The algorithm consists of the following steps:

(1) Transfer the image from the square grid to the hexagonal grid [2], Denote the image on the hexagonal grid as $I = (f, \Omega)$, where Ω is the domain of the image and f is a function from Ω to $\mathbb{R}$. Let f_E be the extended function of f to the whole plane, which is equal to f on Ω, and assumes $-\infty$ out of Ω.

[1] In my experiments with thousands of natural images of variable sizes, I never found a image whose reduced monotonic tree has depth greater than 10.

[2] The transformation is discussed in Sub-section 4.1.1.

(2) Find simply-connected regions where f_E assumes local minimum or maximum values, as shown in Figure 7.2.1(a).

(3) For each local minimum/maximum, simply-connected region X, find the slope including ∂X, where a slope is a sequence of monotonic lines with each line uniquely and directly enclosing the next, as shown in Figure 7.2.1(b). The procedure for finding the slope including ∂X is as follows:

   (a) let $Y = X$ and $S = \{\partial Y\}$;

   (b) if $Y = \Omega$, output S and stop;

   (c) if ∂Y is outward-falling/climbing, let v be the highest/lowest value assumed by pixels [3] adjacent to Y;

   (d) let $\{Z_i\}_{i=1}^n$ be the set of maximal connected regions such that (i) each Z_i is adjacent to Y, and (ii) f_E assumes the constant value v over each Z_i. Let $Z = \bigcup_{i=1}^n Z_i$;

   (e) if $\partial(Y \bigcup Z)$ is not a monotonic line, output S and stop;

   (f) let $Y = Y \bigcup Z$ and $S = S \bigcup \{\partial Y\}$; go to step (b).

   All these slopes are nodes of the reduced monotonic tree, as shown in Figure 7.2.2(a).

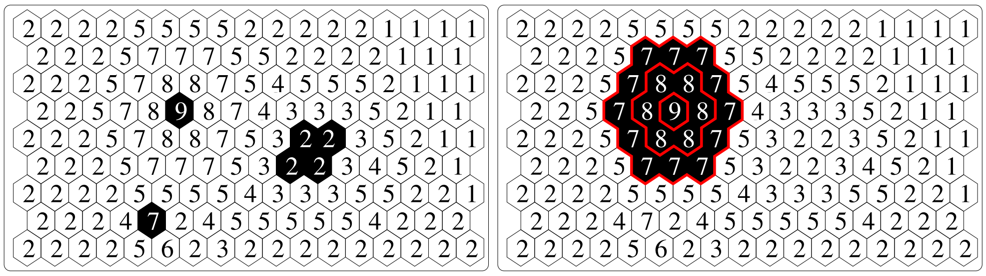
(4) If only one slope is found in step (3) and it covers the whole image, stop. Otherwise, smooth the image by removing all the slopes found in step (3). Let $\{X_i\}_{i=1}^n$ be the

[3] Here, by "a pixel assumes value v", we mean that f_E assumes v over this pixel.

set of regions covered by those slopes, respectively. For each X_i, let v_i be the highest/lowest value assumed by pixels adjacent to X_i if ∂X_i is outward-falling/climbing. The smoothed image is a function $\tilde{f} : \Omega \to \mathbb{R}$ such that:

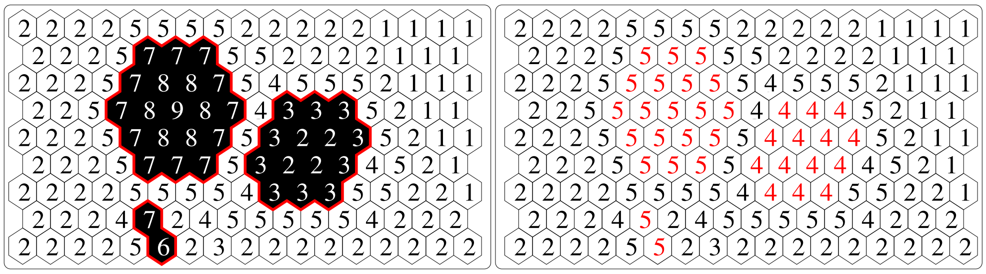
$$\tilde{f}(x) = \begin{cases} v_i & \text{if } x \in X_i \text{ for some } i; \\ f(x) & \text{otherwise.} \end{cases} \tag{7.2.1}$$

Figure 7.2.2(b) shows the smoothed image for our example. Let $f = \tilde{f}$ and f_E be the extended function of f. Go to step (2).



(a) (b)

Figure 7.2.1: (a) Simply-connected regions where function f_E assumes local minimum or maximum values, (b) a slope with respect to a local maximum region (the red lines are the monotonic lines in the slope).



(a) (b)

Figure 7.2.2: (a) All slopes with respect to local minimum/maximum regions, (b) the smoothed image.

7.3 Examples

The image for the first example is given in Figure 7.3.1 (a), its reduced monotonic tree is represented by its elements (nodes) at levels from 1 to 7, which are visually represented in Figure 7.3.1 (b)-(h), respectively. In these pictures, an outward-falling/climbing monotonic slope is shown by the white/black area which is enclosed by this slope. The blank area is shown in gray. The root node of the reduced monotonic tree of this example is at level 8. The root node covers the whole image. A second example is given in Figure 7.3.2.

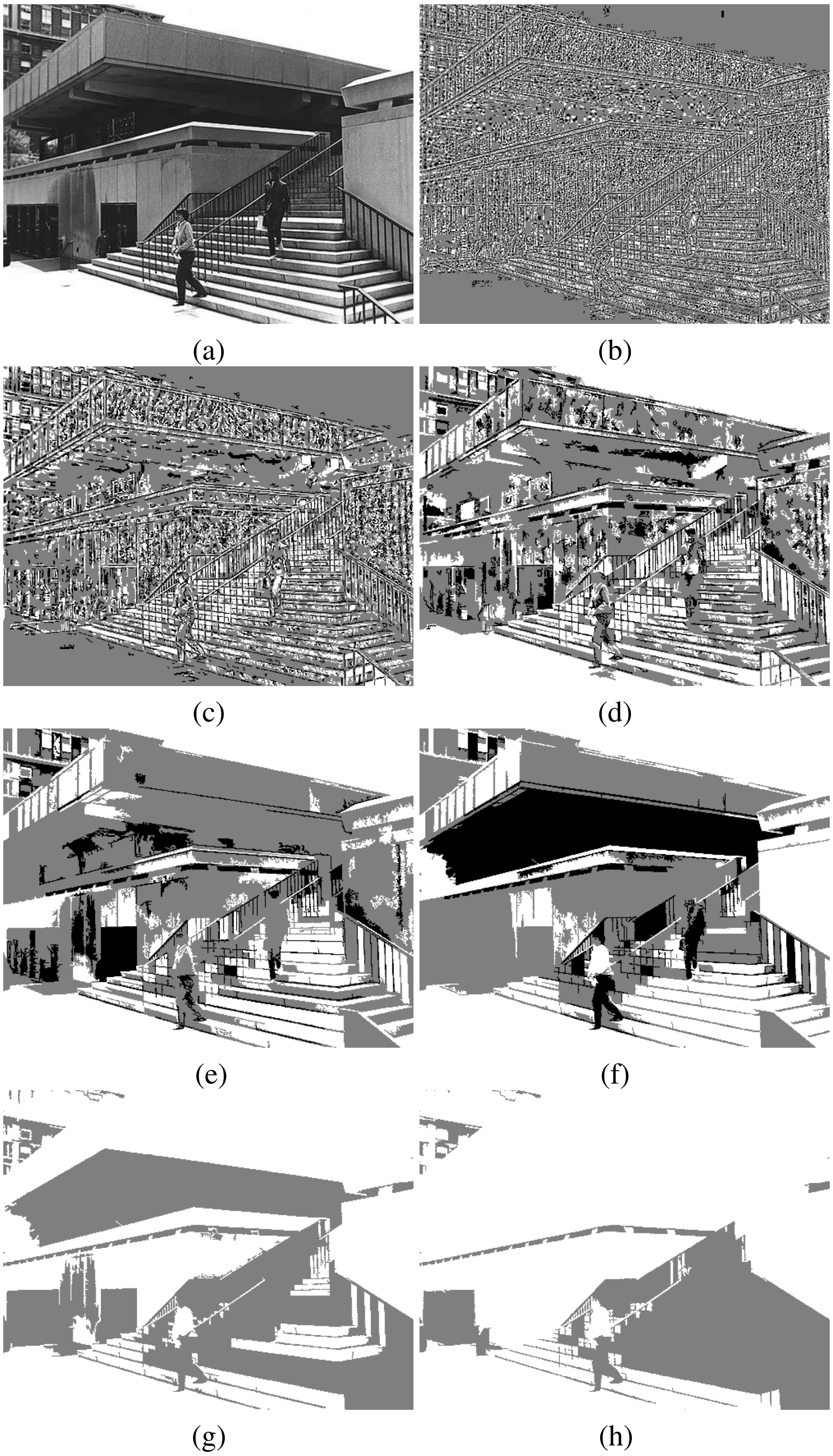
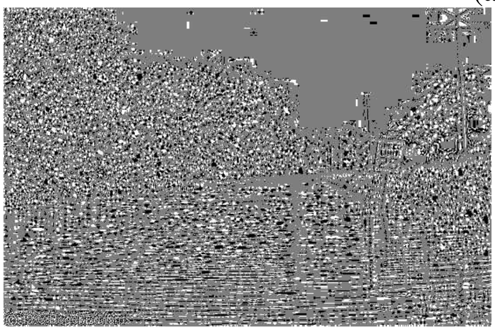


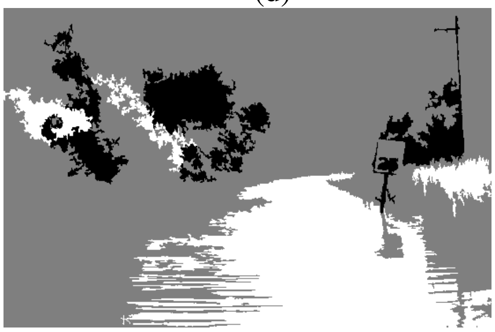
Figure 7.3.1: An example of reduced monotonic trees.

(a)



(b) (c)

(d) (e)



(f) (g)

Figure 7.3.2: An example of reduced monotonic trees: (a) the original image, (b)-(g) the monotonic slopes at levels 1 to 6. The root slope is at level 7.

Chapter 8

SceneryAnalyzer: System Framework

Monotonic trees are used as a hierarchical representation of image structures in image retrieval. On the basis of monotonic trees, a system, termed *SceneryAnalyzer*, was implemented to support semantics-based image retrieval. The system is available on line at "http://monet.cse.buffalo.edu:8888/". The overall architecture is illustrated in Figure 8.0.3. The system has three major components: (1) image databases, (2) feature extraction, and (3) query interface. This chapter gives an introduction to the three components.

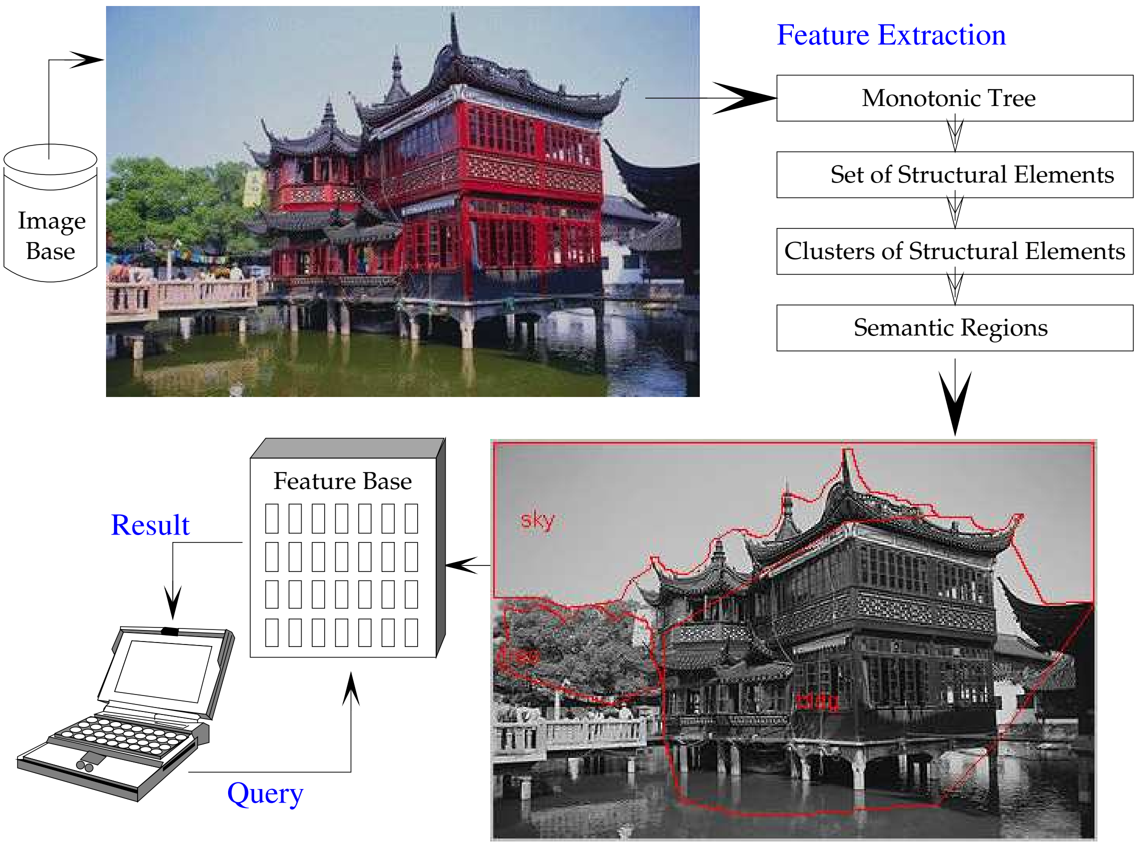


Figure 8.0.3: *SceneryAnalyzer* architecture.

