SELF-DUAL MORPHOLOGICAL OPERATORS BASED ON TREE REPRESENTATIONS OF IMAGES

ALLA VICHIK

SELF-DUAL MORPHOLOGICAL OPERATORS BASED ON TREE REPRESENTATIONS OF IMAGES

RESEARCH THESIS

Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Science

in Electrical Engineering

Alla Vichik

SUBMITTED TO THE SENATE OF THE TECHNION — ISRAEL INSTITUTE OF TECHNOLOGY

NISAN, 5766 HAIFA MARCH, 2006

The Research Thesis was done under the supervision of
Dr. Renato Keshet and Prof. David Malah at the Electrical
Engineering department.

I would like to convey my deepest gratitude to Dr. Renato Keshet and Prof. David Malah for their extremely devoted guidance and support throughout the research. I would like to thank all the staff of Signal and Image Processing Lab and especially to Nimrod Peleg, for the excellent organization and the warm relation and Ziva Avni for the technical support. Finally, I thank my own family, which accompany me in all steps.

Contents

List of Tables

List of Figures

Abstract

This research addresses a new image filtering methodology, based on mathematical morphology.

In this thesis a new general framework for producing morphological, self-dual operators that are compatible to a given tree representation is proposed.

For every tree representation, a set of morphological operators on a complete inf-semilattice in the corresponding tree-representation domain is derived. Morphological erosion, opening, and opening by reconstruction operators are defined using this framework. The proposed image filtering scheme consists of three steps:

1. Transform the input image to the corresponding tree representation,

2. perform morphological operators in the tree representation domain, and

3. transform the resulting tree representation back to the image domain.

A particular case of this general framework is presented and studied. It involves a new tree representation,which we also developed in this research, called the Extrema-Watershed Tree. The particular case example emphasizes the ability of the general framework to generate new and useful sets of morphological operators.

A number of potential applications for the Extrema-Watershed Tree is proposed. The new morphological operators excel in tasks suited for the application of classical morphological operators, but that require, in addition, self-duality. The proposed applications are pre-processing for OCR (Optical Character Recognition) algorithms, de-noising of images, and preprocessing for dust and scratch detection. In addition we show that the tree has an implicit segmentation property that could be used in image segmentation algorithms.

In a previous work, Keshet has defined a complete inf-semilattice of images of alternating sequences. In this research an efficient implementation of morphological operators in this semilattice is proposed. In addition, the "trench" problem that

arises, when applying erosion and opening based on the semilattice, is studied. Possible solutions for the trench problem are proposed. Moreover, the trenches are used in order to exploit the implicit segmentation property of the Extrema-Watershed Tree.

List of Abbreviations and Symbols

SE	Structuring element
RAG	Region Adjacency Graph
AS	Alternating Sequences
BTD	Boundary Topographic Distance
BTVT	Boundary Topographic Variation Transform
OCR	Optical Character Recognition
EWT	Extrema Watershed Tree
$\mathbb{Z}$	Set of integers
$\mathbb{E}$	Euclidian space
$f, g, f(x), g(x)$	Functions from an Euclidian space to the reals
τ	Tree Transform
t	Tree
T	Tree Representation
M	Mapping function
L	Set of "labels"
V	Vertex of a tree
E	Edge connecting two vertices of a tree
$\subseteq$	Contained or equal
$\cup$	Union
$\cap$	Intersection
$\leq, \preceq, \sqsubseteq, \trianglelefteq$	Order relation in a complete semilattice
$\wedge, \curlywedge, \sqcap, \triangle$	Infimum in a complete semilattice
$\vee, \curlyvee, \sqcup, \bigtriangledown$	Supremum in a complete semilattice
δ	Dilation in a complete semilattice
ε	Erosion in a complete semilattice
γ	Opening in a complete semilattice
ϕ	Closing in a complete semilattice
Ψ	Arbitrary operator
Δ	Gray levels delta

Chapter 1

Introduction

1.1 Self-Duality

Morphological operators are widely used for image analysis and processing. The classical mathematical morphology theory was originally developed by Matheron and Serra [1], and later generalized to a complete-lattice framework by Serra [2]. A practical description of mathematical morphology and its uses can be found in Soille's book [3]. An extension of mathematical morphology from complete lattices to complete semilattices was developed by Keshet [4], and later further studied by Heijmans and Keshet in [5].

Most of existing morphological operators in complete lattices have the major disadvantage of not being self-dual. A grayscale operator ψ is called self-dual when $\psi(f) = -\psi(-f)$. Operators that are not self-dual typically do not treat bright and dark "objects" in a similar way, which is often an undesirable feature.

For example, consider the problem of denoising the simple binary image in Fig. 1.1(a). A simple erosion operation (as a first step of an opening) does remove the bright components of the noise, but the dark components are actually dilated, as seen in Fig. 1.1(b). The standard approaches of opening-closing or closing-opening approximately achieve self-duality, but are not actually self-dual; the filtering results depend on the order in which the opening and closing operators are applied. For instance, opening removes bright noise, but damages fine bright details of the image. Closing after opening can not restore the details damaged by the opening. In addition, lack of symmetry in the treatment of bright and dark components often causes damage to

the image edges, as can be seen in Figs. 1.1(c) and 1.1(d).

The most popular approach to approximately achieve self-duality in mathematical morphology is to use alternating dual operators. For instance, opening-closing and closing-opening filters, or the more general alternating sequential filters (ASF's) [1, 3]. However, these are not really self-dual, and some of the difficulties related to the lack of self-duality still appear (as in the above example of Fig. 1.1).

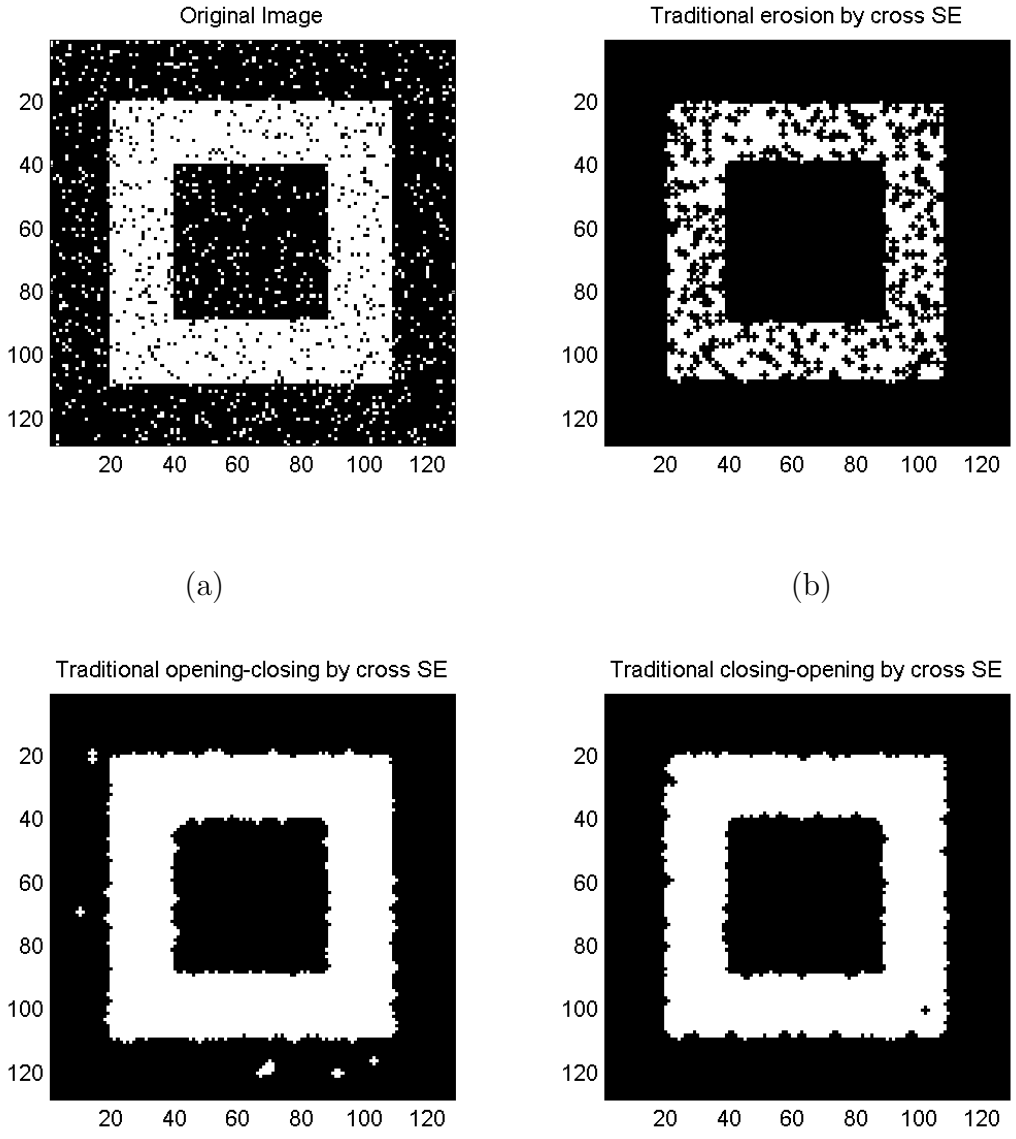
Development of self-dual morphological operators can be found, for instance, in the works of Serra, [1, chapter 8], Heijmans [6], and Mehnert & Jackway [7]. Although most of these operators are *morphological filters* (i.e., they are *idempotent and increasing*, see [1, 3]), the underlying approaches do not provide for *adjunctions*, or designing *extensive or anti-extensive* operators, see [1, 3]. As a consequence, the ability to design basic morphological operators (erosions, dilations, openings, and closings) is lost. Useful morphological tools, such as *skeletons, granulometries and gradients* [1, 3], are not available either.

Another important branch of research is that of self-dual *connected* operators. Connected operators remove objects from an image without affecting the edges of the remaining objects. In [8], Heijmans provides a summary of the activity in this area, and characterizes in depth *binary* connected operators. A summary and characterization of *grayscale* connected operators can be found in the works of Serra and Salembier [9]. A subclass of connected operators are operators based on *tree* representations. Salembier and Garrido proposed a *Binary Partition Tree* for hierarchical segmentation in [10, 11]. A *tree of shapes* was proposed by Monasse and Guichard [12, 13] (see also [14, 15]). These representations are all self-dual and very powerful. However, the above methods do not define non-connected morphological operators, such as an erosion.

1.2 Background and motivation

A new semilattice, based on the so-called *Boundary-Topographic-Variation* (BTV) Transform, was defined by Keshet in [16]. The importance of that new semilattice is that it allows to define self-dual non-connected morphological operators (like openings, erosions, etc.) without need of a reference image.

In contrast, the other morphological approaches have one or more of the following



(a) (b)

(c) (d)

Figure 1.1: (a) A noisy binary image, (b) Erosion by a 3×3 squared structuring element, (c) The result of opening-closing with the same s.e., and (d) The result of closing-opening with the same s.e.

disadvantages:

1. Are not really self-dual (see [1, 3]).
2. Allow to design filters (see [9, 8, 10, 11]), but not openings (or erosions, etc.).
3. Need a reference image (like in the reference semilattices, developed by Keshet [4], and further studied by Heijmans and Keshet in [5]).

The erosion and opening operators that are based on the BTV Transform suffer from a "trench" problem. The "trench" problem arises, especially in complex gray-scale images, when the operators are applied (see Fig. 4.3(b)). Our initial goal in this research was to solve this problem. Afterwards, the next goal was to design improved self-dual morphological operators that are not necessary connected, and to check application possibilities for the new operators.

1.3 Original contributions

Initially, an efficient implementation of self-dual morphological operators in Boundary-Topographic-Variation (BTV) domain, is proposed. In addition, the "trench" problem introduced by the BTV morphological operators is studied and a few solutions are proposed.

We also noticed that an efficient implementation of these morphological operators is achieved by first transforming the input image into an appropriate tree representation.

While working on our research, another, more robust semilattice (without trench problems) was introduced by Keshet [17]. This new semilattice is based on yet another tree representation, the tree of shapes, proposed by Monasse and Guichard [12, 13] (see also [14, 15]). The tree of shape is slightly different from that of the BTV. Notice that both the BTV and the shape tree semilattices, and their corresponding morphological operators, can be directly associated to self-dual tree representations.

There are many other self-dual tree representations in the literature. For instance, Salembier and Garrido proposed a *Binary Partition Tree* for hierarchical segmentation in [10, 11]. These tree representations are usually used for obtaining connected filtering operations on an image; however, they do not yield non-connected operators,

such as erosions, dilations or openings. It would be interesting to associate these tree representations with semilattice operators, to augment the set of imaging tools related to them.

In this research we developed a general framework for tree-based morphological image processing. This framework yields a set of new morphological operators (erosion, dilation, opening, etc.), for each given tree representation of images. Because many of the properties of the tree are inherited by the resulting morphological operators, the choice of the tree representation is of high importance. We focus mostly on self-dual trees, such as the Binary Partition Tree, which represent dark and bright elements equally. The heart of the proposed approach is a novel complete inf-semilattice of tree representations of images.