8.1 Image Databases

We used two image databases: COREL image base and PhotoDisc image base. The COREL image base consists of over 30,000 COREL images from CD7 and CD8 of Corel Gallery 1,300,000 (Search ID:Gallery13M_01). These images are stored in JPEG format with the sizes either 384×256 or 256×384. The COREL image base has a wide variety, including both scenery pictures (taken at globally diverse locations) and non-scenery images (covering various subjects, such as fish, molecules, space scenes, insects, and other topics).

Our 1444 PhotoDisc images were from PhotoDisc Comping Disc 3 and Disc 4. They fall into five categories: homes-and-gardens, international sports, nature scenes, panoramic landscapes, and people-lifestyles-and-vacations. There images are stored in JPEG format with various sizes.

8.2 Feature Extraction

The feature extraction is the central component of the system, which makes *SceneryAnalyzer* different from other systems. The feature extraction is done off-line. During feature extraction, each image is processed to extract semantic features. The semantic features are automatically annotated in the gray copy of the original image. SceneryAnalyzer handles smooth background and six types of scenery features: sky, building, tree, water wave, placid water, and ground. The visual properties of water wave and placid water are structurally different, and they are therefore treated as separate types. The ground feature in images can be further split into snow, lawn, and other subtypes.

The feature extraction is based on the concept of the reduced monotonic tree. The structural elements of an image are modeled as branches (or subtrees) of the reduced monotonic tree of the image. These structural elements are classified and clustered on the basis of such properties as color, spatial location, coarseness, and shape. Each cluster corresponds to some semantic feature. Following these steps, images can be automatically annotated with category keywords, thus facilitating high-level (semantics-based) image querying and browsing.

Feature extraction consists of three consecutive steps: classifying structural elements; clustering structural elements; and rendering semantic regions. After the region rendering step, the features of the input image are extracted and stored in a feature vector. A feature vector is an array which records the qualifying scores for all semantic features. The qualifying score of an image with respect to a feature indicates the degree that the image can be considered to have this feature. It's the sum of the qualifying scores of all regions rendered in the image with respect to the feature. All feature vectors are stored in a vector base.

8.3 Query Interface

Figure 8.3.1 illustrates the interface of SceneryAnalyzer. To retrieve images, a user can select any combination of the listed features. After the query is submitted, the icons of images with the selected features are output in pages (Figure 8.3.2). The output images are sorted by their qualifying scores with respect to the submitted list of semantic features. The qualifying score of an image with respect to a list of semantic features is the product of the qualifying scores of that image with respect to the features in that list. If the user click an icon, a window pops up with the original and annotated images (Figure 8.3.3). Users can also try with images from the internet or their local machines. After an image is uploaded, it's analyzed in the server side, and its semantic features are reported to the user. Then the user can submit the list of these semantic features as an image query.

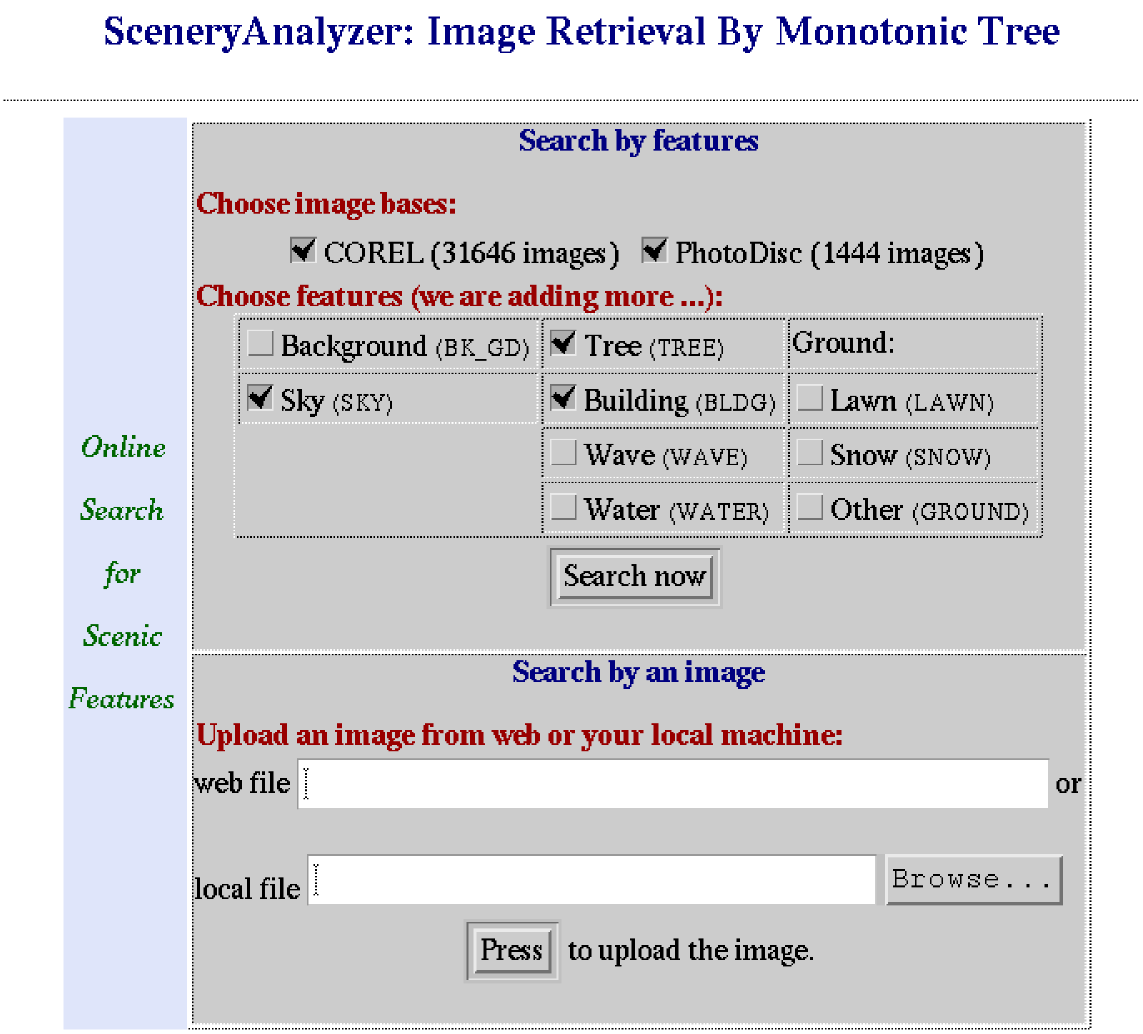


Figure 8.3.1: Interface of SceneryAnalyzer.

Figure 8.3.2: Retrieval results.

Figure 8.3.3: The original and annotated images are shown in a popout window.

Chapter 9

Extraction of Semantic Features

The feature extraction scheme used here is based on reduced monotonic trees. For color images, the reduced monotonic trees are defined on their gray copies. We use the branches of the reduced monotonic tree to model the basic structures in an image, which are termed *structural elements*. Structural elements have low-level features such as color, shape, and spatial location. They are clustered to form high-level features.

Feature extraction consists of three consecutive steps:

(a) classifying structural elements;

(b) clustering structural elements; and

(c) rendering semantic regions.

Figure 9.0.4 shows the relationships among the steps given above.

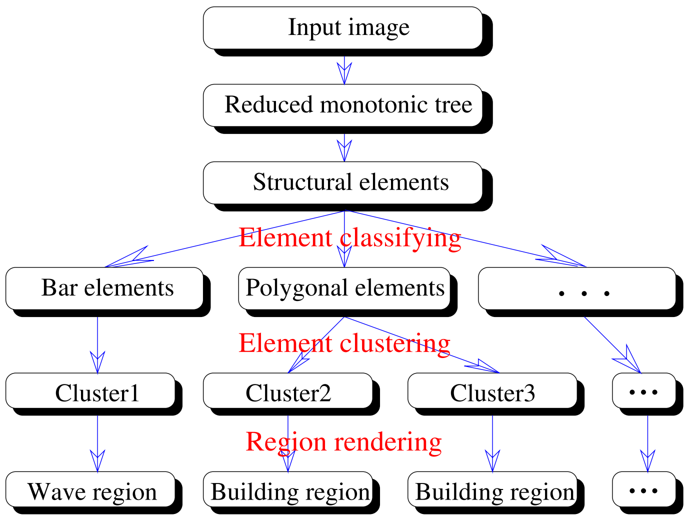


Figure 9.0.4: System design for feature extraction.

The features of a given image are extracted and stored in a feature vector. A feature vector is an array which records the qualifying scores for all semantic features. The qualifying score of an image with respect to a feature indicates the degree that the image can be considered to have this feature. Figure 9.0.5 illustrates the data structure of a feature vector.

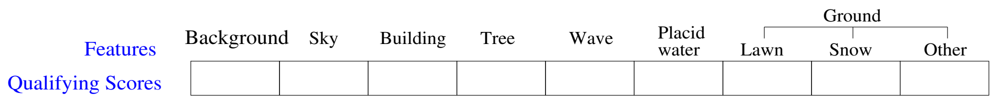


Figure 9.0.5: Data structure of a feature vector.

9.1 Classifying Structural Elements

Each branch (or subtree) of a reduced monotonic tree is called a *structural element* if the image area it covers doesn't exceed an area threshold. A structural element is considered *positive/negative* if its root (i.e., the root of the subtree) is outward-falling/climbing, as shown in Figure 9.1.1(a). Positive/negative elements are like peaks/valleys. The *altitude* of a positive/negative element is defined as the absolute value of the average altitude of all its pixels above/blow the highest/lowest pixels adjacent to the structural element, as shown in Figure 9.1.1(b). The *coarseness* of a structural element is determined by the number, area, and altitude of its sub-elements.[1] We define the coarseness of an element t by

$$Coarseness(t) = \frac{\Sigma_{b \in SubElementSet(t)} Altitude(b) * Area(b)}{Area(t)},$$

where $SubElementSet(t)$ is the set of sub-elements of t, $Altitude(b)$ is the altitude of b, and $Area(b)$ is the area of the region covered by b.

A structural element can be classified by its:

(1) color (the average color of all pixels in this element);

(2) altitude;

(3) coarseness; and

(4) shape (the shape of its covered region).

[1] An element is a branch of the reduced monotonic tree. All sub-branches of this branch are sub-elements.

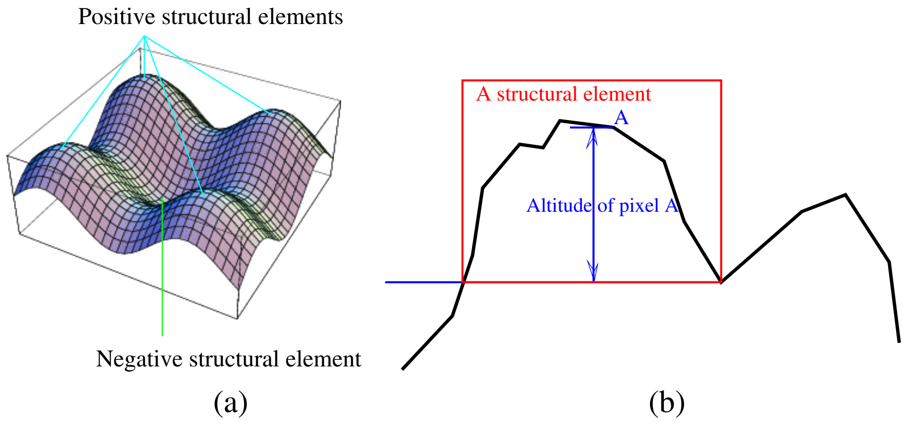


Figure 9.1.1: (a) Positive/negative structural elements; (b) the altitude of a pixel in a structural element.