The proposed general framework is shown to unify previous schemes, and a new one is obtained by means of a new tree representation, Extrema-Watershed Tree. Following the general framework, we derive self-dual morphological operators from the Extrema-Watershed Tree, and demonstrate its application to denoising. The new morphological operators are effective in tasks suited for the application of classical morphological operators, but which require, in addition, self-duality. Moreover, we briefly study the implicit segmentation property of the Extrema-Watershed Tree, and suggest that it might be used for segmentation purposes (more research is required). "Example of applications discussed here are pre-processing for OCR (Optical Character Recognition) algorithms, de-noising of images, and feature extraction as a pre-processing for dust and scratch detection. The particular case of Extrema-Watershed Tree stresses the strength of the general framework, as a tool for generating new, useful sets of operators.

1.4 Summary and thesis organization

The main contributions of this thesis are summarized bellow:

1. An efficient implementation of the Boundary Topographic Variation (BTV) Transform, defined by Keshet in [16], was proposed.

2. The "trench" problem related to operators defined by Keshet in [16] was solved.

3. A general framework for producing morphological connected and non-connected operators associated to a given tree representations is developed.

4. A new particular case of tree representation, named Extrema-Watershed Tree, is proposed. A novel set of morphological operators was defined on the Extrema-Watershed Tree, based on this general framework.

5. Some applications using morphological filtering, based on the Extrema-Watershed Tree, are proposed. Those applications include pre-processing for OCR (Optical Character Recognition) algorithms, intermediate step for image segmentation and initial step for dust and scratch removal from images.

The thesis is organized as follows:

Chapter 2 provides a theoretical background on semilattices and graph theory, and summarizes the work presented in [17, 16] . In Chapter 3, we propose an efficient implementation of the methods defined in [16]. Chapter 4 proposes possible solutions for the "trench" problem that arises when applying erosion and opening in the semilattice defined in [16]. In Chapter 5 we develop a general framework for tree-based morphological image processing, which unifies the schemes presented in [17, 16], and enables the definition of new morphological operators that are based on tree representations. The heart of the proposed approach is a novel complete inf-semilattice of tree representations of images. For every tree representation, a complete inf-semilattice on the tree-representation domain is derived, and a set of morphological operators on that inf-semilattice is obtained.

Based on the general framework of Chapter 5, all that is needed in order to obtain a new set of morphological operators is a given tree representation. The more this tree representation is useful, the more useful these morphological operators will likely be, since many of the properties of the tree are inherited by the morphological operators (like self-duality).

Chapter 6 is devoted to the Extrema-Watershed Tree semilattice; a particular case, obtained from the general framework, and to the investigation of its usefulness.

The Extrema-Watershed Tree is a kind of "Binary Partitioning Tree", which is a state-of-the-art general framework for tree generation, developed by Salembier in [10]. It was designed in order to be used for tasks such as self-dual filtering. The

result is a new set of morphological operators, derived from the Extrema Watershed Tree, by the general framework of chapter 5.

Chapter 2

Theoretical Background

2.1 Graph theory notions

This section lists basic graph theory notions needed in the following chapters and is based on [18, chapter 1].

A *graph* is a pair $G = (V, E)$ of sets satisfying $E \subseteq [V]^2$; thus, the elements of E are 2-element subsets of V . To avoid notational ambiguities, we shall always assume tacitly that $V \bigcap E = \emptyset$. The elements of V are the vertices of the graph G, the elements of E are its vertex edges.

A *path* is a non-empty graph $P = (V, E)$ of the form:
$V = \{x_0, x_1, ..., x_k\}$ $E = \{x_0x_1, x_1x_2, ..., x_{k-1}x_k\}$,
where the x_i are all distinct. The vertices x_0 and x_k are linked by P and are called its ends. Note that k is allowed to be zero.

The *degree* of a vertex is the number of edges at that vertex. This is equal to the number of neighbors this vertex has in the graph.

A graph not containing any cycles, is called a *forest (or acyclic graph).* A connected forest is called a tree. (Thus, a forest is a graph whose components are trees.) The vertices of degree 1 in a tree are its leaves. Every non trivial tree has at least two leaves, take, for example, the ends of a longest path.

Sometimes it is convenient to consider one vertex of a tree as special; such a vertex is then called the *root* of this tree. A tree with a fixed root is a rooted tree. Choosing a root r in a tree t imposes the following partial ordering on V(t): $x \preceq y,\ if\ x \in rty$. This means that $x \preceq y$, if x belongs to the path rty, which is the path, connecting y

to the root of the tree t. This is the tree-order on $\mathrm{V}(t)$ associated with t and r. Note that r is the least element in this partial order, every leaf $x \neq r$ of t is a maximal element, the ends of any edge of t are comparable, and every set of the form $\{x | x \preceq y\}$ (where y is any fixed vertex) is a chain, a set of pairwise comparable elements. The partial order may also be applied on trees and subtrees. We say that $t_1 \preceq t_2$ if $t_1 \subseteq t_2$.

From the above definition it is clear that the infimum between vertices is the common father vertex. When given 2 vertices x and y, the infimum $z = x \wedge y$ is the vertex that is smaller or equal than x and y, thus $z \in rtx \; and \; z \in rty$, and there is no other vertex bigger than z that is smaller than x and y.

2.2 Known tree representations

Image representations can be different depending on their purpose. The raw information, that is the values of the samples, or pixels, is a too low level representation, and the image must be described by more elaborate models.

Once the image is segmented, one way or another, the resulting topology must be described. The usual notion of segmentation is a partition of the image into connected regions, also called flat zones, and the relations between these regions are meaningful.

In [19], Salembier defined *flat zones* as connected regions of the gray-level image, which are determined by a specified connectivity. In binary images these connected regions are called connected components. Each "flat zone" of a gray-level image can contain a range of gray levels or a single gray level. The range of gray levels contained in each connected region is denoted by [gray level, gray level+Δ]. For the flat zone with single gray level: $\Delta = 0$; for a range of gray levels: $\Delta > 0$. Using $\Delta > 0$, it is possible to simplify the image by quantizing it into a set of gray scales. Thus all the pixels in the "flat zone" get the same value of gray level inside the specified range. This reduces the number of flat zones in the image and can be useful for filtering and segmentation. Of course, if $\Delta > 0$, some information is lost because of the gray scale quantization.

In order to encode the adjacency relations between flat zones, we need to know when two regions have a common boundary. The classical way to represent this relation is through a graph, the Region Adjacency Graph (RAG): Each region (flat zone) is represented as a vertex in the graph and when two regions are adjacent,

an edge links the corresponding vertices. Nevertheless, adjacency is not the only meaningful relation between regions. In addition, a hierarchy between regions can be created by building a tree based on a RAG, using a merging algorithm. A merging algorithm on a RAG is simply a technique, which removes some of the links and merges the corresponding nodes, creating new regions. When two or more regions are merged, a newly created region becomes the father of the original regions.

To completely specify a merging algorithm one has to specify: the *merging order* (the order in which the links are processed), the *merging criterion* (each time a link is processed, the merging criterion decides if the merging has to be done or not), and the *region model* (when two regions are merged, the model defines how to represent the union). In the case of a Region Growing algorithm, the merging order is defined by a similarity measure between two regions (for example similar gray level), the merging criterion states that the pair of most similar regions have to be merged until a termination criterion is reached (for example a given number of regions has been obtained) and the region model is usually the mean of the pixels gray levels or color values. Note that the merging order (similarity between neighboring regions) is quite flexible and allows the definition of complex homogeneity models. By contrast, the merging criterion is very simple and crude: it states that the pair of most similar regions have always to be merged until the termination criterion is reached.

Two examples of region tree representations are presented in [19] by Salembier: Max-tree (Min-tree) and Binary Partition Tree. An additional example of image representation, called Shape-Tree, is presented in [12, 13] by P. Monasse and F. Guichard.

2.2.1 Max-Tree and Min-Tree

The Max-tree (Min-tree) is a structured representation of the image, which is oriented towards the local maxima (minima) of the image. Each node V in the tree represents a connected component of the image, which is extracted by the following thresholding process: For a given threshold T, consider the set of pixels X that have a gray level value greater or equal to T and the set of pixels Y that have a gray level value equal to T: $X = \{x,\ such\ that\ f(x) \geq T\}$, $Y = \{x,\ such\ that\ f(x) = T\}$. The tree nodes V represent the connected components C_i of X, such that $C_i \cap Y \neq \emptyset$. In other words, the nodes of the tree represent the binary connected components,

resulting from thresholding of the original image at all possible gray level values and the leaves of the Max-tree (Min-tree) correspond to the local maxima (minima) of the image. The root node corresponds to the lowest (highest) gray level value, which is the support of the entire image.

An example of a Max-tree is shown in Fig. 2.1. The original image is made of 7 flat zones: $\{A, ..., G\}$. The number following each letter defines the gray level value of the flat zones. The binary images, X , resulting from the thresholding with $0 \leq T \leq 2$ are shown in the center of the figure. Finally, the Max-tree is given in the right side. It is composed of 5 nodes that represent the connected components shown in black. The number inside each square represents the threshold value where the component was extracted. Finally, the links in the tree represent the inclusion relationships among the connected components following the threshold values. Note that, when the threshold is set to $T = 1$, the circular component does not create a connected component that is represented in the tree. This is because none of pixels of the circle G has a gray level value equal to 1, meaning that for $T = 1$, $G \cap Y = \emptyset$. However, the circle itself is obtained when $T = 2$. The maxima are represented by three leaves and the tree root represents the entire image support.

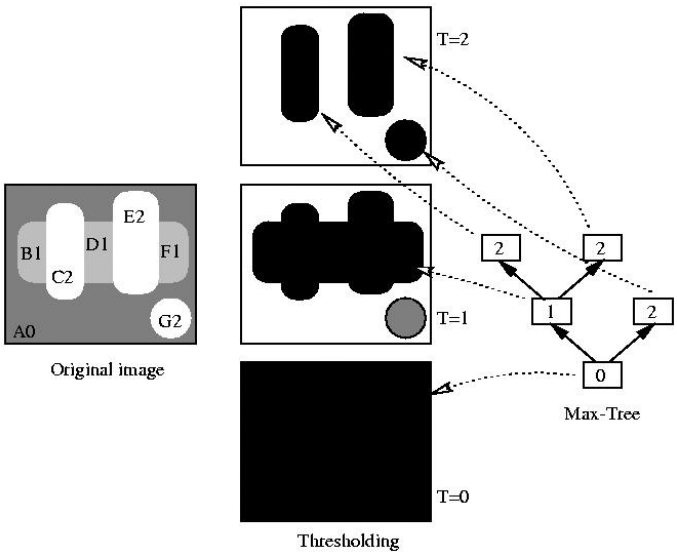


Figure 2.1: Max Tree representation of images.

2.2.2 Binary Tree

The second region-oriented image representation that is presented by Salembier in [19] is the Binary Partition Tree. The leaves of the tree represent all flat zones of the image. The remaining nodes represent regions that are obtained by merging the regions represented by the children. The root node represents the entire image support. This representation can be considered as a compromise between representation accuracy and processing efficiency. Indeed, all possibilities of flat zones merging are not represented in the tree. Only the most "useful" merging steps are represented. However, as will be seen in the sequel, the main advantage of the tree representation is that it allows a fast implementation of sophisticated processing techniques.

The Binary Partition Tree should be created in such a way that the most "useful" regions are represented. This issue can be application dependent. However, a possible solution, suitable for a large number of cases, is to create the tree by keeping track of the merging steps performed by a segmentation algorithm based on region merging. In the following, this information is called the merging sequence. Starting from the partition of flat zones, the algorithm merges neighboring regions following a homogeneity criterion until a single region is obtained. An example is shown in Fig. 2.2.

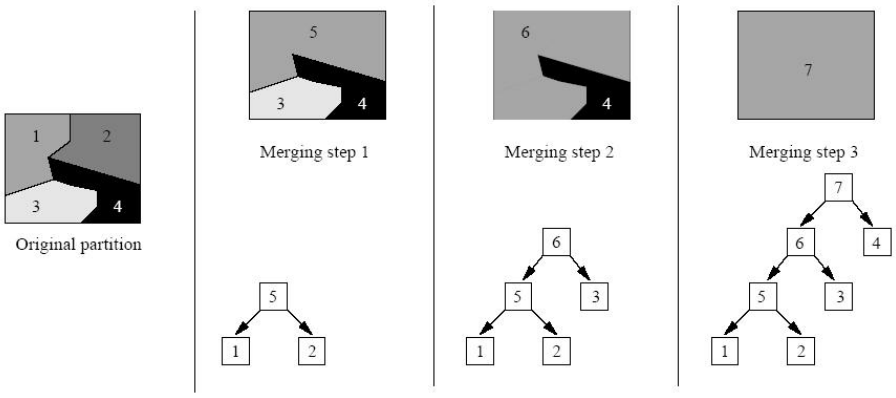


Figure 2.2: Example of Binary Partition Tree creation with a region merging algorithm.

The original partition involves four regions. The regions are indicated by numbers. Each flat zone has the different grey level value. The algorithm merges the four regions in three steps. In the first step, the pair of most similar regions, 1 and 2, are

merged to create region 5. Then, region 5 is merged with region 3 to create region 6. Finally, region 6 is merged with region 4 and this creates region 7, corresponding to the region of support of the whole image. In this example, the merging sequence is: $(1;2)|(5;3)|(6;4)$. This merging sequence progressively defines the Binary Partition Tree as shown in Fig. 2.2.