For example, we can classify the structural elements by shape as:

(a) bars;

(b) polygons;

(c) irregular elements;

(d) smooth-boundary elements; and

(e) other shapes.

A bar element is characterized by its high length-to-width ratio, while a polygonal element is a structural element whose boundary consists mainly of line segments. A smooth-boundary element has a boundary which forms a smooth curve. Irregular elements are those with irregular boundaries. Figure 9.1.2 shows structural elements of different shapes.

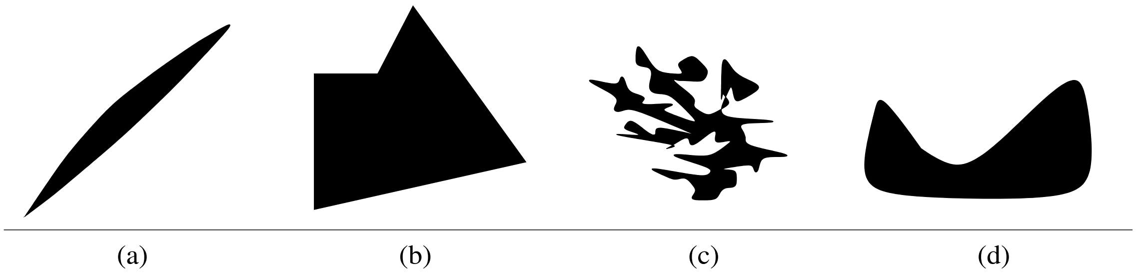


Figure 9.1.2: (a) A bar element, (b) a polygonal element, (c) an irregular element, and (d) a smooth-boundary element.

The semantic features of scenery images are characterized by categories of structural elements which are clustered in the images. Three examples of the categories are polygonal elements (for buildings), horizontal bar elements (for waves), and green coarse irregular elements (for trees), as shown in Figures 9.1.3 and 9.1.4.

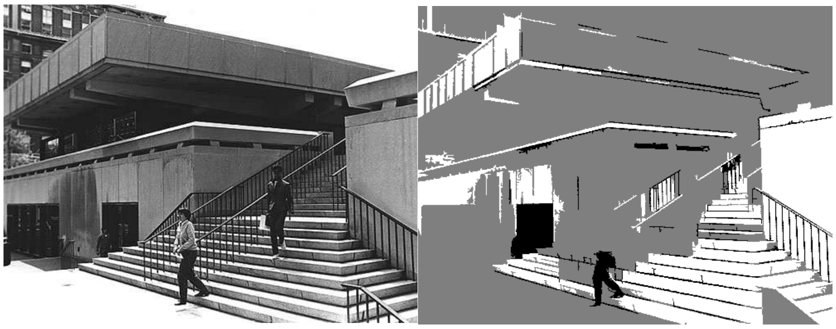


Figure 9.1.3: An image and its polygonal elements (shown in black and white).

9.2 Clustering Structural Elements

Once the structural elements of an image have been classified, algorithms are applied to identify clusters of elements. First, the set of qualified (i.e., "belong-to") elements must

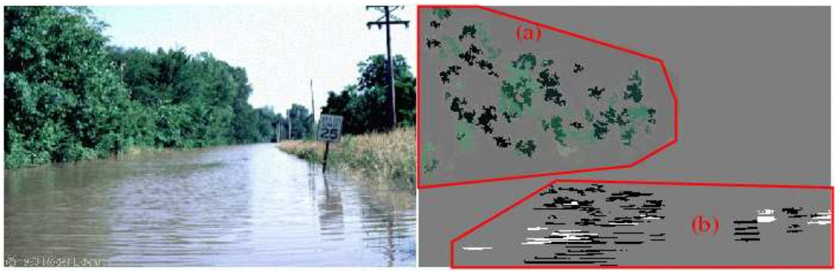


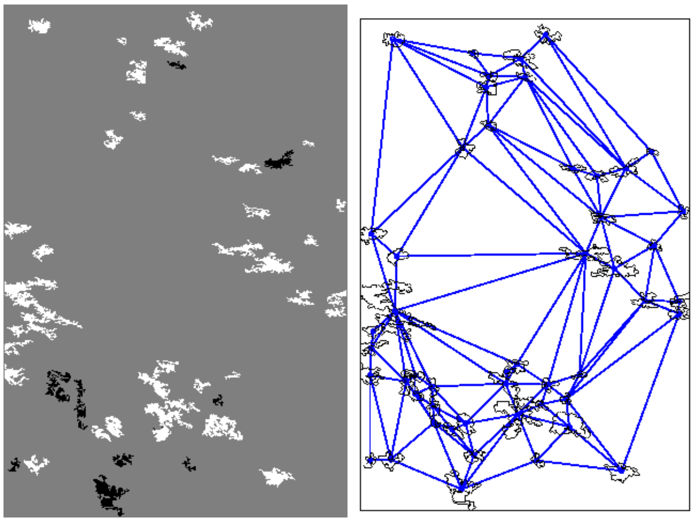
Figure 9.1.4: An image and its green coarse irregular elements (shown in green or dark green) as well as its horizontal bar elements (shown in black and white).

be determined for each category. If two qualified elements overlap, the one with the lower qualifying score[2] is eliminated from the category. This process is called *element sifting*. The sifting process reduces the multi-level elements of the image to single-level elements, all belonging to the given category. The elements after sifting form an element pattern in the 2D plane.

Next, as shown in Figure 9.2.1, the Delaunay graph of the element pattern is constructed; this is the neighboring graph of the element pattern. We then apply clustering algorithms to the neighboring graph to find the clusters in the element pattern. Here, the clustering algorithm is based on the minimal spanning tree of the neighboring graph. Briefly described, the process of clustering by minimal spanning tree is as follows. Let V be the vertex set of the graph and D be the distance threshold. Then V is grouped into disjoint sets by joining all edges in the minimal spanning tree whose lengths (or weights) are less than or equal to D. Each set thus obtained is said to be a cluster at level D. Further background on

[2]For a given category, the qualifying score of an element indicates the degree that the element can be considered "qualified" to belong to this category. The process of arriving at a qualifying score will be discussed later.

pattern processing by neighboring graphs can be found in [1, 2]. References on clustering by minimal spanning tree can be found in [105].



(a) (b)

Figure 9.2.1: (a) An element pattern, and (b) the neighboring graph of the pattern.

9.3 Rendering Semantic Regions

The process of rendering semantic regions from a cluster of structural elements consists of three steps: (1) element-connecting; (2) hole-filling; and (3) boundary-smoothing. In the first step, all elements in the cluster are connected by line segments of lengths within a threshold. The second step involves filling all holes which are smaller than a given area threshold. Finally, the boundary of the region is smoothed by removing irregular angles and branches. These steps are illustrated in Figure 9.3.1.

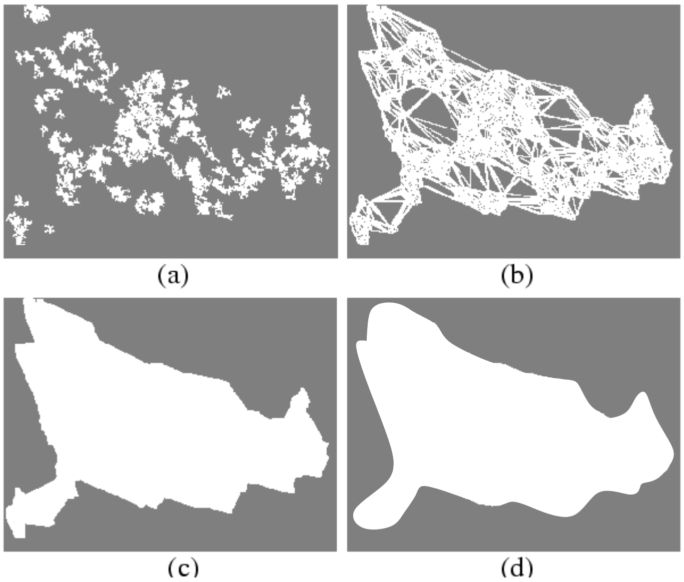


Figure 9.3.1: (a) A cluster of structural elements, (b) element connecting, (c) hole filling, and (d) boundary smoothing.

9.4 Qualifying Scores

Qualifying scores must be established for structural elements, regions, and images. The qualifying score of a structural element measures the degree to which the element is considered to belong to a given category of elements. Qualifying scores for different element categories are determined in different ways. Formulas for computing the qualifying scores for various scenery-feature element categories will be presented in Section 9.5.

Similarly, the qualifying score of a region with a semantic feature measures the degree that the region manifests this particular feature. For a region generated from a cluster, its qualifying score is determined by the qualifying scores of the elements in the cluster and the area of the region. For example, a wave region is rendered from a cluster of horizontal bar elements. Let the input image be defined on the domain Ω and R be a wave region generated from a cluster S of horizontal bar elements. The qualifying score for R with respect to wave is defined as

$$Score_{wave}(R) = \frac{\Sigma_{t \in S} Score_{hbar}(t)}{\alpha_{wave}} * \frac{Area(R)}{Area(\Omega)},$$

where $Score_{hbar}(t)$ is the qualifying score for t to be a horizontal bar element, and α_{wave} is a parameter to scale the qualifying score. $Score_{hbar}(t)$ is defined in the next section.

In contrast, sky and ground regions are not generated from clusters. As will be discussed in the next section, we will assume that these regions are smooth. Coarse regions are identified first; the smooth regions are then the complement of the coarse regions. The qualifying score of a smooth region is determined by the smoothness, color, and area of the region. As with the coarseness of a structural element, the smoothness of a region is determined by the area and altitude of the structural elements intersecting this region:

$$Smoothness(R) = \frac{Area(R)}{\Sigma_{t \in ElementSet} Altitude(t) * Area(Region(t) \bigcap R)},$$

where $ElementSet$ is the set of all structural elements of the input image and $Region(t)$ is the region covered by t. The qualifying score for a region R to be considered a sky region is defined as [3]

$$Score_{sky}(R) = \frac{Smoothness(R)}{\alpha_{sky}} * \frac{Area(R)}{Area(\Omega)} * SkyColorRatio(R),$$

[3] Region R should first be examined to ascertain that it satisfies our assumptions about sky regions.

where α_{sky} is a parameter to scale the qualifying score and $SkyColorRatio(R)$ is the ratio of sky-colored pixels in R. [4] The qualifying scores for ground and placid-water regions are defined similarly.

An input image may have several regions which manifest a specific feature. The qualifying score for the input image with respect to this feature is defined as the sum of qualifying scores of these regions. For example, let I be the input image and $R_1, R_2, ..., R_n$ be the tree regions found in this image. The qualifying score of I with respect to "tree" is

$$Score_{tree}(I) = \Sigma_{i=1}^{n} Score_{tree}(R_i).$$

If an input image contains no regions which manifest a specific feature, then the qualifying score for the image with respect to this feature is zero.

9.5 Case Study of Semantic Features

The previous sections described a general scheme for the extraction of semantic features. In this section, we will present methods for defining the categories of structural elements and identifying the regions with scenery features.

The following discussion is predicated upon some simple assumptions about the colors of

[4] We assume that the sky color is either blue or gray (the color of clouds). These assumptions will be discussed in the next section.

scenery features. For example, trees are assumed to be green and water to be blue. More comprehensive color patterns for scenery features can be integrated into our system, as is not the focus of this dissertation.

We will first briefly introduce our approach to color modeling. A pixel is a *gray* pixel if none of its red, green, and blue components is dominant; i.e.,

$$\max(r, g, b) - \min(r, g, b) \leq T^g, \tag{9.5.1}$$

where r, g, and b are the red, green, and blue components of the pixel, respectively, and T^g is a threshold. A pixel is *red*, *green*, or *blue* if

$$r > \max(g, b) + T^g,\ g > \max(r, b) + T^g,\ \textit{or}\ b > \max(r, g) + T^g, \tag{9.5.2}$$

respectively. A pixel is *white* if it is gray and

$$(3 * r + 10 * g + b)/10 \geq T^w, \tag{9.5.3}$$

where $(3 * r + 10 * g + b)/10$ is the brightness of the pixel and T^w is a threshold. A region is *red (green, blue, gray, or white)* if the ratio of red (green, blue, gray, or white) pixels in the region is greater than or equal to a threshold.