2.2.3 Tree of Shapes

The tree of shapes was proposed by P. Monasse in [20]. The tree of shapes represents an image as a hierarchy of shapes, were dark and light shapes are treated in the same way. It is build according to the inclusion order. Therefore each father vertex area includes also all sons area. An article by Monasse and Guichard [12] includes a compact definition of the tree of shapes. The following paragraphs are based on this paper.

This tree is based on upper and lower level sets: Let E be an arbitrary set, and $f : E \mapsto I\!R$. The *upper level sets* (or just level sets) of f are the collection of sets $\{\theta_t(f)\}$, $t \in I\!R$, given by:

$$\theta_t(f) \stackrel{\triangle}{=} \{x \in E | f(x) \geq t\}. \tag{2.1}$$

The sets $\{\theta_t(f)^c\}$ are called the *lower level sets*.

Given the collection of levels sets of f, the latter can be reconstructed according to:

$$f(x) = \sup\{t \in I\!R | x \in \theta_t(f)\}. \tag{2.2}$$

Note that the level sets are nested; the family of upper (resp. lower) level sets is decreasing (resp. increasing):

$$\forall \lambda \preceq t, \theta_\lambda(f) \supset \theta_t(f), \theta_\lambda(f)^c \subset \theta_t(f)^c \tag{2.3}$$

The relation (2.3) states that the level sets are nested. When going from the whole level sets to their connected components, these relations are of course still true. Now, a connected component can contain several connected components. These inclusions can be represented into a tree, as shown in Fig. 2.3(a). As we can see, the connected components of upper and lower level sets trees differ. In addition, we see that we

end up with a non natural description of the inclusion. Naturally, in the example of Fig. 2.3(a), one would have expected to have the two small squares included into the gray rectangle, and included into the white background. But the inclusion for these trees is mostly driven by the gray level rather than by the geometrical inclusion. Finally, we see that we need both trees if we want to have the two small squares represented, since each of them appears in one description, and not in the other.

Now, instead of upper and lower sets, lets us introduce a term of *shape*. A shape is a connected component of upper or lower set without any holes in it. If a hole exists, it is filled. The shape corresponds to the connected component and its filled "holes".

The sorting of shapes can then be made thanks to their geometrical inclusions. We can then create a tree structure as follows: each node corresponds to a shape; descendants are the shapes included into it, and the parent is the smallest shape that contains it (see Fig. 2.3(b)). Each shape can specify either a gray level difference between it and its father or an absolute gray level.

This will give us one single inclusion tree describing the image, in which a white object on black background is represented in the same manner as a black object on a white background.

2.3 Theoretical Background on semilattices

Mathematical Morphology is a nonlinear image processing theory, which was based on complete lattices. In [21] Keshet has extended its scope to complete semilattices, which are more general. This section provides a brief overview of Mathematical Morphology on complete semilattices.

2.3.1 Complete Semilattices and Lattices

A *partially ordered set* A is a set associated with a binary operator $\leq$, satisfying the following properties for any $x, y, z \in A$: reflexivity ($x \leq x$), anti-symmetry ($x \leq y, y \leq x \Rightarrow x = y$), and transitivity ($x \leq y, y \leq z \Rightarrow x \leq z$). In a partially ordered set A, the least majorant $\vee X$ (also called *supremum*) of a subset $X \subseteq A$ is defined as an element $a_0 \in A$, such that:

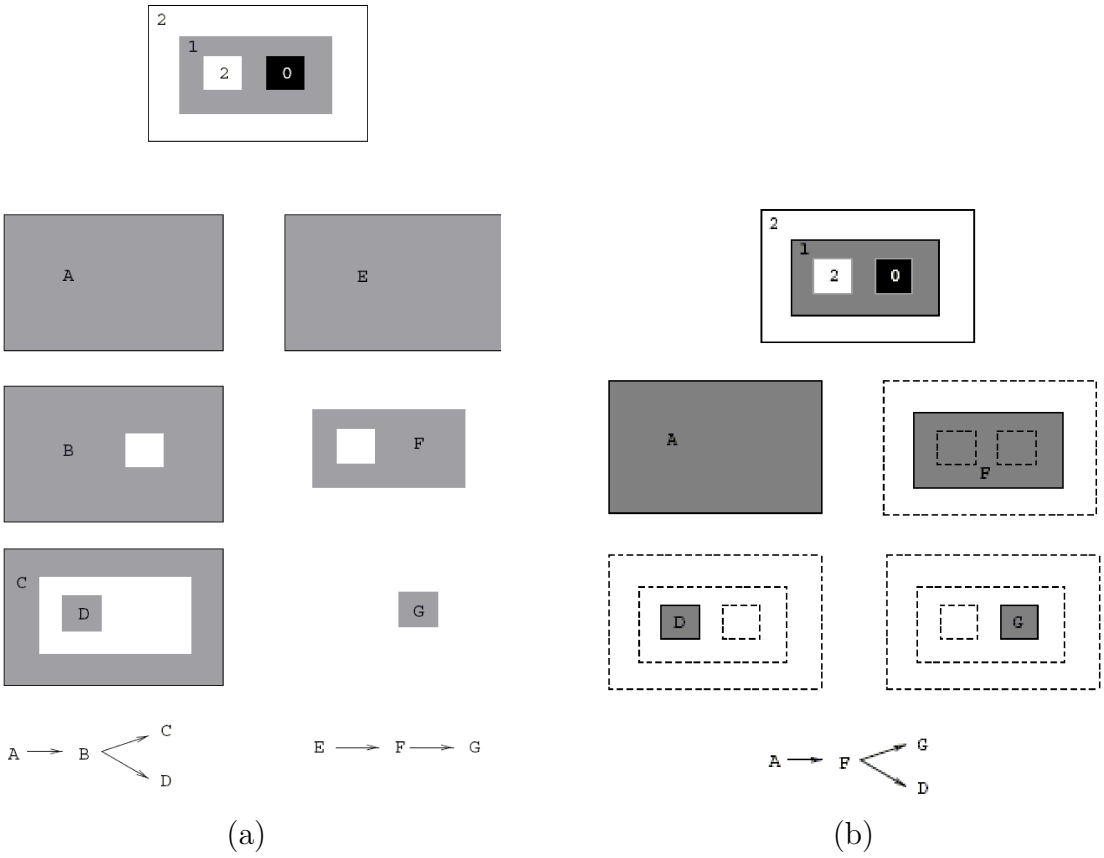


Figure 2.3: (a) The trees of connected components of upper and lower level sets of a simple image. (b) The resulting tree of shapes corresponding to the same image. Notice that D is a hole in F.

1. $x \leq a_0, \forall x \in X$,

2. if there exists y, such that $x \leq y \leq a_0$, for all $x \in X$, then $y = a_0$. One defines the greatest minorant $\wedge X$ (also called *infimum*) of X, dually.

A partially ordered set P is an *inf-semilattice* (resp. *sup-semilattice*) if every two-element subset $\{X_1, X_2\}$ in P has an infimum $X_1 \wedge X_2$ (resp., a supremum $X_1 \vee X_2$) in P. If P is both an inf and a sup-semilattice, then it is called a *lattice*. An inf-semilattice (resp., sup-semilattice) is *complete*, when *every* non-empty subset $B \subset P$ has an infimum $\wedge B$ (resp. supremum $\vee B$). In this case, there exists in the semilattice a unique element 0, called zero element, (resp., U, called universe), such that, for any $X \in P$, $0 \wedge X = 0$ (resp. $U \vee X = U$). A complete lattice is a lattice, which is both a complete inf and complete sup-semilattice.

2.3.2 Morphology on Semilattices

Some basic notions and tools of mathematical morphology on complete lattices were extended to semilattices in [21] by Keshet. In this section, we shortly review the extensions. Proofs of propositions will be omitted, but they are available in [21]. Without loss of generality, we restrict our discussion to inf-semilattices, since all definitions and results are valid in sup-semilattices as well, by duality.

Erosions and openings are basic notions that are directly and naturally extendible from complete lattices to complete inf-semilattices.

Erosion is defined in complete inf-semilattice S as any operator that distributes over the infimum, see (2.4). Preservation of the universe is not required, because it does not necessarily exist in S.

Definition 1. *A binary operator ε in an inf-semilattice S is an erosion iff, for all $\{X_i\} \subseteq S$:*

$$\varepsilon(\bigwedge_i X_i) = \bigwedge_i \varepsilon(X_i) \tag{2.4}$$

Proposition 1. *Erosions in inf-semilattices are increasing*[1].

The extension of algebraic openings to semilattices is also straightforward:

[1] A transform Ψ is increasing $\Leftrightarrow \forall f \leq g \Rightarrow \Psi(f) = \Psi(g)$.

Definition 2. *(Algebraic Opening): A binary operator in an inf-semilattice S is an algebraic opening, iff it is idempotent*[2]*, increasing, and anti-extensive*[3]*.*

Compared to the above extensions, that of morphological opening, presented below, is a little less direct. This is because: i) In complete lattices, the morphological opening associated with an erosion is defined using its adjoint dilation, and ii) since there is no general definition of supremum in an inf-semilattice, one cannot generally define dilations there (however, limited versions of supremum and adjoint dilation will be defined in the sequel). Nevertheless, morphological opening can be completely extended to complete inf-semilattices, without help of dilation.

Definition 3. *(Morphological Opening): In a complete inf-semilattice S, the morphological opening γ_ε associated with an erosion ε is defined, for any $X \in S$, by:*

$$\gamma_\varepsilon(X) \triangleq \bigwedge \{Y \in S | \varepsilon(X) \leq \varepsilon(Y)\} \tag{2.5}$$

Proposition 2. *Given any erosion in a complete inf-semilattice S, the associated morphological opening $\gamma_\varepsilon(X)$ of any element $X \in S$ exists in S and is unique.*

Proposition 3. *The morphological opening in an complete inf-semilattice is an algebraic opening, meaning it is idempotent, increasing, and anti-extensive.*

2.3.3 Supremum and Dilations

We now relate to basic morphological notions that are not directly extendible to inf-semilattices, namely, supremum and dilation. Although it is impossible to define the above operations generally in inf-semilattices, Keshet provides limited versions of them in [21].

Supremum

Unless it is a complete lattice, a complete inf semilattice does not have a well-defined supremum for all its subsets. Nevertheless, a supremum does exist for some subsets of the semilattice.

[2] A transform Ψ is idempotent $\Leftrightarrow \Psi\Psi = \Psi$.

[3] A transform Ψ is anti-extensive $\Leftrightarrow I \geq \Psi$, where I is an identity transform.

Definition 4. *Given a complete inf-semilattice S, define the set U_S of upper-bounded subsets of S as follows:*

$$U_S \triangleq \{S' \subset S | (\exists Y_0 \in S | Y_0 \geq X, \forall X \in S')\} \tag{2.6}$$

In words, U_S contains all the subsets of S that have a majorant Y_0. Supremum is defined only over elements of U_S.

Proposition 4. *The least majorant (supremum) $\vee B$ of a set $B \subset S$ exists iff $B \in U_S$, in which case it is equal to:*

$$\vee B = \bigwedge \{Y \in S | Y \geq X, \forall X \in B\} \tag{2.7}$$

The above supremum is well defined over and only over U_S. This is because the set $\{Y \in S | Y \geq X, \forall X \in B\}$ is non-empty iff $B \in U_S$. In summary, if a subset of the complete inf-semilattice has a majorant, then it has a least majorant, and this is the supremum.

Dilation

Instead of defining dilation generically (as an operator that commutes with the supremum), Keshet defines in [21] the adjoint dilation of a given erosion, restricted to a sub-domain of the semilattice. Before presenting the definition of dilation, let us characterize the domain.

Definition 5. *Given an erosion ε in a complete inf-semilattice S, we define the set of $\varepsilon-$ bounded elements of S, symbolized by S_ε, as:*

$$S_\varepsilon \triangleq \{Y \in S | [\exists X \in S | Y \leq \varepsilon(X)]\} \tag{2.8}$$

In words, S_ε contains the elements in S that are smaller or equal to the erosion of some element in S. This is the domain for the adjoint dilation of the erosion, as defined below.

Definition 6. *(Adjoint Dilation) Let S be a complete inf-semilattice, and ε an erosion. If $X \in S_\varepsilon$, then the adjoint dilation of ε, denoted δ_ε, is defined as:*

$$\delta_\varepsilon = \bigwedge \{Y \in S | X \leq \varepsilon(Y)\} \tag{2.9}$$

Note that δ_ε is well-defined over (and only over) S_ε, since $\{Y \in S | X \leq \varepsilon(Y)\}$ is non-empty iff $X \in S_\varepsilon$. We drop from now on the index ε from the notation of its adjoint dilation, for simplification.

Proposition 5. *The adjoint dilation of ε is increasing in S_ε.*

Proposition 6. *For all $X \in S, \gamma(X) = \delta\varepsilon(X)$.*

2.4 Shape Tree Semilattice Background

A type of complete inf-semilattice called Semilattice of Images of Shape Sequences and a set of morphological operations in it, is defined in [17] by Keshet. In this section, a theoretical background on Shape Tree Semilattice is provided.

2.4.1 Tree of Shapes

As was mentioned in section 2.2.3, Monasse and Guichard define the *shapes* of an image f in [12, 13] as the collection $\mathcal{T}$ of sets given by (according to our notation):

$$\phi\left[\gamma_x\left(\theta_t\left(f\right)\right)\right] \text{ and } \phi\left[\gamma_x\left(\theta_t\left(f\right)^c\right)\right], \tag{2.10}$$

for all $x \in E$, and $t \in \mathbb{R}$.

Fig. 2.4 shows a simple grayscale image, and its associated shapes.

The above researchers show that, for $E = \mathbb{R}^2$ and the usual topological connectivity, every two shapes either contain one an other or are disjoint. This provides $\mathcal{T}$ with a tree structure, where the parent of every shape $\tau \in \mathcal{T}$ is the smallest shape that contains τ. They call the resulting tree the *tree of shapes* of f (whose details were investigated by Ballester, Caselles, and Monasse in [15]), and they show that it is a contrast-invariant representation. Fig. 2.5(a) depicts the tree of shapes corresponding to the above example.

In [12], a fast algorithm for calculating the tree of shapes of discrete images is presented. The algorithm takes natural image (i.e., function with range $\mathbb{N}$), and returns the tree $\mathcal{T}$, along with the sign of the differences $s(\tau)$ (either $+1$ or -1) between the levels of each shape τ and its parent $p(\tau)$, as well as the smallest shape $\tau(x)$ that contains the pixel x. This information permits the reconstruction of the

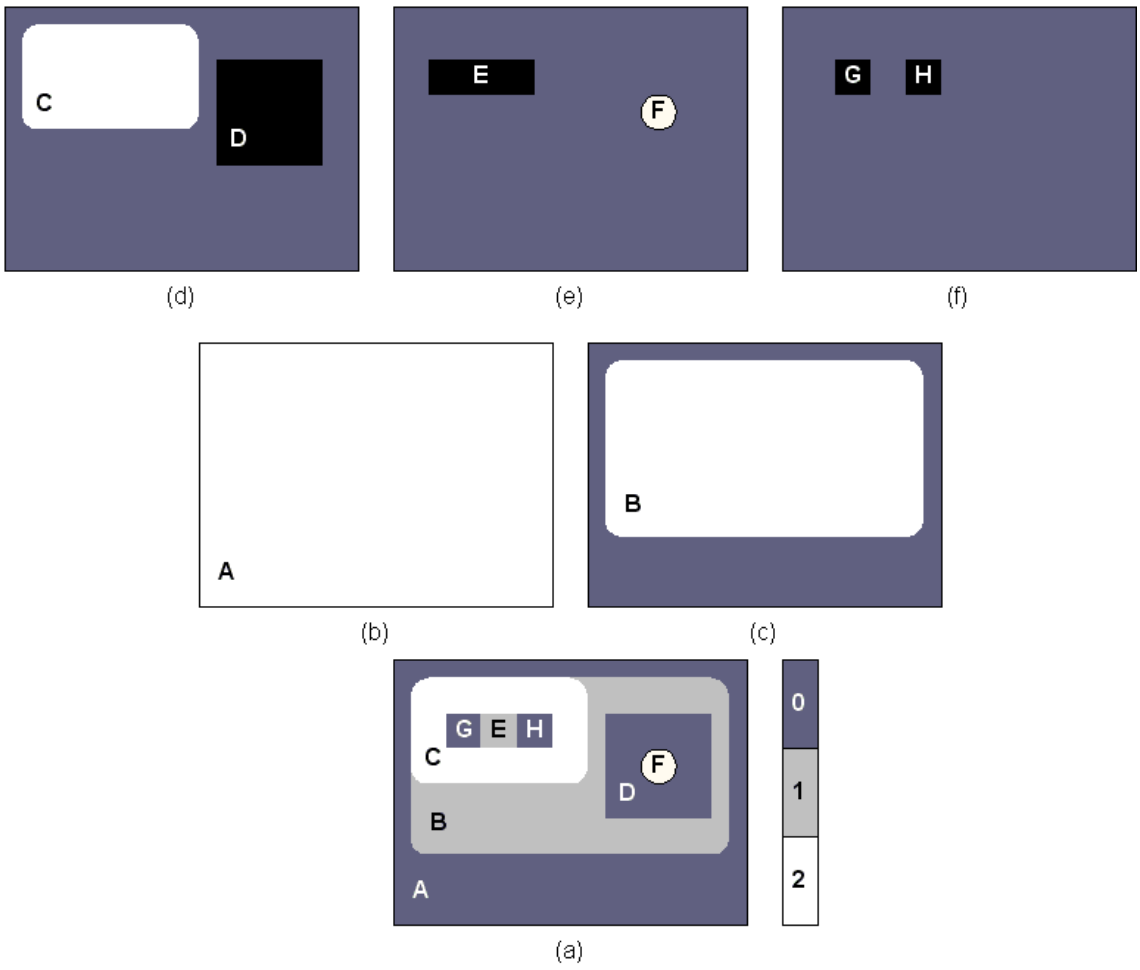


Figure 2.4: Shape decomposition and associated Tree of shapes. (a) Original gray-scale image f and (b)-(f) its shapes. The white shapes correspond to connected components of the collection of upper levels sets $\{\theta_t(f)\}$, whereas the black shapes correspond to connected components of the collection of lower level sets $\{\theta_t(f)^c\}$.

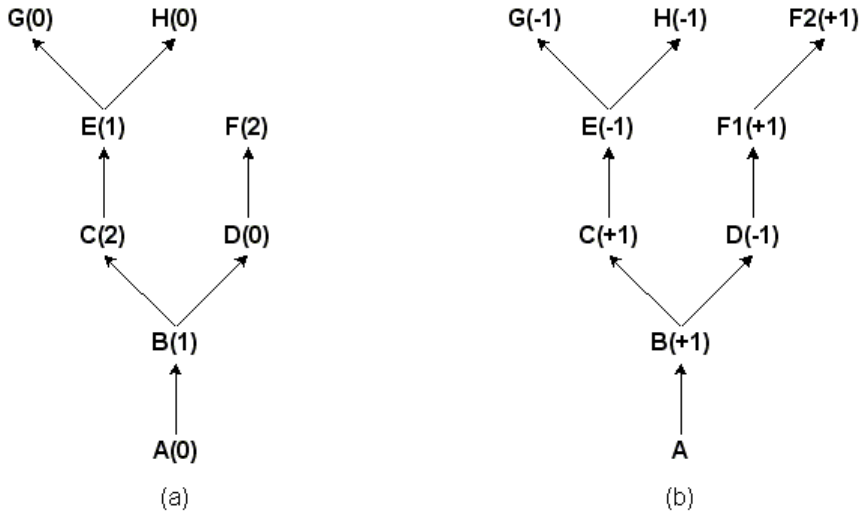


Figure 2.5: Tree of shapes associated with Fig. 2.4. (a) Original tree; the numbers in parenthesis are the gray-levels of the shapes. (b) Modified tree; the number in parenthesis are the differences $s(\tau)$ between the levels of each shape τ and its parent $p(\tau)$. Notice that shape `F` was duplicated.

original image, up to a local change of contrast. In order to perfectly reconstruct the original image, one can also store the original gray levels of the shapes [as shown in Fig. 2.5(a)], or modify the tree by replicating the shapes for all gray-levels they appear [as shown in Fig. 2.5(b)]. In the latter situation (which is the one adopted here), $f(x)$ is equal to the sum of $s(\tau)$ for all shapes that contain $\tau(x)$.

2.4.2 Inf-Semilattice of Binary Sequences

Keshet defines in [17] a partial ordering of images that takes into consideration the data of the tree of shapes. However, he does not use the tree itself in order to provide such ordering. He proposes a representation that keeps the datum of the tree readily available, but makes it simpler to compare images, and develops an inf-semilattice of binary sequences.

Consider the set $\mathcal{S}$ formed by the null sequence and all finite binary sequences, with elements in $\{+1, -1\}$.

Definition 7. *Let $s = \{s_n\}, r = \{r_n\} \in \mathcal{S}$. Sequence s is a* prefix *of sequence r, marked $s \trianglelefteq r$, when $\ell(s) \leq \ell(r)$ and $s_n = r_n$, $n = 1, \ldots, \ell(s)$, where $\ell(s)$ is the length*

of the sequence s.

For instance, $\{-1,+1,-1\} \trianglelefteq \{-1,+1,-1,+1,+1\}$, and $\{\} \trianglelefteq \{1,1\}$.

Proposition 7. *The prefix relation $\trianglelefteq$ is a partial ordering, and $(\mathcal{S}, \trianglelefteq)$ is a complete inf-semilattice, where the null sequence $\{\}$ is the least element, and the infimum $s \bigtriangledown r$ is given by the greatest common prefix, i.e.:*

$$(s \bigtriangledown r)_n = s_n, \quad n = 1, \ldots, L(s,r), \tag{2.11}$$

where

$$L(s,r) = \sup \{n \in I\!N, n \leq \min [\ell(s), \ell(r)] \mid \ s_n = r_n\}. \tag{2.12}$$

The proof is given in [17].

In order to facilitate notation in the sequel, trailing zeros can be added to all binary sequences. I.e., let $\mathcal{Z}$ be the operator that maps each sequence into an infinite counterpart by appending an infinite number of zeros to it. The operator $\mathcal{Z}$ is invertible, and the inverse $\mathcal{Z}^{-1}$ consists of removing all zero elements of an appended sequence. Considering the set $\mathcal{Z}(\mathcal{S})$ of all 0-appended sequences, then it also forms a complete inf-semilattice with respect to the order given by the concatenation $\trianglelefteq \circ \mathcal{Z}^{-1}$. In the remainder of the section, sequences are referred in this complete inf-semilattice only, and use $\trianglelefteq$ to denote the partial order of 0-appended sequences as well. Moreover, the length of a 0-appended sequence will be by convention the length of its finite counterpart, i.e., $\ell(s)$ actually meaning $\ell(\mathcal{Z}^{-1}s)$.

2.4.3 Images of Shape Sequences

Images of Shape Sequences are defined in [17] by Keshet.

Definition 8. *Let the* image of shape sequences *associated with an image f be the mapping $\mathbf{s}_f : E \mapsto \mathcal{Z}(\mathcal{S})$, such that $\mathbf{s}_f(x)$ is the sequence of differences $s(\tau)$ for all shapes that contain $\tau(x)$, ordered from the largest to the smallest, followed by zeros.*

For instance, referring to the image in Fig. 2.4(a), and its modified tree in Fig. 2.5(b), for all x in region `G`, set $\mathbf{s}_f(x) = \{+1,+1,-1,-1,0,\ldots\}$, for all x in `D` and not in `F`, $\mathbf{s}_f(x) = \{+1,-1,0,\ldots\}$, and for all x in `F`, $\mathbf{s}_f(x) = \{+1,-1,+1,+1,0,\ldots\}$.

Let the operator, that maps f to $\mathbf{s}_f$, to be denoted by $\mathcal{U}$.

The original function f can be reconstructed from $\mathbf{s}$ by:

$$f(x) = \sum_{n=1}^{\infty} [\mathbf{s}_f(x)]_n. \tag{2.13}$$

Therefore, by defining, $\mathcal{U}^*\{\mathbf{s}\} \triangleq \sum_{n=1}^{\infty} [\mathbf{s}(x)]_n$ for an arbitrary image of sequences $\mathbf{s} : E \mapsto \mathcal{Z}(\mathcal{S})$, we have $\mathcal{U}^{-1}\{\mathbf{r}\} = \mathcal{U}^*\{\mathbf{r}\}$ for any image of sequences $\mathbf{r}$ in the range of $\mathcal{U}$.

The datum of the image of sequences $\mathbf{s}_f$ associated to f is identical to that of the tree of shapes of f. Indeed, all pixels in a shape have a common sequence prefix, and all connected components of pixels sharing a common prefix is a shape.

2.4.4 Inf-Semilattice of Images of Shape Sequences

In [17], Keshet defines a type of complete inf-semilattice called Semilattice of Images of Shape Sequences and a set of morphological operations in it.

Definition 9. *Let the binary relation $\sqsubseteq$ be given by:*

$$f \sqsubseteq g \iff \mathbf{s}_f(x) \trianglelefteq \mathbf{s}_g(x), \quad \forall x \in E. \tag{2.14}$$

Select a fixed point in E, and denote it by o.

Theorem 1. *Let $\tilde{F}_{\mathbb{N}}(E)$ be the set of all discrete natural functions, for which $f(o) = 0$. Then according to [17], $(\tilde{F}_{\mathbb{N}}(E), \sqsubseteq)$ is a complete inf-semilattice, with infimum given by*

$$[f \sqcap g](x) = \sum_{n=1}^{\infty} \left[\mathbf{s}_f(x) \bigtriangledown \mathbf{s}_g(x)\right]_n. \tag{2.15}$$

Notice that, even though the binary case provides us with a lattice, in the gray-scale case the supremum is not well defined, which leaves us with an inf-semilattice only. The reason for that stems from the inf-semilattice structure of the set of sequences. Indeed, if f and g are given for instance by the images in Fig. 2.6(a) and (b), respectively, then the infimum $f \bigtriangledown g$ is given by the image in Fig. 2.6(c), because the

central regions in those images are characterized by the sequences $\{+1, -1, 0, \ldots\}$ and $\{+1, +1, 0, \ldots\}$, respectively, whose infimum is given by $\{+1, 0, \ldots\}$. The supremum $f \triangle g$, on the other hand, does not exist in this case.