We also make assumptions about the location of sky and ground regions in images. For a region R in an image I, its location is modeled by its top, bottom, left, and right positions:[5]

$$TopPos(R) \quad = \quad \min\{y | \exists x, (x, y) \in R\}, \tag{9.5.4}$$

[5]Here, we assume the monitor coordinate system: the x-axis is pointing right and the y-axis is pointing down.

$$BottomPos(R) = \max\{y|\exists x, (x,y) \in R\}, \tag{9.5.5}$$

$$LeftPos(R) = \min\{x|\exists y, (x,y) \in R\}, \tag{9.5.6}$$

$$RightPos(R) = \max\{x|\exists y, (x,y) \in R\}. \tag{9.5.7}$$

Suppose the domain of image I is $\{0, 1, ..., N-1\} \times \{0, 1, ..., M-1\}$. We say that a region R is in the lower part of the image if $\frac{TopPos(R)}{M}$ is greater than a threshold; we say that a region occupies an upper part of the image if $\frac{TopPos(R)}{M}$ is less than a threshold.

9.5.1 Smooth Background

In natural images, background scenes have complicated relationships with foreground objects. To simplify the situation, we make **two assumptions**: as compared with foreground shapes, the background regions are

- (background.a1) smoother in intensity; and
- (background.a2) peripheral in location.

The assumption (background.a1) is true in most cases, because usually

- the background consists of homogeneous surfaces; and/or

- the background scene is far from the camera, thus the detail in the background is not projected clearly to the picture; and/or
- the camera is focused on the foreground objects, thus the background gets blurred.

By the assumption (background.a1), the foreground is coarser in intensity, which is characterized by intensity peaks and valleys. Under monotonic tree, the intensity peaks and valleys are modeled as small structural elements whose altitudes are high. Thus we detect the coarse regions of the image by clustering the structural elements whose altitudes are high. The smooth regions are the complement of the coarse regions. Then we check the location of the smooth regions. Those peripheral smooth regions are annotated as background regions.

9.5.2 Sky

The sky is a kind of background. Without clouds, a sky region is a homogeneous region consisting of blue pixels. Due to their physical properties, clouds in an image tend to change smoothly at the pixel level. From a location's stand point, there is usually no other object above a sky region. To retrieve sky regions, we make three simple assumptions:

- (sky.a1) A sky region is smooth;

- (sky.a2) A sky region occupies an upper part of the image; and
- (sky.a3) The color of sky regions is either blue or the color of clouds.

For our current implementation, we assume that the color of clouds is gray. To find the sky regions, we first identify the smooth regions in the image. The smooth regions are the complement of the coarse regions, which are characterized by peaks and valleys in intensity. Within the monotonic tree, these peaks and valleys are modeled as small structural elements with high altitudes. Thus, the coarse regions of the image are detected by clustering these small, high-altitude elements, as discussed in the last sub-section. Identification of the coarse regions automatically distinguishes the smooth regions, as well. These smooth regions are then checked for location and color to find the sky regions.

9.5.3 Ground and Placid Water

We make three assumptions about ground regions:

- (ground.a1) A ground region is smooth;
- (ground.a2) A ground region is in the lower part of the image; and
- (ground.a3) In a ground region, structural elements are more horizontal than vertical.

When scenery images are taken, the direction of the projection is usually horizontal or nearly horizontal. Thus, as the natural scene is projected onto the image plane, ground-level structures appear more horizontal than vertical, leading to the third assumption above.

Let us first discuss methods for modeling the horizontality of a structural element. For a region X on the square grid $\mathbb{Z}^2$ and some $x_0 \in \mathbb{Z}$, we define the height of X at x_0 to be the number of pixels in the intersection between X and the vertical line crossing $(x_0, 0)$. Formally,

(9.5.8) $$Height(X, x_0) = |\{(x_0, y)|(x_0, y) \in X\}|,$$

where $|\bullet|$ gives the number of elements in a set. Similarly, we define the width of X at y_0 as

(9.5.9) $$Width(X, y_0) = |\{(x, y_0)|(x, y_0) \in X\}|.$$

Now we define the horizontal and vertical ratios of X to be

(9.5.10) $$HorizontalRatio(X) = \sqrt{\frac{\Sigma_{y\in\mathbb{Z}}(Width(X, y))^2}{\Sigma_{x\in\mathbb{Z}}(Height(X, x))^2}}; \quad and$$

(9.5.11) $$VerticalRatio(X) = \frac{1}{HorizontalRatio(X)}.$$

For a rectangle $\{0, 1, ..., N-1\} \times \{0, 1, ..., M-1\}$, its horizontal ratio is $\frac{N}{M}$. A structural element is said to be horizontal (or vertical), if the horizontal (or vertical) ratio of its covered region is greater than a given threshold.

As with the detection of sky regions, the identification of ground and placid-water regions starts with finding the smooth regions in the image. Then, for each smooth region, we check the validity of assumptions (ground.a2) and (ground.a3). To test (ground.a3), we count the horizontal and vertical elements in the smooth region. The last assumption holds if the horizontal elements exceed the vertical elements in the region.

A ground region identified by this method could be lawn, snow, or other ground subtypes. These region subtypes are distinguished by color. For example, we assume that lawn regions are green and snow regions are white.

Identification of placid-water regions is based upon four assumptions. The first three are the same as those used for ground identification. In addition, we assume that the color of placid water is blue.

9.5.4 Wave

Small water waves are characterized by very regular patterns which appear as horizontal bars in a horizontally-projected image, such as that shown in Figure 9.1.4. Breaking waves have complicated structures. However, images with breaking waves usually contain an area of water surface which consists of parallel bar structures, as in the image shown in Figure 9.5.1.

We define a wave region as a region consisting of horizontal bar elements. A bar is characterized by its high length-to-thickness ratio. For a 2D region X, we define its length and

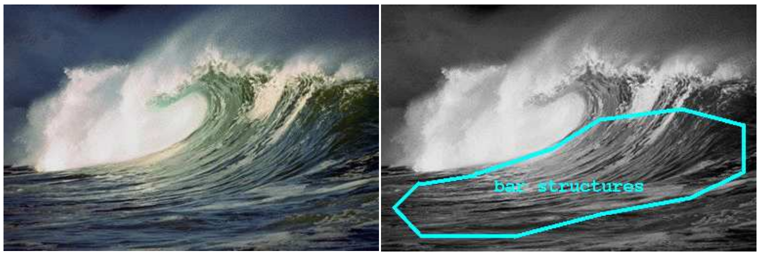


Figure 9.5.1: An example of an image with breaking waves.

thickness as

$$Length(X) = \max_{x,y \in X} \parallel X - Y \parallel, \tag{9.5.12}$$

$$Thickness(X) = \min_{v \in \mathbb{R}^2} \max_{x,y \in X} \frac{(x-y) \bullet v}{\parallel v \parallel}, \tag{9.5.13}$$

respectively, where $(x - y) \bullet v$ is the inner product of vectors $x - y$ and v, and $\parallel v \parallel$ is the length of the vector v. We define the bar ratio of a region X as

$$BarRatio(X) = \frac{Length(X)}{Thickness(X)}. \tag{9.5.14}$$

This situation is illustrated in Figure 9.5.2. A structural element is considered to be a bar element if the bar ratio of its covered region is higher than a given threshold. The qualifying score of a structural element to be a horizontal bar element is defined as the product of the bar ratio and horizontal ratio of its covered region:

$$Score_{hbar}(t) = BarRatio(Region(t)) * HorizontalRatio(Region(t)). \tag{9.5.15}$$

Wave regions are identified by clustering horizontal bar elements in the image.

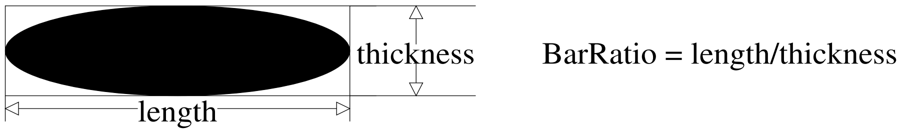


Figure 9.5.2: The bar ratio of an ellipse.

9.5.5 Green Tree

A careful examination of the tree region in Figure 9.1.4 reveals that the micro-structures in the region are very irregular. Based on this observation, we assume that a tree region is a region consisting of green coarse irregular elements. The tree regions in an image are found by clustering the green coarse irregular elements in the image.

For a connected region X, let $ConvexHullLength(X)$ be the length of its convex hull and $Length(\partial X)$ be the length of its boundary. We define the irregularity of X by

$$Irregularity(X) = \frac{Length(\partial X)}{ConvexHullLength(X)}. \tag{9.5.16}$$

The qualifying score of a structural element to be a tree element is defined as the product of its coarseness and the irregularity of its covered region:

$$Score_{treeElement}(t) = Coarseness(t) * Irregularity(Region(t)). \tag{9.5.17}$$

9.5.6 Building

The shapes of most buildings are characterized by the line segments they contain. We assume that a building region in an image is a region consisting of polygonal elements. To ascertain whether a structural element is polygonal, we first partition its boundary into line segments and other kinds of segments. A structural element is polygonal if its boundary mainly consists of line segments. References on curve partitioning can be found in [57, 58, 59].

For a connected region X, let $Length(\partial X)$ be the length of its boundary and

$$LengthOfLineSegments(X)$$

be the total length of the line segments detected in its boundary. We define the polygon ratio of X by

$$PolygonRatio(X) = \frac{LengthOfLineSegments(X)}{Length(\partial X)}. \tag{9.5.18}$$

The qualifying score of a structural element to be a polygonal element is defined as the polygon ratio of its covered region.

Chapter 10

Experiments and Analysis

Comprehensive experiments were conducted to demonstrate the effectiveness of the monotonic tree model on image retrieval; these experiments employed our SceneryAnalyzer system. In the experiments, the SceneryAnalyzer system was compared with the keyblock model [31], traditional color histogram [90], color coherent vector [51], and wavelet (Haar and Daubechies) texture techniques [76, 86].

First, performance comparisons using COREL images were made between the proposed approach and the traditional CBIR techniques. These comparisons were based on the precision-recall for individual scenery features. We then conducted experiments using PhotoDisc images to verify that the performance of our approach is not source-sensitive. Next, the proposed approach was applied to both COREL and PhotoDisc images to explore its effectiveness with different combinations of scenery features. Finally, We made performance comparisons on image background. Examples of some of these images and their copies annotated by SceneryAnalyzer are provided at the end of this chapter.

10.1 Experiment Setup

For each scenery feature, the precision-recall of SceneryAnalyzer was calculated and plotted. Let *RETRIEVELIST* be the list of n images retrieved with a given feature, which is sorted descendingly by qualifying scores. We calculate the precision and recall for each of the first $\frac{n}{30}$, $\frac{2n}{30}$, $\frac{3n}{30}$, ... , and n images in *RETRIEVELIST*. The precision and recall figures were then plotted to demonstrate the performance of SceneryAnalyzer.

In the experiments, SceneryAnalyzer was compared with the keyblock model [31], traditional color histogram [90], color coherent vector [51], and wavelet (Haar and Daubechies) texture techniques [76, 86]. These traditional methods accept only queries by examples. The process of selecting query sets and calculating the precision-recall for these methods can be illustrated with a typical case, where the sky feature is queried using the keyblock approach. Let's suppose that an image database has N images where S images have sky regions. Each sky image is used as a keyblock query on the database; the top 100 images are thus determined, and the number of sky images in this retrieved set is counted. The S sky images are then sorted in descending order by the number of sky images in their corresponding retrieved sets. The sorted list is denoted as *SKYLIST*. The first 5% images from *SKYLIST* are then selected as the query set and denoted as *QUERYSET*. For each image I in the database, we calculate its distance to *QUERYSET*-$\{I\}$ using the keyblock approach. [1] [2] The images in the database are then sorted in ascending order by this distance. The top S (i.e., the number of images with sky) images are retrieved. Using the images retrieved by the keyblock model, we then calculate and plot the precision-recall of the keyblock model for the sky feature.

10.2 Comparison On Individual Scenery Features

Experiments were conducted using 6,776 COREL images to compare SceneryAnalyzer with the keyblock approach [31], traditional color histogram [90], color coherent vector [51], and wavelet (Haar and Daubechies) texture techniques [76, 86]. The precision-recall of these techniques was compared for six scenery features: sky, building, tree, wave, ground, and placid water.

[1] The distance from an image to a set of images is the shortest distance from this image to the images in the set.

[2] If we retrieve by the distances of images to *QUERYSET*, the images in *QUERYSET* will have distance zero. Thus the query images will automatically be retrieved as the top 5% images, which provides an unfair comparison.

Feature	sky	building	tree	wave	placid water	ground		
						lawn	snow	other
images with the feature	2369	1920	1479	161	882	298	68	659

Table 10.2.1: Breakdown by scenery features of images in COREL database.

The COREL images were from CD7 and CD8 of COREL Gallery 1,300,000. These photos are stored in JPEG format and are sized at either 256×384 or 384×256. The COREL database includes both scenery and non-scenery images. There are 4,125 scenery pictures taken at globally diverse locations, along with 2,651 non-scenery images covering a wide variety of subjects, including fish, molecules, space scenes, insects, and other topics. Table 10.2.1 shows a breakdown of the image database by scenery features.

The precision-recall of each method was calculated for each scenery feature listed; these results appear in Figure 10.2.1. The five traditional CBIR methods all performed poorly with the wave feature. This can be partially explained by the small number of wave images in the COREL database, which makes these methods easily misidentify other images as wave images. Furthermore, these methods didn't capture the structures of waves.

A comparison of the graphs in Figure 10.2.1 shows that our method outperforms all the others for each scenery feature type. To further clarify the comparison, we calculated the average precision-recall over all six scenery features, which is shown in Figure 10.2.2(a). The average precision-recall of the five traditional methods suffers particularly from their poor performance with the wave feature, as discussed above. To exclude this effect, we

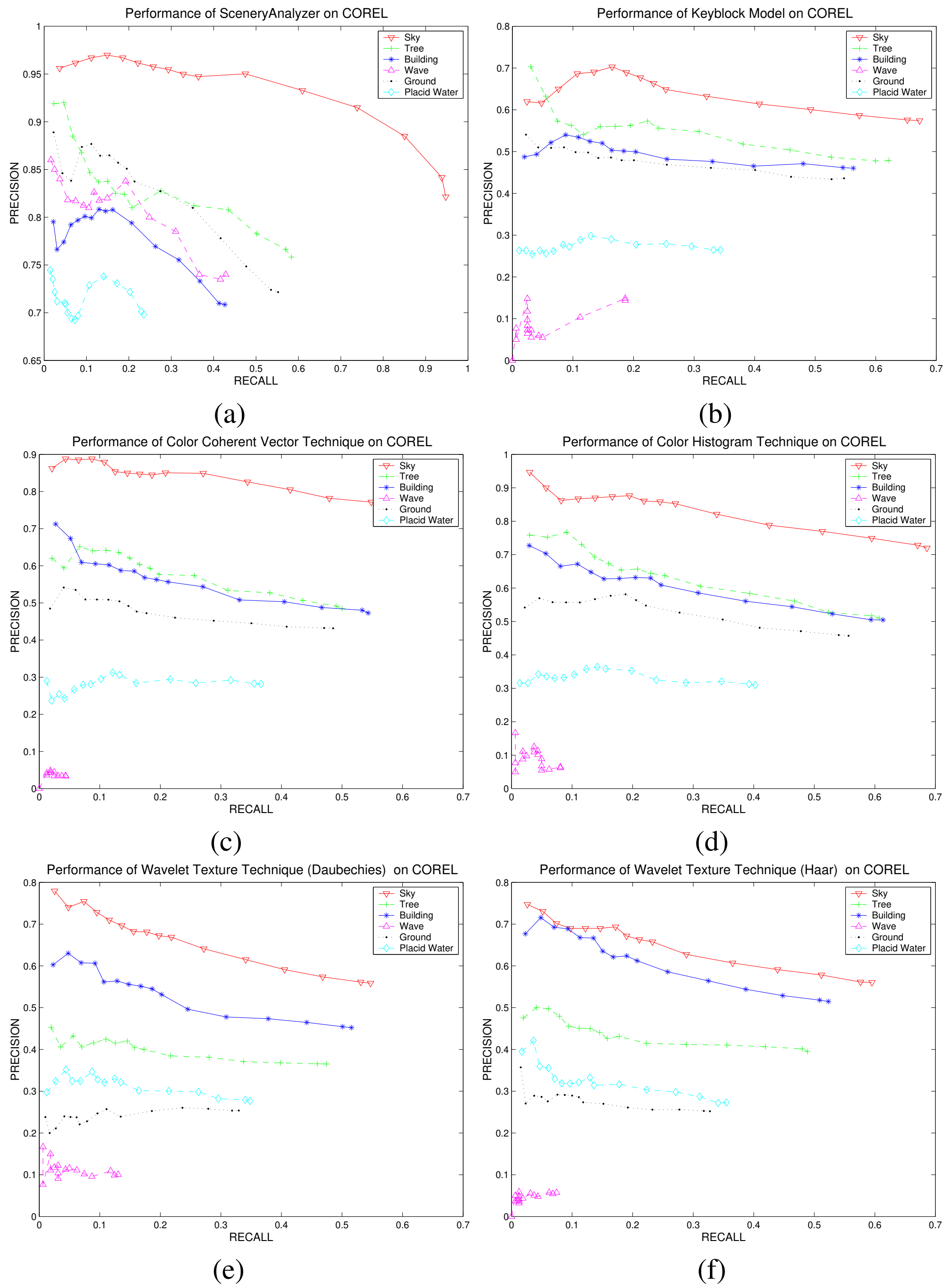
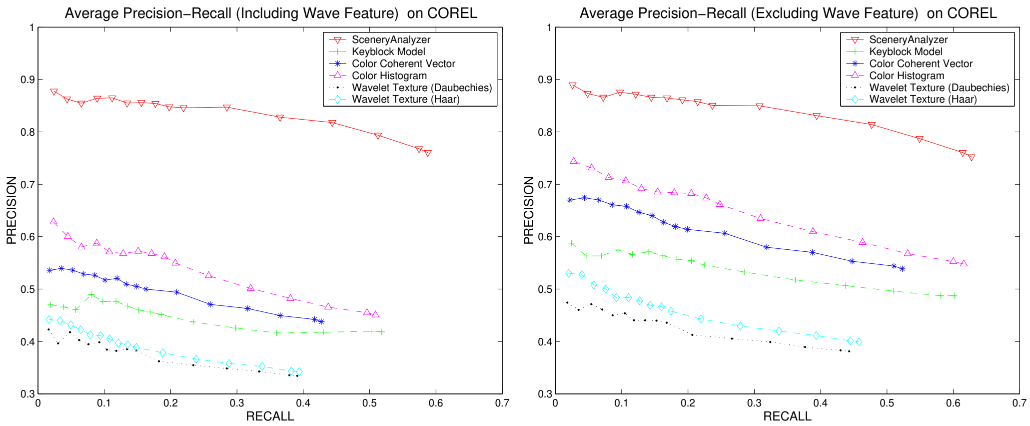


Figure 10.2.1: Performance of (a) SceneryAnalyzer, (b) keyblock model, (c) color coherent vector, (d) color histogram, (e) Daubechies wavelet, and (f) Haar wavelet with COREL images.



(a) (b)

Figure 10.2.2: (a) Average precision-recall for six features: sky, building, tree, wave, ground, and placid water; (b) average precision-recall for five features: sky, building, tree, ground, and placid water.

also calculated the average precision-recall over the five remaining scenery feature types (sky, building, tree, ground, and placid water); this is shown in Figure 10.2.2(b). These comparisons indicate that our proposed method provides much better precision-recall of scenery features than the traditional techniques.

10.3 Experiments Using PhotoDisc Database

We also conducted experiments on 1,444 PhotoDisc images which were from PhotoDisc Comping Discs 3 and 4. These images fall into five categories: (1) homes and gardens, (2) international sports, (3) nature scenes, (4) panoramic landscapes, and (5) people, lifestyles, and vacations. Table 10.3.1 shows a breakdown of the image database by scenery features.

Feature	sky	building	tree	wave	placid water	ground		
						lawn	snow	other
images with the feature	455	156	382	6	219	57	59	261

Table 10.3.1: Breakdown by scenery features of images in PhotoDisc database.

The performance of SceneryAnalyzer with this PhotoDisc database is shown in Figure 10.3.1(a) and (b). Since the PhotoDisc database included only 6 images with waves, a precision-recall curve could not be plotted for the wave feature. Thus, Figure 10.3.1(a) shows only five precision-recall curves. It's worth noting, however, that when queried for wave images on this database, SceneryAnalyzer retrieved three images, all with waves. Therefore, the precision of these queries was 100%, with a 50% recall. As a comparison, the performance of SceneryAnalyzer with the COREL database (discussed in last section) is shown in Figure 10.3.1(c) and (d). These experiments show that SceneryAnalyzer works effectively with both COREL and PhotoDisc images.

With both the COREL and PhotoDisc databases, the performance of SceneryAnalyzer is relatively poor with respect to snow subtype of ground. In our implementation, we have assumed that the ground is smooth and the snow ground is white. However, the snow ground in natural images may not be smooth and may vary in color. In addition, many other objects in natural images appear to be white and smooth. These factors negatively impact the performance with respect to snow ground.

features	sky tree	sky wave	tree building	tree ground	sky building lawn	sky tree ground	sky tree building lawn	tree water	sky snow
$\lvert\mathbf{F}\rvert$	987	92	571	458	100	262	43	363	78
$\lvert\mathbf{R}\rvert$	20	20	20	20	20	20	5	20	20
$\lvert\mathbf{F}\bigcap\mathbf{R}\rvert$	17	17	18	18	18	18	5	11	8
precision	85%	85%	90%	90%	90%	90%	100%	55%	40%
recall	1.7%	18%	3.1%	3.9%	18%	6.9%	12%	3%	10%

Table 10.4.1: Performance of SceneryAnalyzer on combinations of features.

10.4 Experiments On Combinations of Scenery Features

Experiments were conducted to demonstrate the effectiveness of SceneryAnalyzer with combinations of scenery features. The experiments involved all the images in the COREL and PhotoDisc databases. The top 20 images for each combination of scenery features were retrieved and the precision and recall calculated; results are given in Table 10.4.1. In this table, for each combination of scenery features, **F** denotes the set of images with the stipulated features and **R** denotes the set of images retrieved. For a set X, $|X|$ denotes the number of elements in X. As a comparison, the performance of the five traditional CBIR techniques is given in Table 10.4.2, where the keyblock model is abbreviated as kb, the color histogram technique as ch, the color coherent vector as ccv, the Daubechies texture technique as daub, and the Haar texture technique as haar.

Table 10.4.1 demonstrates that the SceneryAnalyzer system is effective for the feature combinations listed. In contrast, Table 10.4.2 shows that traditional CBIR techniques are not

features	sky tree	sky wave	tree building	tree ground	sky building lawn	sky tree ground	sky tree building lawn	tree water	sky snow
$\|\mathbf{F}\|$	987	92	571	458	100	262	43	363	78
$\|\mathbf{R}_{kb}\|$	20	20	20	20	20	20	20	20	20
$\|\mathbf{F} \bigcap \mathbf{R}_{kb}\|$	7	5	8	3	5	3	0	3	1
precision	35%	25%	40%	15%	25%	15%	0%	15%	5%
recall	0.7%	5.4%	1.4%	0.7%	5%	1.1%	0%	0.8%	1.3%
$\|\mathbf{R}_{ch}\|$	20	20	20	20	20	20	20	20	20
$\|\mathbf{F} \bigcap \mathbf{R}_{ch}\|$	10	2	4	1	3	2	1	5	5
precision	50%	10%	20%	5%	15%	10%	5%	25%	25%
recall	1%	2.2%	0.7%	0.2%	3%	0.8%	2.3%	1.4%	6.4%
$\|\mathbf{R}_{ccv}\|$	20	20	20	20	20	20	20	20	20
$\|\mathbf{F} \bigcap \mathbf{R}_{ccv}\|$	10	1	8	5	0	2	0	7	0
precision	50%	5%	40%	25%	0%	10%	0%	35%	0%
recall	1%	1.1%	1.4%	1.1%	0%	0.8%	0%	1.9%	0%
$\|\mathbf{R}_{daub}\|$	20	20	20	20	20	20	20	20	20
$\|\mathbf{F} \bigcap \mathbf{R}_{daub}\|$	7	1	1	2	3	1	1	3	1
precision	35%	5%	5%	10%	15%	5%	5%	15%	5%
recall	0.7%	1.1%	0.2%	0.4%	3%	0.4%	2.3%	0.8%	1.3%
$\|\mathbf{R}_{haar}\|$	20	20	20	20	20	20	20	20	20
$\|\mathbf{F} \bigcap \mathbf{R}_{haar}\|$	7	3	5	2	1	2	1	7	1
precision	35%	15%	25%	10%	5%	10%	5%	35%	5%
recall	0.7%	3.3%	0.9%	0.4%	1%	0.8%	2.3%	1.9%	1.3%

Table 10.4.2: Performance of the five traditional CBIR techniques on combinations of features.

satisfactory when handling combinations of semantic features. SceneryAnalyzer thus provides a new, semantics-based approach to handle queries for images.