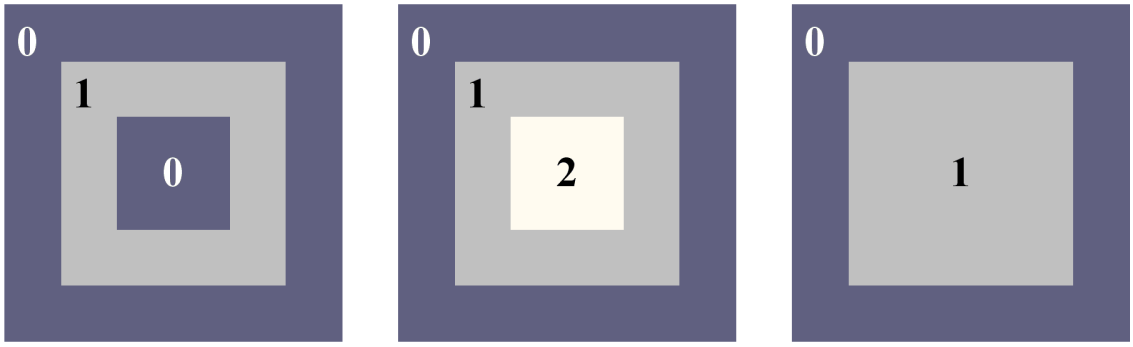


Figure 2.6: (a)-(b) Two grayscale images, and (c) their infimum according to $\sqsubseteq$. The supremum does not exist in this case.

Here is another way of understanding why the binary case lends itself to a lattice framework: Binary images are represented by *alternating* binary sequences starting with $+1$. These are fully characterized by their length alone, and are completely ordered.

Even though working with an inf-semilattice means that dilations can not be defined in all cases, the inf-semilattice theory described in [4, 5] assures that a great deal can be accomplished nevertheless. Not only can erosions be defined, but also openings, internal gradients, white top-hats, skeletons, openings by reconstruction, etc.

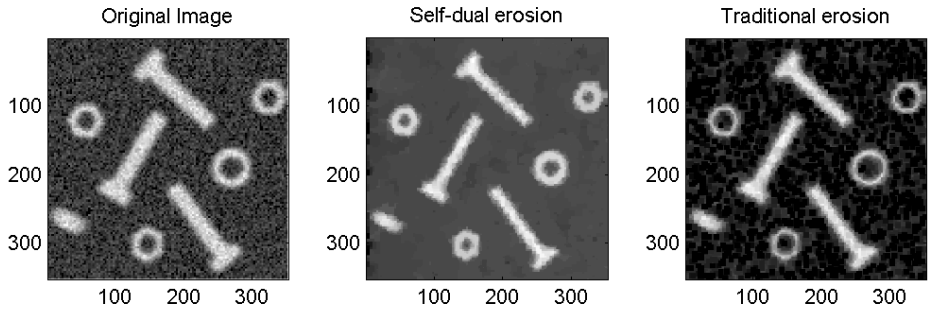
Let us define the translation-invariant erosion:

$$\hat{\varepsilon}_S(f) = \mathcal{U}^{-1}\left\{\bigtriangledown_{y \in \check{S}} \mathcal{U}\{f_y\}\right\}, \tag{2.16}$$

where f_y denotes the translation of f by y: $f_y(x) = f(x - y)$. Fig. 2.7 shows an example of using the above "shape erosion" on a gray-scale image, compared to using the standard gray-scale erosion.

The adjoint dilation is given by

$$\hat{\delta}_S(f) = \bigtriangledown\{g \mid \ f \sqsubseteq \hat{\varepsilon}_S(g)\}, \tag{2.17}$$



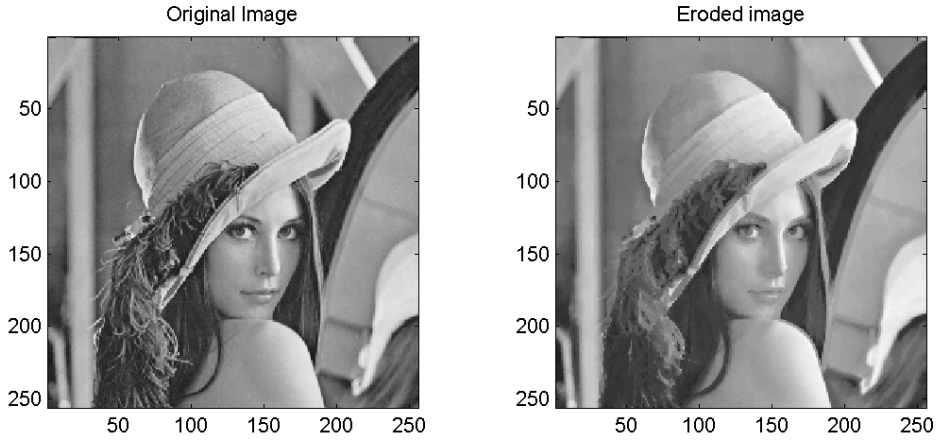
(a) (b) (c)

Figure 2.7: Erosion in the complete inf-semilattice of shape sequences. (a) Original gray-scale image, (b) proposed self-dual erosion, and (c) standard erosion. The structuring element in both cases is the cross 7×7.

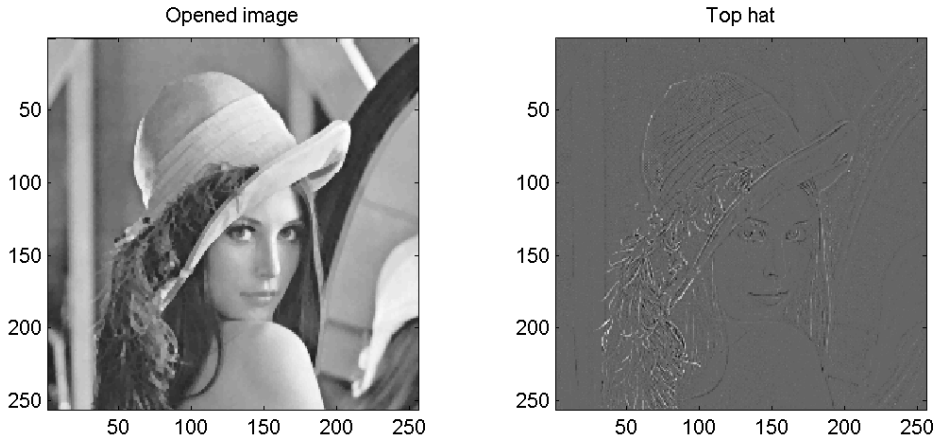
which always exists if f is the erosion of some other function [4]. In fact, the adjoint dilation can be computed by $\hat{\delta}_S(f) = \mathcal{U}^{-1}\{\triangle_{y \in S}\mathcal{U}\{f_y\}\}$, where $\triangle$ is the trivial supremum—defined only when all operands are point-wise comparable w.r.t. $\trianglelefteq$, in which case it returns the largest. When f is in the range of the erosion $\hat{\varepsilon}_S$, then all operands during the computation of the adjoint dilation are point-wise comparable.

Like for any erosion in an inf-semilattice, the opening operator $\hat{\gamma}_S$ associated to $\hat{\varepsilon}_B$ is well defined in all cases, and given by $\hat{\gamma}_S = \hat{\delta}_S\hat{\varepsilon}_S$.

Fig. 2.8 illustrates the result of applying the erosion, its adjoint dilation (yielding the opening), and the corresponding top-hat transform to a complex grayscale image.



(a) (b)



(c) (d)

Figure 2.8: (a) Original grayscale image, (b) shape-tree erosion, (c) opening, and (d) top-hat. A square structuring element of size 2×2 was used in all cases.

2.4.5 Conclusion

A new, self-dual approach for morphological image processing was proposed by Keshet in [17]. It is based on the tree of shapes, and its resulting morphological framework is that of a inf-semilattice. In the discrete case this semilattice is complete. The corresponding erosion has the effect of shrinking all the elements of the image, regardless to their contrast. In order to compare images in this inf-semilattice, Keshet proposes a new representation of images in terms of binary sequences, which keeps the information displayed in the tree of shapes readily available.

The binary version of our scheme is related to the adjacency tree, which is the binary counterpart of the tree of shapes. In this case, a lattice framework (complete in the discrete case) is obtained.

2.5 Alternating Sequences Semilattice Background

A type of complete inf-semilattice called Alternating Sequences (AS) Semilattice and a set of morphological operations in it, is defined in [16] by Keshet. In this chapter, the theoretical background on AS Semilattice is provided.

2.5.1 Boundary Topographic Distance (BTD)

The topographic distance between two pixels x and y of an image f, defined by Meyer in [22], is the least total variation needed to walk from x to y (or vice-versa) on the topographic relief defined by f.

In [16], Keshet utilizes a simplified definition of topographic distance, in which the cost of walking on a topographic function f between two consecutive points p_1 and p_2 is given by:

$$cost(p_1, p_2) = |f(p_1) - f(p_2)|. \tag{2.18}$$

For instance, suppose that a connected path between x and y has the following values on f: $f(x = p_1, p_2, \ldots, p_6, p_7 = y) = (1, 4, 8, 5, 2, 3, 5)$. The total variation on that path is given by the sum of absolute differences between consecutive values, which is equal to 16 in this case. If no other connected path linking x to y has smaller total variation, then 16 is the topographic distance between x and y.

Keshet also defines in [16] the notion of *boundary topographic distance* (BTD) of a pixel x of a *bounded* image f, as the topographic distance between x and the boundary of f. That is, the BTD of x is the least topographic distance between x and any point on the boundary of f. The BTD *function*, BT_f, is the mapping from each pixel x to its BTD, $BT_f(x)$.

2.5.2 An extension of boundary topographic distance definition

The notion of BTD, defined by Keshet, can be easily extended, allowing the *boundary* of the image f to be any connected flat zone of f. The notion of flat zone is discussed in section 2.2. In some cases, a boundary of an image isn't the best starting point for topographic distance calculation, because it is arbitrary set by the properties and position of the camera used to take the picture. When choosing another flat zone, as a reference for the topographic distance, the choice can be more meaningful and suited for the problem to be solved.

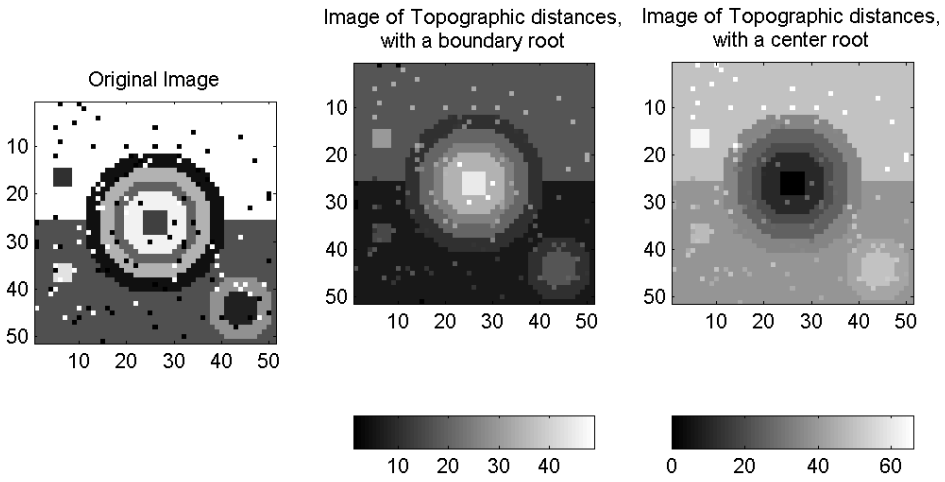
For instance, in some applications it can be useful to define the boundary as the largest flat zone of the image. Another example is shown in Fig. 2.9. In this example a root flat zone is chosen to a be a symmetric center of the image - the circle center.

2.5.3 Boundary Topographic Variation (BTV) Transform

Let $\pi_f(x)$ be a connected path linking the boundary of an image f to a pixel x and $\hat{\pi}_f(x)$ be one such path, with least topographic distance between x and the boundary. The topographic distance (total variation) on $\hat{\pi}_f(x)$ is exactly $BT_f(x)$ and $\hat{\pi}_f(x)$ is called a minimal-BTD path for x. There may be more than one minimal-BTD path for a given point x.

Assume that a minimal-BTD path is associated with an *alternating sequence* (AS). The AS describes the "ups and downs" that occur on that path. These "ups and downs" are called *path variation*. For instance, consider a grayscale image f, and suppose that a minimal-BTD path $\hat{\pi}_f(x)$ from the boundary to a pixel x has the following function values:

$$f(\hat{\pi}_f(x)) = (0, 2, 6, 9, 8, 8, 5, 1, 3, 7, 7, 4, 5). \tag{2.19}$$



(a) (b) (c)

Figure 2.9: An example of different topographic distance reference. (a) An original image. (b) An image of a topographic distance from the boundary. (c) An image of a topographic distance from a center flat zone.