10.5 Experiments On Image Background

We experimented on an image database consisting of 5553 COREL images and 1202 PhotoDisc images. The image database covers a large variety, including marine fishes, birds, insects, flowers, animals, nature scenes, sports, etc. In this database, 1843 images have obvious background regions.

We experimented with our approach on the image base. In our experiments, 2218 images are retrieved. Among them, 1763 images really have background regions. The precision is 79.5%, and the recall is 95.7%.

As a comparison, we also experimented with the five traditional CBIR methods. The result is listed in Table 10.5.1, where we use R to denote the set of images being retrieved, F to denote the set of images which really have background regions. In this table, our approach by monotonic tree is abbreviated as MT, the keyblock model as KB, the color histogram technique as CH, color coherent vector as CCV, Daubechies texture technique as Daub, and Haar texture technique as Haar. From the table we can see that our method outperforms all the others in retrieving images with obvious background.

methods	$\|F\|$	$\|R\|$	$\|R \bigcap F\|$	*precision*	*recall*
MT	1843	2218	1763	79.5%	95.7%
KB	1843	2218	836	37.7%	45.4%
CH	1843	2218	880	39.7%	47.7%
CCV	1843	2218	991	44.7%	53.8%
Daub	1843	2218	1164	52.5%	63.2%
Haar	1843	2218	1262	56.9%	68.5%

Table 10.5.1: Comparison on image background.

10.6 Summary

We have conducted comprehensive experiments to demonstrate the effectiveness of the monotonic tree model on image retrieval. We compared our approach with traditional color or texture based methods on image background and scenery features. Our approach outperformed all the others in each case. Our image retrieval system *SceneryAnalyzer* is very effective for the feature combinations. It provides a new, semantics-based approach to handle queries for images.

Figures 10.6.1 and 10.6.2 provide sample images and their copies as annotated by SceneryAnalyzer.

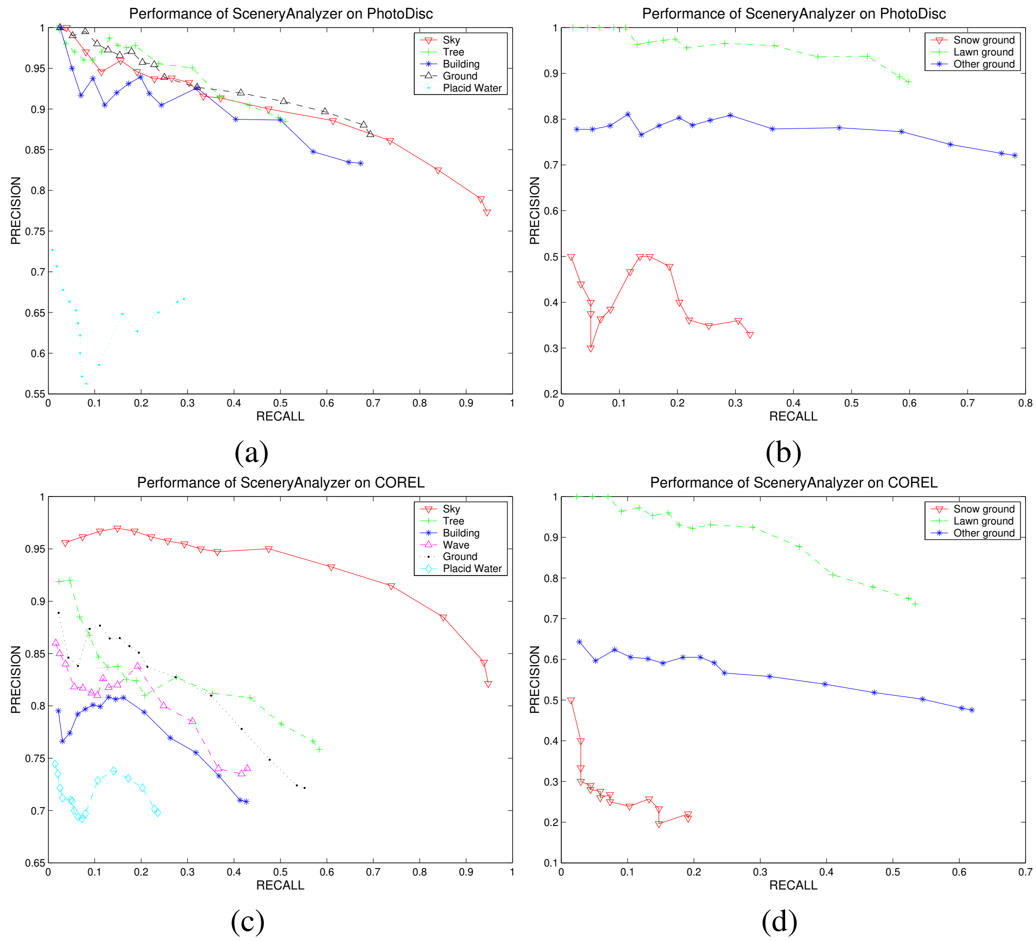


Figure 10.3.1: (a) Performance of SceneryAnalyzer using PhotoDisc database with respect to sky, building, tree, placid water, and ground; (b) performance of SceneryAnalyzer using PhotoDisc database with respect to snow, lawn, and other subtypes of ground ; (c) Performance of SceneryAnalyzer using COREL database with respect to sky, building, tree, wave, placid water, and ground; and (d) performance of SceneryAnalyzer using COREL database with respect to snow, lawn, and other subtypes of ground.

Figure 10.6.1: Examples of scenery images and their annotated copies.

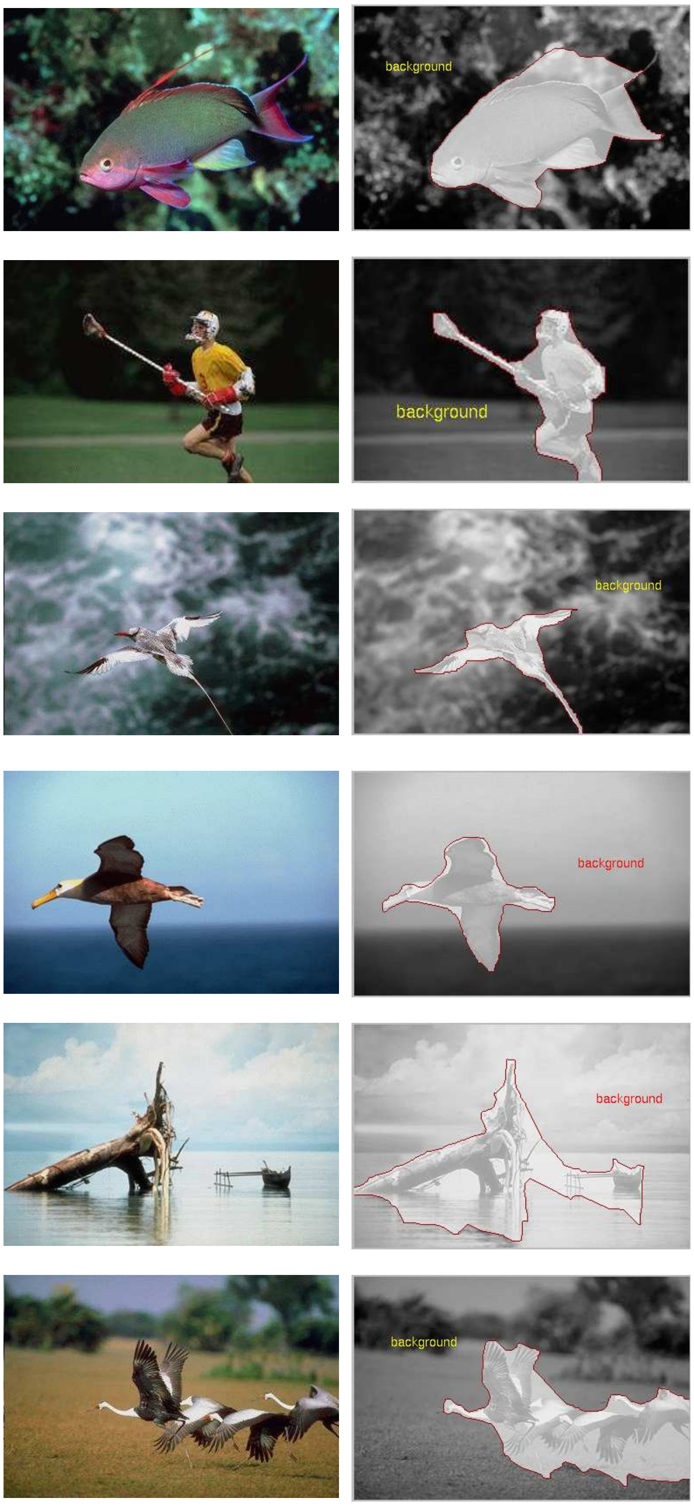


Figure 10.6.2: Original images and their background regions.

Chapter 11

Conclusion and Future Work

11.1 Contributions

This dissertation has two main contributions. The first contribution is the mathematical theory of monotonic tree, which is the first theory addressing definitions, properties and computations of contour structures directly on discrete functions. An optimal algorithm was given to compute monotonic trees on two-dimensional data. Because it's defined directly on discrete data and the interpolation step is avoided, the monotonic tree model is a better choice than contour tree models in GIS, computer cartography, and image coding.

The second contribution is the application of monotonic tree model to semantics-based image retrieval. The success of this model on scenery features and smooth background implies its potential in analyzing general semantics of both images and videos. As compared with other hierarchical or multi-scale models, such as wavelet [106, 54, 76], scale space [32], and peaks & ridges [13], our monotonic tree model has the following advantages.

(1) The monotonic tree retrieves and represents the structures of an image at all scales. In addition, these structures are organized hierarchically as a tree, which gives us a better way to analyze the relationship between different levels.

(2) The monotonic tree retrieves the structures of an image directly and maintains their original shapes.

(3) Image structures are classified by their color, coarseness and shape, then clustered and organized into clusters. Semantic features are drawn from the features of the clusters, including their spatial locations. The monotonic tree provides a frame to combine primitive features together to catch semantic features.

11.2 Future Work

Current work can be enhanced or extended along two directions: extending the monotonic tree model for all dimensions and applying it for extraction of general semantics from images and videos. Monotonic trees are only defined on the two-dimensional hexagonal grid.

The theory can be generalized to higher dimensions and to grids other than the hexagonal grid. The main difficulty is in the definition of monotonic lines/surfaces. The generalized definition must ensure that monotonic lines/surfaces don't cross each other.

Automatically retrieving semantic information from images/videos is extremely difficult. A quick review of computer vision research over the last few years points to this difficulty. I believe that the monotonic tree model will be a powerful tool to attack this problem. As a novel approach, the potential of the monotonic tree model is not fully exploited yet. In future work, the monotonic tree model can be extended for general semantic categories on both images and videos.

In extending the monotonic tree model, it should first be combined with existing theories on curves, regions, and point patterns. All these theories together provide basic mathematical properties of images, forming a mathematical base for higher-level processing. Then traditional computer vision techniques can be integrated, such as those in edge detection, texture analysis, shape from shading, shape matching, motion detection etc. For general images, domain knowledge is needed to detect objects in the images and derive the semantics of the images. Thus the monotonic tree model will also be unified with knowledge representation and machine learning techniques to extract semantic information from multimedia data efficiently and effectively. See Figure 11.2.1.