This path is divided into the following overlapping monotonic sub-sequences: $(0, 2, 6, 9)$, $(9, 8, 8, 5, 1)$, $(1, 3, 7, 7)$, $(7, 4, 5)$. The corresponding path variation is given by:

$$V(\hat{\pi}_f(x)) = \{9, -8, 6, -3, 1\}, \tag{2.20}$$

meaning that one has to climb 9 gray levels, following the monotonic sub-sequence $(0, 2, 6, 9)$, then go down 8, following the next monotonic sub-sequence $(9, 8, 8, 5, 1)$, and so on for the whole path.

Assuming that $f(x) \geq 0$, $\forall x$, and that the boundary of f has null values, then the first element of a path variation is always non-negative.

The mapping $\mathcal{V} : f \mapsto V_f$, where $V_f(x) \stackrel{\triangle}{=} V(\hat{\pi}_f(x))$, is called the *boundary-topographic-variation (BTV) transform*. Because it is based on the topographic distance, the BTV transform can be calculated fast, by modifying a fast (topographic-distance-based) implementation of the watershed algorithm.

As was noted before, the minimal boundary path is not necessarily unique, which means that there maybe more than one BTV transform for a given image f. In practice, different BTV transforms are identical up to a relatively small number of

x	$f(x)$	$BT_f(x)$	$\hat{\pi}_x$	$[\mathcal{V}(f)](x)$
0	0	0	(0)	$\{0\}$
1	-2	2	$(0,-2)$	$\{-2\}$
2	7	11	$(0,-2,7)$	$\{-2,9\}$
3	9	13	$(0,-2,7,9)$	$\{-2,11\}$
4	11	15	$(0,-2,7,9,11)$	$\{-2,13\}$
5	10	16	$(0,-2,7,9,11,10)$ $(0,3,8,5,10)$	$\{-2,13,-1\}$ $\{8,-3,5\}$
6	5	11	$(0,3,8,5)$	$\{8,-3\}$
7	8	8	$(0,3,8)$	$\{8\}$
8	3	3	$(0,3)$	$\{3\}$
9	0	0	(0)	$\{0\}$

Table 2.1: The BTV transform of a 1-D function f. The boundary here are the first and last elements of the function. Notice that the point $x = 5$ is a skeleton (or watershed) point of the transform, with two different transform possibilities.

pixels that are called *skeleton* pixels or *watershed* pixels.

Table 2.1 presents a simple 1-D case for illustration.

The plot of $f(x)$ is given in Fig. 2.10.

The original image f can be obtain back from $V_f(x)$ as follows:

$$f(x) = \sum_i V_f(x)_i, \tag{2.21}$$

where $\{V_f(x)_i\}$ are the elements of the alternating sequence $V_f(x) = [\mathcal{V}(f)](x)$. Equation (2.21) represents the inverse BTV transform, $\mathcal{V}^{-1}$.

It is also simple to obtain the BTD function from $\mathcal{V}(f)$:

$$BT_f(x) = \sum_i |V_f(x)_i| \,. \tag{2.22}$$

For the example given in table 2.1, for $x = 5$, $[\mathcal{V}(f)](x = 5)$ is $\{-2, 13, -1\}$ or $\{8, -3, 5\}$. For both variants of $[\mathcal{V}(f)](x = 5)$:
$f(x) = (-2) + 13 + (-1) = 8 + (-3) + 5 = 10$,
$BT_f(x) = |-2| + |13| + |-1| = |8| + |-3| + |5| = 16$.

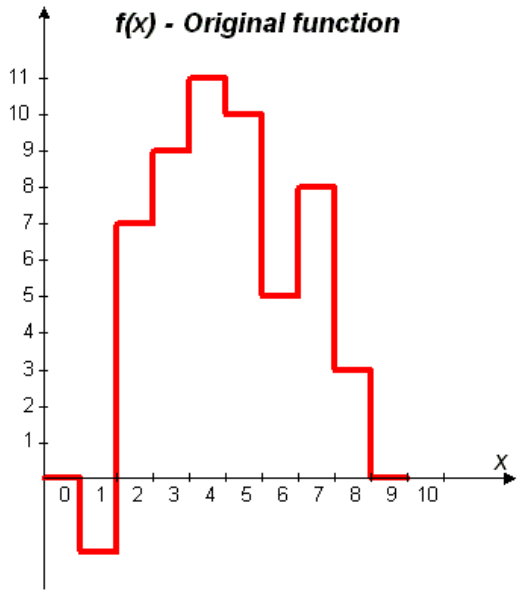


Figure 2.10: Plot of 1-D Function $f(x)$.

2.5.4 Semilattice in BTV Domain

In order to define morphological operations in the BTV transform domain, a complete inf-semilattice of variations (alternating sequences) was defined in [16] by means of the following partial ordering. Let V_1 and V_2 be two alternating sequences with lengths L_1 and L_2, respectively.

$$V_1 \sqsubseteq V_2 \iff \begin{cases} (V_1)_i = (V_2)_i, & \forall i < L_1, \\ |(V_1)_{L_1}| \leq |(V_2)_{L_1}|. \end{cases} \tag{2.23}$$

For instance, $\{9,-2\} \sqsubseteq \{9,-3,8,4\}$, $\{0\} \sqsubseteq \{1,-2\}$ and $\{7,-2,5\} \sqsubseteq \{7,-2,6,-1,10\}$, but $\{7,-2,7\} \not\sqsubseteq \{7,-2,6,-1,10\}$ and $\{1,-3,-5\} \not\sqsubseteq \{1,-3,4,-6,7\}$.

The infimum operation associated to the above partial ordering is the common prefix, followed by the the weakest of the next elements. More precisely,

$$V_1 \sqcap V_2 = \{P(V_1,V_2), \text{med}[(V_1)_{L_P+1}, (V_2)_{L_P+1}, 0]\}, \tag{2.24}$$

where $P(V_1,V_2)$ is the common prefix of V_1 and V_2, L_P is the length of $P(V_1,V_2)$, and med is the median operation.

The trivial supremum $\sqcup$ of two alternating sequences V_1 and V_2 is given by:

$$V_1 \sqcup V_2 = \begin{cases} V_1, & V_2 \sqsubseteq V_1, \\ V_2, & V_1 \sqsubseteq V_2, \\ \not\exists, & \text{otherwise.} \end{cases} \tag{2.25}$$

Table 2.2 exemplifies calculations of the infimum and the supremum of pairs of alternating sequences.

	Relation	Infimum	Supremum
0	$\{9,-2,8,-4\} \sqsubseteq \{9,-2,8,-4,7\}$	$\{9,-2,8,-4\}$	$\{9,-2,8,-4,7\}$
1	$\{-1,7,-9,2\} \not\sqsubseteq \{-5,4,-7,1,-3\}$	$\{-1\}$	$\not\exists$
2	$\{7,-2,5\} \sqsubseteq \{7,-2,6,-1,10\}$	$\{7,-2,5\}$	$\{7,-2,6,-1,10\}$
3	$\{7,-2,7\} \not\sqsubseteq \{7,-2,6,-1,10\}$	$\{7,-2,6\}$	$\not\exists$
4	$\{1,-3,5\} \not\sqsubseteq \{-4,6,-2,8,-3\}$	$\{0\}$	$\not\exists$

Table 2.2: Infimum and supremum of pairs of alternating sequences.

The following operator can now be defined:

$$\hat{\varepsilon}_B(f) = \mathcal{V}^{-1}\left\{\sqcap_{y\in B^s}\mathcal{V}\{f_y\}\right\}, \tag{2.26}$$

where f_y denotes the translation of f by y: $f_y(x) = f(x-y)$. It can be shown that the above operator is an erosion in the complete inf-semilattices of BTV-transform images. The adjoint dilation is given by

$$\hat{\delta}_B(f) = \mathcal{V}^{-1}\left\{\sqcup_{y\in B}\mathcal{V}\{f_y\}\right\}, \tag{2.27}$$

Like for any erosion in a complete inf-semilattice, the opening operator $\hat{\gamma}_B$ associated to $\hat{\varepsilon}_B$ is well defined and is given by $\hat{\gamma}_B = \hat{\delta}_B\hat{\varepsilon}_B$.

Chapter 3

Implementation of BTVT

In this chapter, we propose an efficient implementation of the BTV transform defined in section 2.5.3. To this end, we introduce below (section 3.1) what we call a *Topographic Distance Tree* (TD-Tree). Each node of this tree relates to a flat zone of the image with its corresponding gray level value and topographic distance to the boundary. The TD-Tree is an efficient representation of an image, from which the boundary topographic variation of each flat zone can be easily computed. Therefore, we utilize the TD-Tree for BTV Transform (BTVT) implementation. In addition, we show a few examples of BTV-based morphological filtering of sample images.

3.1 Topographic Distance Tree definition

In this section, we propose an intermediate representation for the process of BTVT computation. It consists of a tree representation, which we call the Topographic Distance Tree, or TD-Tree for short. The TD-Tree is useful for calculating the alternating sequences, which are a key component of the BTVT.

Let us first define a TD-graph as (V_{TD}, E_{TD}). Each node of a TD-graph has a gray level function $GL(v)$, which holds its corresponding gray level value and a topographic distance value, which holds its topographic distance to the boundary. If RAG is a set (V_{RAG}, E_{RAG}), then the TD-graph is a sub-graph of the RAG: $V_{TD} = V_{RAG}$, $E_{TD} \subseteq E_{RAG}$. The edge set E_{TD} includes all edges that belong to a minimal topographic distance path of every flat zone of an image. The skeleton points in this graph have more than one father because there is more than one minimal topographic distance

path for those flat zones. A TD-tree (V_{TDT}, E_{TDT}) is a sub-graph of the TD-graph. The vertex set is the same, but $E_{TDT} \subseteq E_{TD}$. A TD-tree includes no skeleton points, because all multiple minimal pathes were removed, leaving a single path for every flat zone. In the TD-Tree, the root node is the boundary of the image.

An example of this tree representation is shown in Fig. 3.1. In this example, the source image consists of 4 flat zones: a background component V1, a small simple component V2 located on the background, and a complex component V3 that includes a component V4. The resulting tree, shown in the same figure, contains information about topographic distance and gray level of each flat zone.

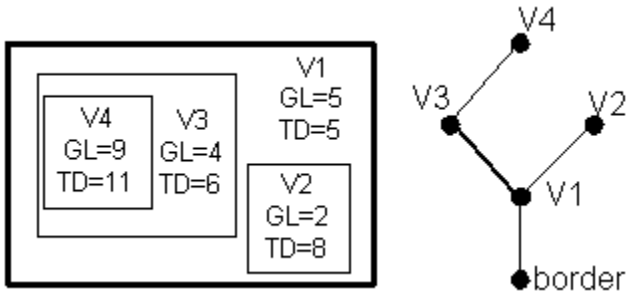


Figure 3.1: An example of a given image and its TD-tree. Each flat zone of the image corresponds to a node of the TD-tree (indicated by the letter V, followed by the node number), its gray-level (shown in the variable GL), and its boundary topographic distance (shown in the variable TD).

3.2 Topographic Distance Tree implementation

Our implementation of the TD-Tree is based on the algorithm of Moore, reviewed in [22]. This is an efficient algorithm for finding the topographic distance function of a given flat zone of an image. The original algorithm of Moore calculates the topographic distance, but does not keep the path variation of each flat zone. Therefore, we have adapted the algorithm of Moore, so that not only does it calculate the topographic distance, but also builds the TD-Tree. Another proposed modification of the Moore algorithm is the use of the modified cost function described in (2.18). In every iteration of the Moore algorithm, the value of the topographic distance function for a new flat zone is computed. This new flat zone is the nearest neighbor (according

to the topographic-distance) of a previously processed flat zone. The new flat zone then becomes a new node of the TD-Tree, linked as a son to the latter flat zone. Its topographic-distance-function value is computed as that of the father plus the topographic-distance between them.

A pseudo code of the modified Moore algorithm is given in Algorithm 1.

Algorithm 1 Modified Moore Algorithm

We modified algorithm of Moore, for topographic distance calculation, using definition (2.18) of cost. Our modified algorithm is used not only for topographic distance calculation, but also for TD-Tree building.
The algorithm, involves the following stages:

1: Assign zero to the topographic distance of the root node of the TD-Tree, and, to the other nodes, assign an infinite topographic distance.
2: Initialize status vector of all flat zones to status "in-queue".
3: **while** not all nodes are "done" **do**
4: Find the node with the minimal topographic distance.
5: Based on this minimal topographic distance, calculate the topographic distance of the neighboring nodes, whose status are "in-queue", using the graph structure. Store the node with the minimal topographic distance as a father node of these neighboring nodes in the TD-Tree.
6: In the status vector, assign status "done" to the node with minimal topographic distance.
7: **end while**

The proposed representation is more efficient in storing the information about alternating sequences than storing an alternating sequence for every flat zone. The storage volume required for the alternating sequence for every flat zone depends on the number of flat zones multiplied by the mean alternating sequence length. $Size = O(N \cdot \overline{Length(AS)}) \simeq O(N^2)$, where N is a number of flat zones. Typical values of N is in order of 10^4 for small images, and in order of 10^5 for larger images. When the same information is presented as a tree, the required storage volume depends on the number of flat zones multiplied by the size of the pointer to the father node, which is much smaller than the average size of an alternate sequence. In this case the required storage volume is linear to N: $Size \simeq O(N)$.