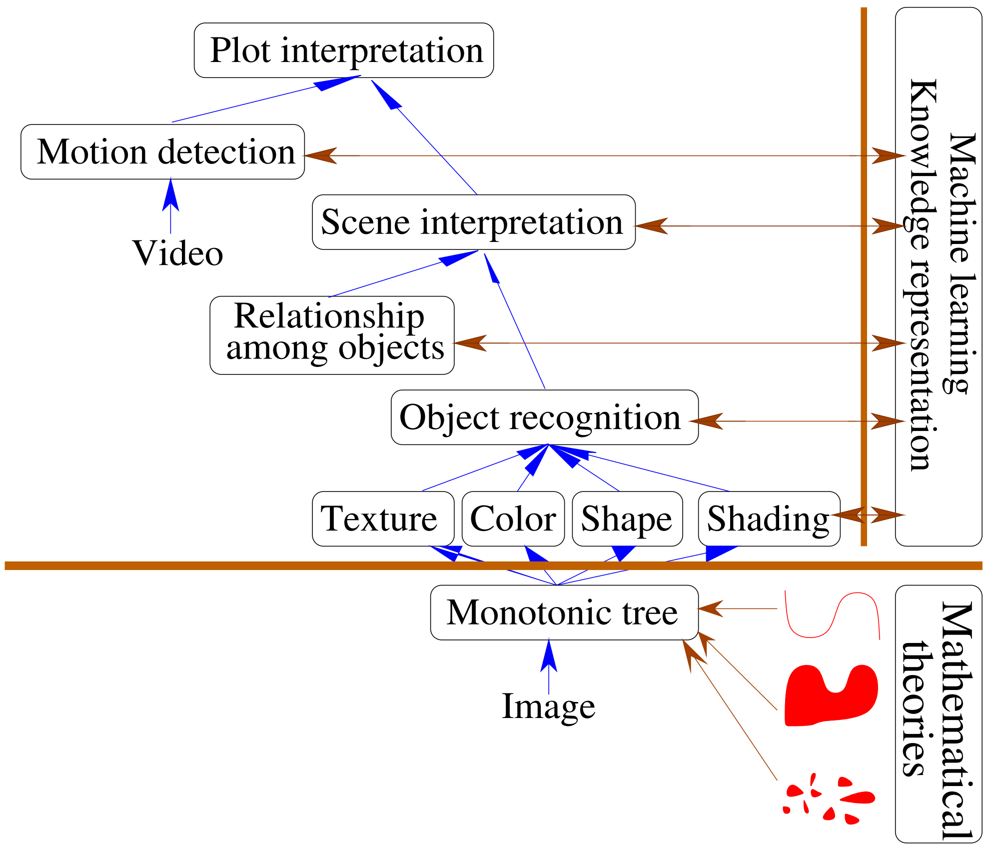


Figure 11.2.1: Extending the monotonic tree model for general semantics retrieval.

Appendix A

Publications

Book chapters:

1. Y. Song and A. Zhang, "SceneryAnalyzer: a System Supporting Semantics-based Image Retrieval", in *Intelligent Multimedia Documents*, book edited by Chabane Djeraba, Kluwer Academic Publishers, to appear. (invited chapter)

2. R. Aygun, Y. Song, and A. Zhang, "Multimedia Database Management", in *Multimedia Information Retrieval and Management*, book edited by D. Feng, W. C. Siu and Hong-Jiang Zhang, Springer, 2002. (invited chapter)

Refereed journal papers:

1. Y. Song and A. Zhang, "Analyzing Scenery Images by Monotonic Tree", in *ACM Multimedia Systems*. (to appear)

2. A. Zhang, Y. Song and M. Mielke, "*NetMedia*: Streaming Multimedia Presentations in Distributed Environments", in *IEEE Multimedia*, Vol. 9, No. 1, January-March, 2002, pp. 56–73.

3. M. Mielke, R. Aygun, Y. Song, and A. Zhang, "PLUS: A Probe-Loss Utilization Streaming Mechanism for Distributed Multimedia Presentation Systems", in *IEEE Transactions on Multimedia*. (to appear)

4. L. Zhang, C. Tang, Y. Song, A. Zhang, and M. Ramanathan, "VizCluster and Its Application on Clustering Gene Expression Data", in *International Journal of Distributed and Parallel Databases*. (to appear)

Refereed conference papers:

1. Y. Song and A. Zhang, "Locating Image Background By Monotonic Tree", in the proceedings of the *International Conference on Computer Vision, Pattern Recognition and Image Processing* (CVPRIP'2002), in conjunction with Sixth Joint Conference On Information Sciences (JCIS2002), Durham, North Carolina, March 8 - 14, 2002.

2. Y. Song and A. Zhang, "Monotonic Tree", in the proceedings of the *10th International Conference on Discrete Geometry for Computer Imagery (DGCI'2002)*, Bordeaux, France, April 3-5, 2002.

3. Y. Song, W. Wang, and A. Zhang, "Automatic Annotation and Retrieval of

Images", in the proceedings of the *IFIP conference on Visual Database Systems (VDB-6)*, special session on Multimedia Information Management and Retrieval, Brisbane, Australia, May 29-31, 2002. (invited paper).

4. W. Wang, Y. Song, and A. Zhang, "Semantics Retrieval by Content and Context of Image Regions", in the proceedings of the *15th International Conference on Vision Interface (VI'2002)*, Calgary, Canada, May 27-29, 2002.

5. W. Wang, Y. Song, and A. Zhang, "Semantics-based Image Retrieval by Region Saliency", in the proceedings of the *International Conference on Image and Video Retrieval (CIVR'2002)*, July 18-19, 2002, British Library, London, UK.

6. L. Zhang, C. Tang, Y. Shi, Y. Song, A. Zhang, and M. Ramanathan, "VizCluster: An Interactive Visualization Approach to Cluster Analysis and Its Application on Microarray Data", in the proceedings of the *Second SIAM International Conference on Data Mining* (SDM'2002), Arlington, April 11-13, 2002.

7. Y. Song, M. Mielke and A. Zhang, "NetMedia: Synchronized Streaming of Multimedia Presentations in Distributed Environments", in the Proceedings of *the Sixth IEEE International Conference on Multimedia Computing and Systems* (ICMCS'99), Florence, Italy, June 7-11, 1999, IEEE Computer Society Press, Vol II, pp. 585-590.

8. M. Mielke, Y. Song, R. Aygun and A. Zhang, "NetMedia: Giving Control to Distributed Multimedia Presentation Systems", in *the Eleventh International*

Scientific and Statistical Database Management conference (SSDBM-99), Cleveland, Ohio, August 1999, pp. 276-276. (demo)

Bibliography

[1] N. Ahuja. Dot pattern processing using voronoi neighborhoods. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 4(3):336–343, 1982.

[2] Narendra Ahuja and Mihran Tuceryan. Extraction of early perceptual structure in dot patterns: integrating region, boundary, and component gestalt. *Computer Vision, Graphics, and Image Processing*, 48(3):304–356, 1989.

[3] J.R. Bach, C. Fuller, A. Gupta, A. Hampapur, B. Horowitz, R. Jain, and C.F. Shu. The virage image search engine: An open framework for image management. In *Proceedings of SPIE, Storage and Retrieval for Still Image and Video Databases IV*, pages 76–87, San Jose, CA, USA, February 1996.

[4] T. F. Bancho. Critical points and curvature for embedded polyhedra. *Journal of Differential Geometry*, 1:245–256, 1967.

[5] R. L. Boyell and H. Ruston. Hybrid techniques for real-time radar simulation. In *IEEE Proceedings of the 1963 Fall Joint Computer Conference*, pages 445–458, November 1963.

[6] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Proceedings of Symposium on Discrete Algorithms*, pages 918–926, 2000.

[7] Hamish Carr. Efficient generation of contour trees in three dimensions. *Master thesis, Department of Computer Science, University of British Columbia*, 2000.

[8] S.-K. Chang and A. Hsu. Image information systems: Where do we go from here? *IEEE Trans. on Knowledge and Data Engineering*, 4(5), 1992.

[9] G. C.-H. Chuang and C-C. J. Kuo. Wavelet descriptor of planar curves: Theory and applications. *IEEE Transactions on Image Processing*, 5(1):56–70, January 1996.

[10] Paolo Cignoni, Claudio Montani, Enrico Puppo, and Roberto Scopigno. Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):352–369, 1997.

[11] H. E. Cline, C. L. Dumoulin, H. R. H. Jr., and W. E. Lorensen. 3d reconstruction of the brain from magnetic resonance images using a connectivity algorithm. *Magnetic Resonance Imaging*, 5:345–352, July 1987.

[12] Thomas H. Cormen, Charles E Leiserson, , and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[13] J. Crowley and A. Parker. A representation of shape based on peaks and ridges in the difference of low-pass transform. *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, 6(2):156–169, March 1984.

[14] Dov Dori and Hagit Hel-Or. Semantic content based image retrieval using object-process diagrams. In *Proceedings of the 7th International Workshop on Structural and Syntactic Pattern Recognition - SSPR98*, volume 1451, pages 230–241, 1998.

[15] John Eakins and Margaret Graham. Content-based image retrieval. In *Reports of JISC Technology Applications Programme*, Jan. 10, 1999. URL=http://www.jtap.ac.uk/reports/htm/jtap-039.html.

[16] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, July 1994.

[17] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, and et al. Query by Image and Video Content: The QBIC System. *IEEE Computer*, 28(9):23–32, September 1995.

[18] D.A. Forsyth, J. Malik, M.M. Fleck, H. Greenspan, T. Leung, S. Belongie, C. Carson, and C. Bregler. Finding pictures of objects in large collections of images. In

Report of the NSF/ARPA Workshop on 3D Object Representation for Computer Vision, page 335, 1996.

[19] H. Freeman and S. P. Morse. On searching a contour map for a given terrain elevation profile. *Journal of the Franklin Institute*, 248(1):1–25, 1967.

[20] W. I. Grosky and R. Zhao. Improved text-based web document retrieval using visual features. In *The First International Conference on Integration of Multimedia Contents*, 2001.

[21] Baining Guo. Interval set: A volume rendering technique generalizing isosurface extraction. In *IEEE Proceedings on Visualization*, pages 3–10, 1995.

[22] Gabor T. Herman. *Geometry of Digital Spaces*. Birkhauser, Boston, 1998.

[23] K. Hirata and T. Kato. Rough sketch-based image information retrieval. *NEC Research & Development*, 34(2):263–273, 1993.

[24] Berthold Klaus Paul Horn. *Robot Vision*. The MIT Press, forth edition, 1988.

[25] Takayuki Itoh and Koji Koyamada. Automatic isosurface propagation using an extrema graph and sorted boundary cell lists. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):319–327, 1995.

[26] Christoph Klauck. Graph grammar based object recognition for image retrieval. In *Second Asian Conference on Computer Vision*, 1995.

[27] R. Korfhage. *Information Storage and Retrieval*. New York: John Wiley & Sons, 1997.

[28] Flip Korn, Nikalaos Sidiropoulos, Christous Faloutsos, Eliot Siegel, and Zenon Protopapas. Fast nearest-neighbor search in medical image databases. In *Conference on Very Large Data Bases (VLDB96)*, September 1996.

[29] J. Kreyss, M. Roper, P. Alshuth, Th. Hermes, and O. Herzog. Video retrieval by still image analysis with imageminer. In *Proceedings of IS&T/SPIE's Symposium on Electronic Imaging: Science & Technologie*, 1997.

[30] I. S. Kweon and T. Kanade. Extracting topographic terrain features from elevation maps. *CVGIP: Image Understanding*, 59:171–182, 1994.

[31] L. Zhu, A. Rao and A. Zhang. Theory of keyblock-based image retrieval. *ACM Transactions on Information Systems*, Oct 2000.

[32] Tony Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, 1994.

[33] F. Liu and R. Picard. Periodicity, directionality, and randomness: Wold features for image modeling and retrieval. Technical Report 320, MIT Media Laboratory Perceptual Computing, 1996.

[34] X. Liu and J. R. Ramirez. Automated vectorization and labeling of very large hypsographic map images using a contour graph. *Surveying and Land Information Systems*, 57(1):5–10, 1997.

[35] Yarden Livnat, Han-Wei Shen, and Christopher R. Johnson. A near optimal isosurface extraction algorithm using the span space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[36] Sven Loncaric. A survey of shape analysis techniques. *Pattern Recognition*, 31(8):983–1001, 1998.

[37] William E. Lorenson and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[38] W. Y. Ma and B. S. Manjunath. NETRA: A toolbox for navigating large image databases. In *IEEE International Conference on Image Processing*, 1997.

[39] W.Y. Ma and B.S. Manjunath. Netra: A toolbox for navigating large image databases. In *International Conference on Image Processing*, pages I:568–571, 1997.

[40] B.B. Mandelbrot. *Fractals - Form, Chance, Dimension*. W.H. Freeman, San Francisco, California, 1977.

[41] B.S. Manjunath and W.Y. Ma. Texture Features for Browsing and Retrieval of Image Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, August 1996.

[42] D. M. Mark. Topological properties of geographic surfaces: Applications in computer cartography. In G. Dutton, editor, *First Int'l Adv. Study Symp. on Topological Data Structures for Geo. Info. Sys.*, volume 5. Harvard, 1978.

[43] N. Max. Optical models for direct volume rendering. *IEEE Transcations on Visualization and Computer Graphics*, 1, June 1995.

[44] Rajiv Mehrotra and James E. Gary. Similar-shape retrieval in shape data management. *IEEE Computer*, 28(9):57–62, September 1995.

[45] J. Milnor. *Morse Theory*. Princeton University Press, 1963.

[46] Farzin Mokhtarian, Sadegh Abbasi, and Josef Kittler. Efficient and Robust Retrieval by Shape Content through Curvature Scale Space. In *Proc. International Workshop on Image Databases and MultiMedia Search*, pages 35–42, Amsterdam, The Netherlands, 1996.

[47] Farzin Mokhtarian, Sadegh Abbasi, and Josef Kittler. Robust and efficient shape indexing through curvature scale space. In *Proceedings of British Machine Vision Conference*, pages 53–62, Edinburgh, UK, 1996.

[48] S. P. Morse. Concepts of use in computer map processing. *Communications of the ACM*, 12(3):147–152, March 1969.

[49] W. Niblack, R. Barker, W. Equitz, M. Flickner, E. Glasman, and D. Petkovic et al. The qbic project: Querying images by content using color, texture, and shape. Technical report, IBM Technical Report, 1993.

[50] G. Pass and R. Zabih. Histogram refinement for content-based image retrieval. *IEEE Workshop on Applications of Computer Vision*, pages 96–102, December 1996.

[51] Greg Pass, Ramin Zabih, and Justin Miller. Comparing images using color coherence vectors. In *Proceedings of ACM Multimedia 96*, pages 65–73, Boston MA USA, 1996.

[52] T. Pavlidis. *Structural Pattern Recognition*. Springer Verlag, Berlin, Heidelberg, New York, 1977.

[53] A. Pentland, R. Picard, and S. Sclaroff. Photobook: Tools for Content-based Manipulation of Image Databases. In *Proceedings of the SPIE Conference on Storage and Retrieval of Image and Video Databases II*, pages 34–47, 1994.

[54] K. Perez-Lopez, A. Sood, and M. Manohar. Selecting Image Subbands for Browsing Scientific Image Databases. In *Proceedings SPIE Conference on Wavelet Applications*, Orlando, April 1994.

[55] E. Persoon and K. S. Fu. Shape discrimination using fourier descriptors. *IEEE Transactions on System Management*, 1977.

[56] E. Rasmussen. Indexing images. *Annual Review of Information Science and Technology*, pages 169–196, 1997.

[57] C. Robl and G. Farber. Contour tracer for a fast and precise edge-line extraction. In *IAPR Workshop On Machine Vision Applications (MVA98)*, 1998.

[58] P.L. Rosin and G.A.W. West. Segmentation of edges into lines and arcs. *Image and Vision Computing*, 7(2):109–114, May 1989.

[59] P.L. Rosin and G.A.W. West. Multi-stage combined ellipse and line detection. In *British Machine Vision Conference (BMVC92)*, pages 197–206, 1992.

[60] J. Roubal and T.K. Poiker. Automated contour labelling and the contour tree. In *Proc. AUTO-CARTO 7*, pages 472–481, 1985.

[61] L. I. RUDIN, S. OSHER, and E. FATEMI. Nonlinear total variation based noise removal algorithms. *Journal of Physics D*, 60:259–268, 1992.

[62] Y. Rui, T. Huang, and S. Mehrotra. Content-based image retrieval with relevance feedback in mars, 1997.

[63] Y. Rui, T. S. Huang, and S.-F. Chang. Image retrieval: Past, present and future. *Journal of Visual Communication and Image Representation*, 10:1–23, 1999.

[64] Y. Rui, T. S. Huang, S. Mehrotra, and M. Ortega. Automatic matching tool selection via relevance feedback in mars. *Proceedings of the Second International Conference on Visual Information Systems*, pages 109–116, December 1997.

[65] Y. Rui, T. S. Huang, M. Ortega, and S. Mehrotra. Relevance feedback: A power tool in interactive content-based image retrieval. *Special issue on Segmentation, Description, and Retrieval of Video Content, IEEE Transactions on Circuits and Systems for Video Technology*, 8(5):644–655, September 1998.

[66] Y. Rui, A. C. She, and T. S. Huang. Modified fourier descriptors for shape representations – a practical approach. *Proceeding of the First International Workshop on Image Databases and Multimedia Search*, August 1996.

[67] C. L. Sable and V. Hatzivassiloglou. Text-based approaches for non-topical image categorization. *International Journal on Digital Libraries*, 3:261–275, 2000.

[68] M. Safar, C. Shahabi, and X. Sun. Image retrieval by shape: A comparative study. In *Proceedings of IEEE International Conference on Multimedia and Exposition ICME*, USA, 2000.

[69] P. Scheunders, S. Livens, G. Van de Wouwer, and D. Van Dyck. Wavelet-based texture analysis. *International Journal on Computer Science and Information Management*, 1(2):22–34, 1998.

[70] S. Sclaroff, M. La Cascia, L. Taycher, and S. Sethi. Unifying textual and visual cues for content-based image retrieval on the world wide web. *Computer Vision and Image Understanding (CVIU)*, 75(1), 1999.

[71] C. Shahabi and M. Safar. Efficient retrieval and spatial querying of 2d objects. In *IEEE International Conference on Multimedia Computing and Systems (ICMCS99)*, pages 611–617, Florence, Italy, June 1999.

[72] G. Sheikholeslami and A. Zhang. An Approach to Clustering Large Visual Databases Using Wavelet Transform. In *Proceedings of the SPIE Conference on Visual Data Exploration and Analysis IV*, pages 322–333, San Jose, February 1997.

[73] C. Silva and A. Kaufman. Pvr: High performance volume rendering. *IEEE Computational Science and Engineering*, 1996.

[74] J. K. Sircar and J. A. Cebrian. An automated approach for labeling raster digitized contour maps. *Photogrammetric Engineering and Remote Sensing*, 57(7):965–971, 1991.

[75] J. K. Sircar and J. A. Cebrian. Application of image processing techniques to the automated labeling of raster digitized contour maps. In *Proceedings of Second International Symposium on Spatial Data Handling*, pages 171–184, Seattle, Washington, July 5-10, 1986.

[76] J. R. Smith and S. Chang. Transform Features For Texture Classification and Discrimination in Large Image Databases. In *Proceedings of the IEEE International Conference on Image Processing*, pages 407–411, 1994.

[77] J. R. Smith and S.-F. Chang. Single color extraction and image query. In *Proc. IEEE Int. Conf. on Image Proc.*, 1995.

[78] John R. Smith and Shih-Fu Chang. Quad-Tree Segmentation for Texture-Based Image Query. In *Proceedings of ACM Multimedia 94*, pages 279–286, San Francisco, California, October 1994.

[79] John R. Smith and Shih-Fu Chang. VisualSeek: a fully automated content-based image query system. In *Proceedings of ACM Multimedia 96*, pages 87–98, Boston MA USA, 1996.

[80] W. E. Snyder, H. Qi, and W. Sander. A hexagonal coordinate system. In *SPIE Medical Imaging: Image Processing, Pt.1-2*, pages 716–727, February 1999.

[81] Yuqing Song, Wei Wang, and Aidong Zhang. Automatic annotation and retrieval of images. In *IFIP conference on Visual Database Systems (VDB-6), special session on Multimedia Information Management and Retrieval, Brisbane, Australia*, May 29-31, 2002. (invited paper).

[82] Yuqing Song and Aidong Zhang. Analyzing scenery images by monotonic tree. *ACM Multimedia Systems (to appear).*

[83] Yuqing Song and Aidong Zhang. Sceneryanalyzer: a system supporting semantics-based image retrieval. In *Intelligent Multimedia Documents, book edited by Chabane Djeraba, Kluwer Academic Publishers*. to appear. (invited chapter).

[84] Yuqing Song and Aidong Zhang. Monotonic tree. In *The 10th International Conference on Discrete Geometry for Computer Imagery, Bordeaux, France*, April 3-5, 2002.

[85] Yuqing Song and Aidong Zhang. Locating image background by monotonic tree. In *International Conference on Computer Vision, Pattern Recognition and Image Processing, Durham, North Carolina*, Mar 8-14, 2002.

[86] Gilbert Strang and Truong Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, Wellesley, MA, 1996.

[87] M. Stricker and A. Dimai. Color indexing with weak spatial constraints. *SPIE Proceedings*, 2670:29 – 40, February 1996.

[88] Markus Stricker and Markus Orengo. Similarity of color images. *Communications Technology Laboratory, Swiss Federal Institute of Technology, ETH CH-8092 Zurich, Switzerland*.

[89] W.D. Stromberg and T.G. Farr. A Fourier-based textural feature extraction procedure. *IEEE Transactions on Geoscience and Remote Sensing*, 24(5):722–732, 1986.

[90] M.J. Swain and D. Ballard. Color Indexing. *Int Journal of Computer Vision*, 7(1):11–32, 1991.

[91] T.F. Syeda-Mahmood. Finding shape similarity using a constrained non-rigid transform. In *International Conference on Pattern Recognition*, 1996.

[92] S. Takahashi, T. Ikeda, Y. Shinagawa, T. L. Kunii, and M. Ueda. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data. *Eurographics*, 14:c181–c192, 1995.

[93] Hideyuki Tamura, Shunji Mori, and Takashi Yamawaki. Texture Features Corresponding to Visual Perception. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(6), June 1978.

[94] Y. Tao and W.I. Grosky. Delaunay triangulation for image object indexing: A novel method for shape representation. In *Proceedings of the Seventh SPIE Symposium on Storage and Retrieval for Image and Video Databases*, pages 631–942, San Jose, California, January 1999.

[95] S. P. Tarasov and M. N. Vyalyi. Construction of contour trees in 3d in o(n log n) steps. In *Proceedings of 14th Annual ACM Symposium on Computational Geometry*, pages 68–75, 1998.

[96] G. Taubin and D. B. Cooper. Recognition and positioning of rigid objects using algebraic moment invariants. *Geometric Methods in Computer Vision, SPIE*, 1570:175–186, 1991.

[97] A.B. Torralba and A. Oliva. Semantic organization of scenes using discriminant structural templates. In *International Conference on Computer Vision (ICCV99)*, pages 1253–1258, 1999.

[98] M. Tuceryan and A. K. Jain. Texture analysis. In C. H. Chen, L. F. Pau, and P. S. P. Wang, editors, *The Handbook of Pattern Recognition and Computer Vision (2nd Edition)*, pages 207–248. World Scientific Publishing Co., 1998.

[99] M.J. Turner. Design of entropy conscious and constrained image operations using a contour tree image definition. In *Second IMA Conference on Image Processing: Mathematical Methods, Algorithms and Applications*, 1998.

[100] M.J. Turner and N.E. Wiseman. Efficient lossless image contour coding. *Computer Graphics Forum, The Eurographics Association*, 15(2):107–118, 1996.

[101] J. van den Berg. Subject retrieval in pictorial information systems. In *Proceedings of the 18th International Congress of Historical Sciences. Round Table 34. Electronic Filing, Registration, and Communication of Visual Historical Data*, pages 21–29, 1995.

[102] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for iso-surface traversal. In *Proceedings of the 13th ACM Symposium on Computational Geometry*, pages 212–220, 1997.

[103] Jia Wang and Raj Acharya. Efficient access to and retrieval from a shape image database. In *IEEE Workshop on Content Based Access of Image and Video Libraries, (CBAIL 98)*, Santa Barbara, June 1998.

[104] Jia Wang and Raj Acharya. A vertex based shape coding approach for similar shape retrieval. In *ACM symposium on Applied Computing*, pages 520–524, Atlanta, GA, Febraury 1998.

[105] C.T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. on Computers*, C-20:68–86, January 1971.

[106] Aidong Zhang, Biao Cheng, Raj Acharya, and Raghu Menon. Comparison of Wavelets and Fractals in Texture-based Image Retrievals. In *Proceedings of the SPIE Conference on Visual Data Exploration and Analysis III*, pages 116–125, San Jose, January 1996.

[107] R. Zhao and W.I. Grosky. Bridging the semantic gap in image retrieval. In T. Shih, editor, *Distributed Multimedia Databases: Techniques and Applications*, pages 13–36. Idea Group Publishing, Hershey, Pennsylvania, 2001.