3.3 Implementation of the BTV Transform

The first phase of the algorithm is to create a Region Adjacency Graph - RAG. The RAG is a graph (V, E), where V is a set of all flat zones of the image, and E contains all pairs of flat zones that are adjacent to each other. A theoretical background on graphs and trees is given in section 2.1. We developed an algorithm for computing the RAG image representation. A pseudo code of the algorithm is given in Appendix A.

The next step is to use the modified Moore algorithm for creation of a TD-Tree. The modification of the Moore algorithm was explained in section 3.1.

In order to calculate an alternating sequence of every flat zone, we must fetch the path variation of a component. The path variation fetch function, is a recursive function, as can be seen in (3.1).

$$AS(V) = \begin{cases} \{AS(father(V)), GL(V) - GL(father(V))\} & if\ V \neq root \\ \{0\} & if\ V = root \end{cases} \tag{3.1}$$

Where $AS(V)$ is an alternating sequence of vertex V in the TD-tree. $GL(V)$ is the gray level of V. $father(V)$ is the vertex preceding V in the TD-tree. For example, $AS(V_2)$ in Fig. 3.1 gives an alternating sequence of $\{AS(V_1), -3\}$ in the first recursion step, and $\{5, -3\}$ in the next (and last) recursion step.

The alternating sequence computation can be performed as a preliminary step before any operators are applied, or during an execution of an operator. The choice between the two options depends on a specific tradeoff between storage size and execution time.

In summary, the proposed algorithm for the implementation of the BTV transform consists of three stages:

1. The input image is represented as a Regional Adjacency Graph (RAG) of the flat zones partition.

2. Based on this RAG representation, the topographic distance is calculated and the TD-Tree is obtained, using the modified Moore algorithm.

3. The BTV transform is calculated, using the TD-Tree.

Images of topographic distances, computed for synthetic and natural images, using the modified Moore algorithm, are shown in Figures 3.2-3.5. The images are built

from the TD-Tree, where the gray level of each flat zone is taken to be a topographic distance of that flat zone. This technique visualizes the resulting topographic distance and helps to understand properties of an image. Because the computed topographic distances are relative to the image boundary, the values of the images of topographic distances generally increase as one moves from the boundary towards the center of the region of support of the image. The TD is constant for all pixels inside the flat zone of the image. This is clearly visible in the synthetic images. But, it is hard to see this in the natural images, because gray levels change gradually and flat zones are very small.

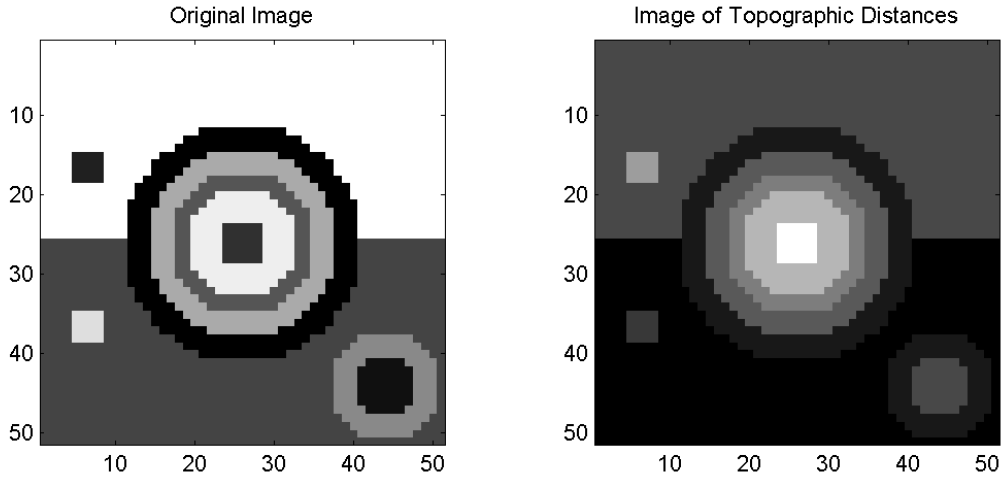


Figure 3.2: Synthetic image with the corresponding image of topographic distances.

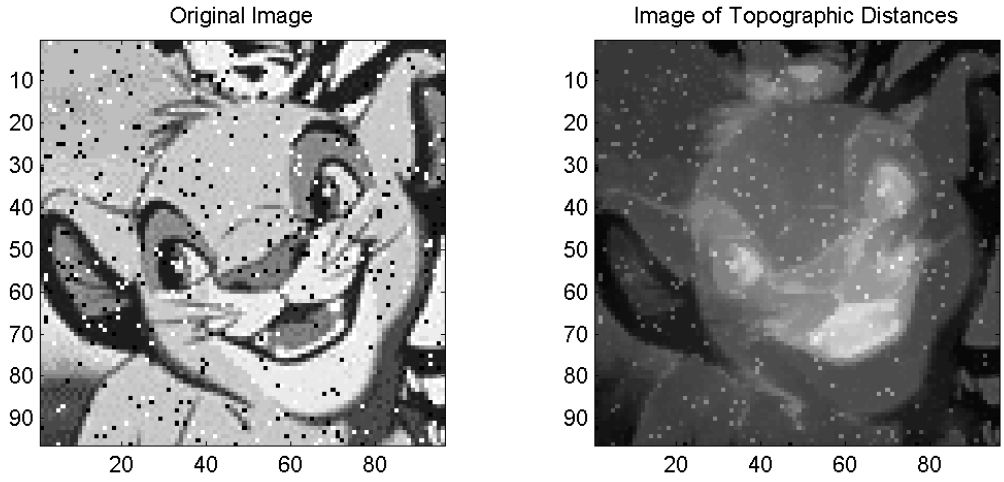


Figure 3.3: Noisy image of Simba with the corresponding image of topographic distances.

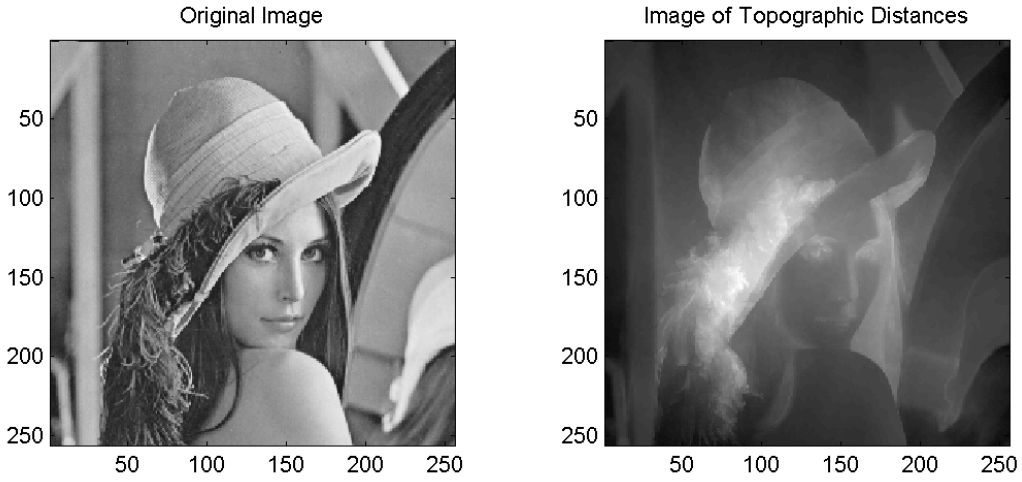


Figure 3.4: Image of Lena 256x256 with the corresponding image of topographic distances.

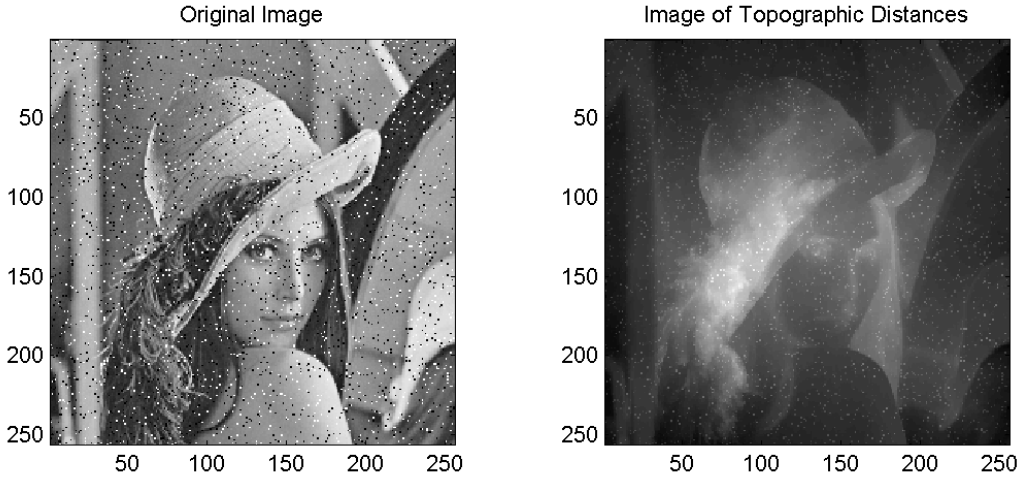


Figure 3.5: Noisy Image of Lena 256x256 with the corresponding image of topographic distances.

Chapter 4

The "trench" problem and the proposed solutions

The BTV-domain morphological operators (erosion, opening etc.) proposed by Keshet in [16] are self-dual. This means that those operators treat bright and dark "objects" similarly. However, when applying those operators, a "trench" problem may arise. This chapter studies this problem, and proposes solutions for it.

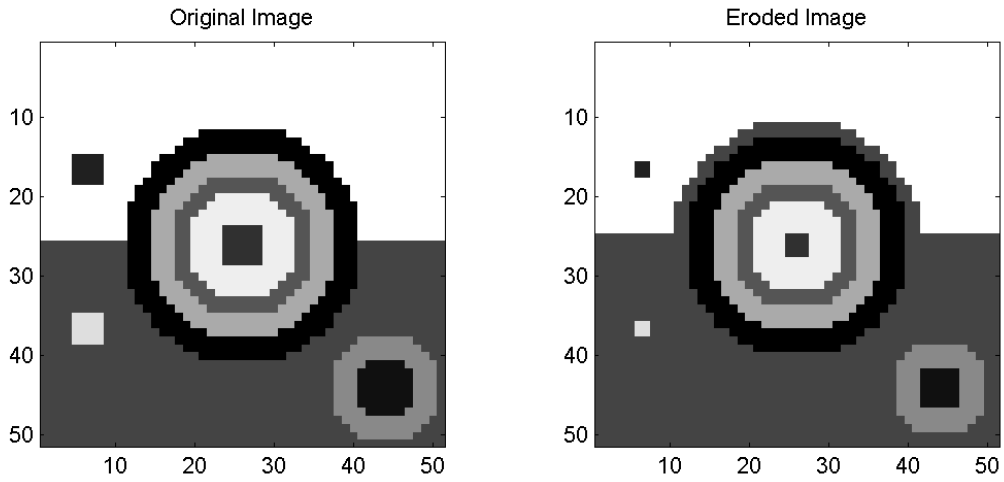
The BTV-domain morphological filtering approach involves three steps. First, the BTV transform is performed on the input image, using the TD-tree representation of the image. Then the transformed image is simplified using morphological filtering, based on the partial ordering of images of alternating sequences, defined in Section 2.5.4. Finally output image is constructed from the processed image by an inverse BTV transform, using Equation (2.21).

Figs. 4.1 and 4.3 show the output of the erosion $\hat{\varepsilon}_B$ on two test images. Figs. 4.2 and 4.4 show the corresponding opening $\hat{\gamma}_B$ of the same two test images.

Notice that the results of self-dual erosion and opening (Figs. 4.1-4.2) on the synthetic image are as expected. The corresponding erosion has the effect of shrinking all the elements of the image, regardless to their contrast. The opening rounds the corners that SE can not get into in all the elements of the image, bright and dark.

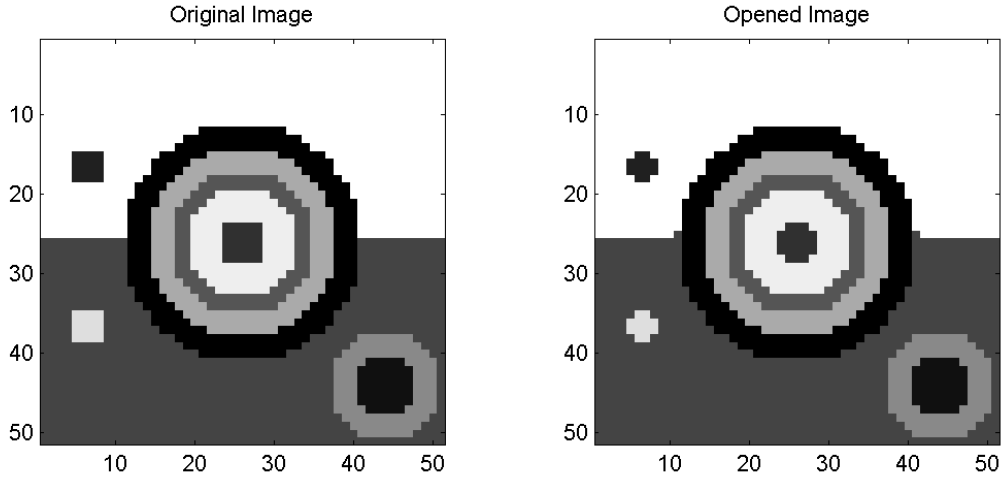
However, the results on the natural image are very strange; the output shows many "trenches" that do not appear to have any justification to exist.

These are typical results. When the number of gray levels is small, and the image is not very complex, the BTV-based erosion returns useful results. However, when



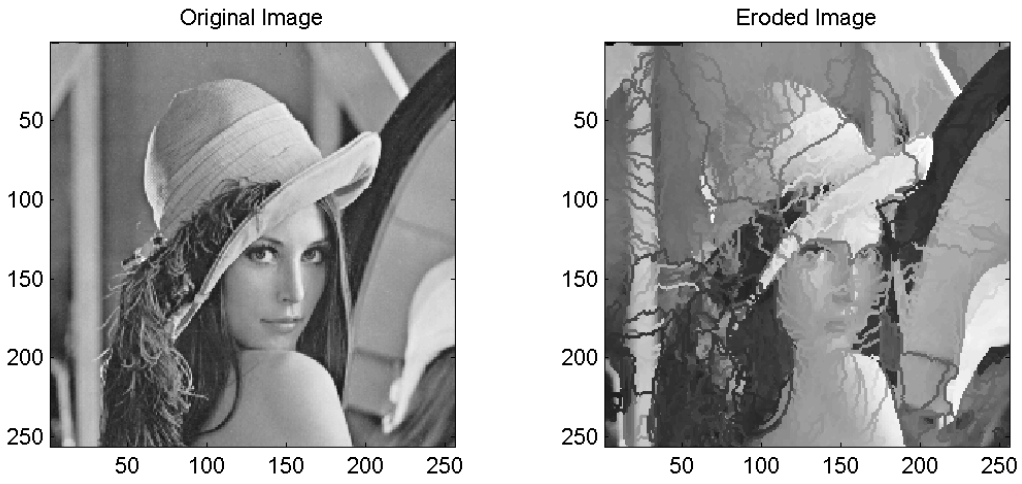
(a) (b)

Figure 4.1: Result of BTV-based erosion $\hat{\varepsilon}_B$. (a) Original Synthetic Image, and (b) Erosion of Synthetic Image for a 2×2 structuring element.



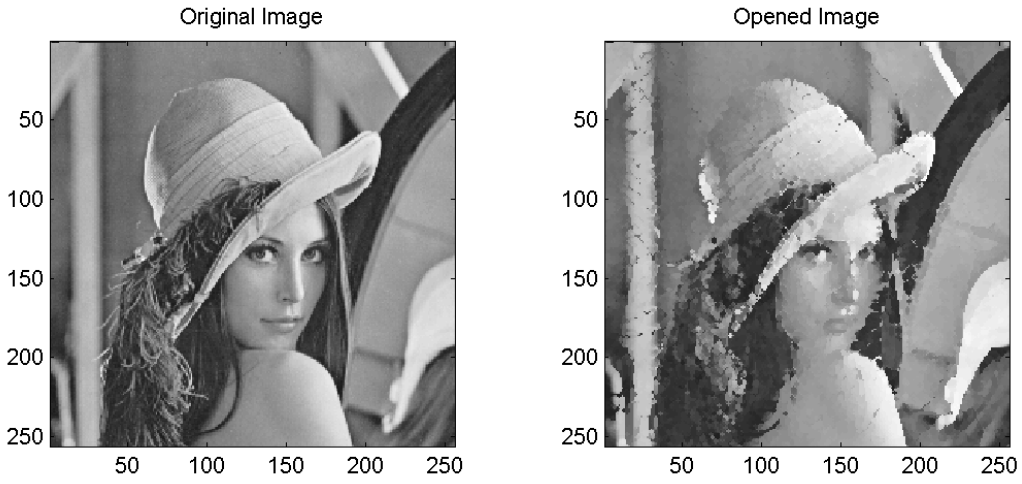
(a) (b)

Figure 4.2: Result of BTV-based opening $\hat{\gamma}_B$. (a) Original Synthetic Image, and (b) Opening of Synthetic Image for a 2×2 structuring element.



(a) (b)

Figure 4.3: Results of BTV-based erosion $\hat{\varepsilon}_B$. (a) Original Natural Image, and (b) erosion of Natural Image for a 2×2 structuring element.



(a) (b)

Figure 4.4: Results of BTV-based opening $\hat{\gamma}_B$. (a) Original Natural Image, and (b) opening of Natural Image for a 2×2 structuring element.

the number of gray levels is high, or if the image is very complex, the BTV transform usually fails. The reason for that is the existence of skeleton (watershed) points in the BTV data, which have neighbors with potentially very dissimilar alternating sequences; when eroded, a "trench" may open at these points, because the infimum of the dissimilar alternating sequences yields a sequence that does not characterize any of the neighbors. Although some of the "trenches" disappear, when we perform opening instead of erosion, the results are still not good, see Fig. 4.4.

Let us consider a one dimentional example, given in Table 2.1, in order to understand the origin of the trenches. In this example, $x = 5$ is a skeleton (or watershed) point of the BTV transform of function $f(x)$, with two different transform possibilities $\{-2, 13, -1\}$ and $\{8, -3, 5\}$. For $x = 4$ the alternating sequence is $\{-2, 13\}$. Therefore, the infimum between two neighbors $\{-2, 13\}$ and $\{8, -3, 5\}$ is $\{0\}$. This produce the trench at $x = 4$.

Fig. 4.5 shows the output of the erosion $\hat{\varepsilon}_B$ on the one dimentional function $f(x)$.

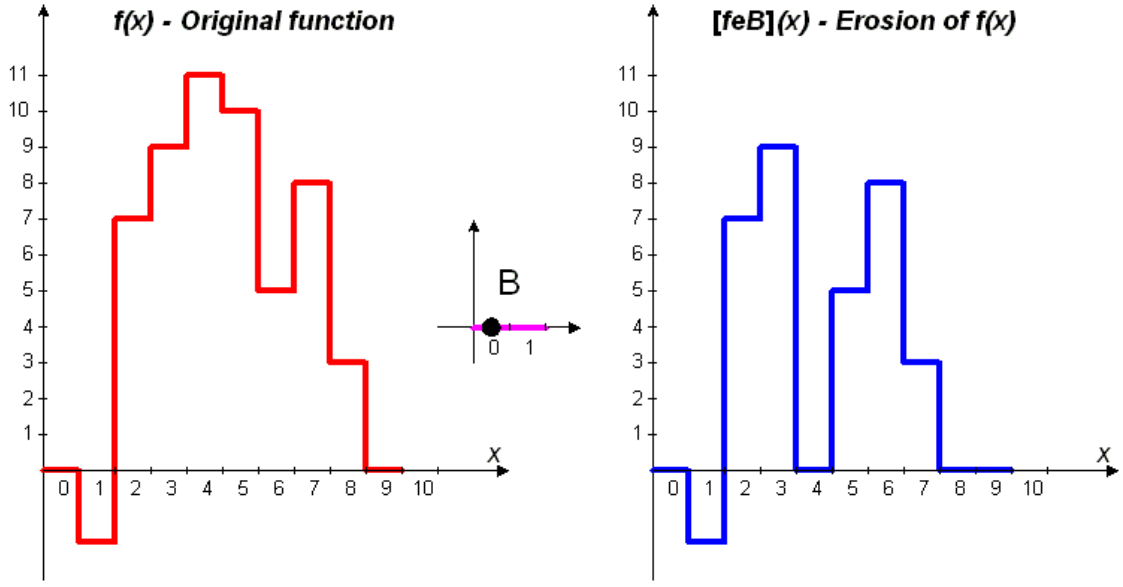


Figure 4.5: Results of BTV-based erosion $\hat{\varepsilon}_B$. (a) Function $f(x)$, and (b) Erosion of $f(x)$.

Solutions for the "trench" problem are proposed in the following sub-sections.

4.1 Filtering using an adaptive structuring element

One of the solutions to the trench problem is, first, to choose smartly only a single variant of the BTV transform. For instance, if there are two flat zones with the same topographic distance, which potentially can be the fathers of the current node in the TD-Tree, choose the one that contains the maximum number of pixels. In our example shown in Fig. 4.6, vertices 4 and 6 can be fathers of vertex 5. We choose the vertex 6 to be the father - this option is drawn with a solid line, and discard the relation to the vertex 4, that is drawn with a dashed line.

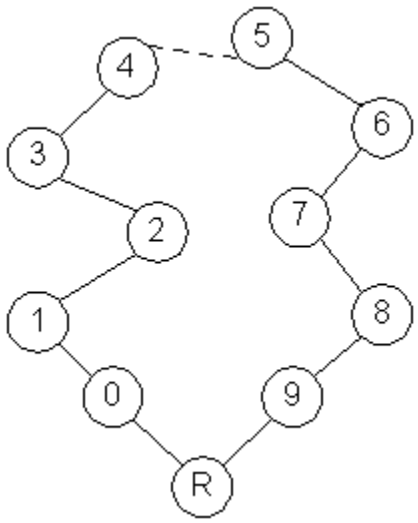
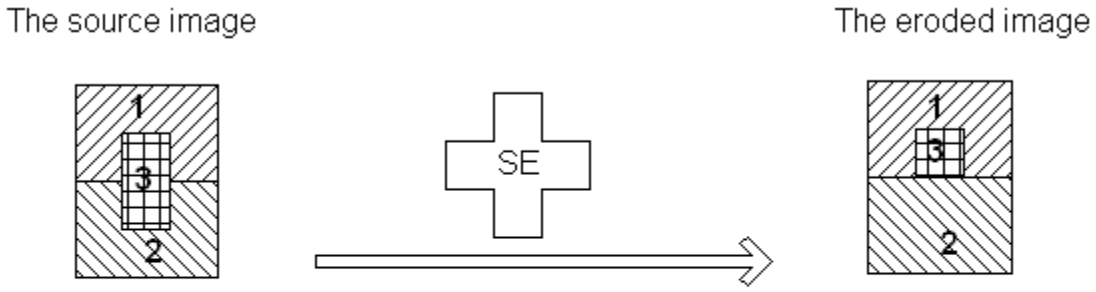


Figure 4.6: A tree representing the BTV transform of function $f(x)$.

Second, use an *adaptive structuring element*, constraining the *filtering depth*, during the filtering operation. That is, while computing the infimum of the neighborhood, do not take into consideration neighboring pixels that cause the path-variation to be pruned by more than a given filtering depth. For example, if the filtering depth is limited to 1, then, while performing erosion of $x = 4$, we ignore the presence of the point $x = 5$ and make a trivial erosion with itself (in the 2D case it usually won't happen). For this method the result will be $\{-2, 13\}$

The use of an adaptive SE has some drawbacks. For instance, in some applications it will be necessary to store this adaptive SE for every pixel. But, in case of opening it can be implemented without the need to store the SE. Another drawback is that the adaptive SE, although prevents a creation of trenches, limits the performance of

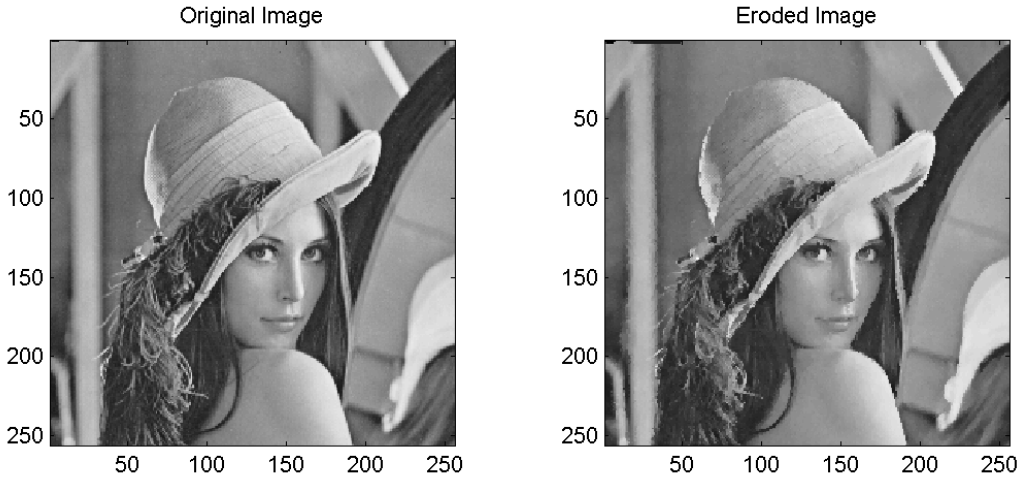
the resulting filter. As an example, Fig. 4.10 shows the filter effect for a filtering depth limit of 1 on a noisy image of Lena. Notice that a number of noisy pixels have survived the filtering. This artifact shows the typical problem of this approach. Fig. 4.7 shows a simple example that explains the effect. In the image shown, the upper pixel of component 3 is connected only to component 1. When in the erosion process the shown structure element (SE) is placed on this pixel, the infimum is taken between {103,-6,4} and {103,-5,2,-100}. The resulting infimum is {103,-5}. That infimum length is 2, and the alternating sequences are of length 3 and 4, respectively. Therefore, the length difference is 2, that is more than the filtering depth limit. In that case the noisy pixel is left untouched because the resulting infimum is much shorter then the source alternating sequence.



	Alternating seq.	Gray level
1	{103,-6,4}	101
2	{103,-5,2}	100
3	{103,-5,2,-100}	0

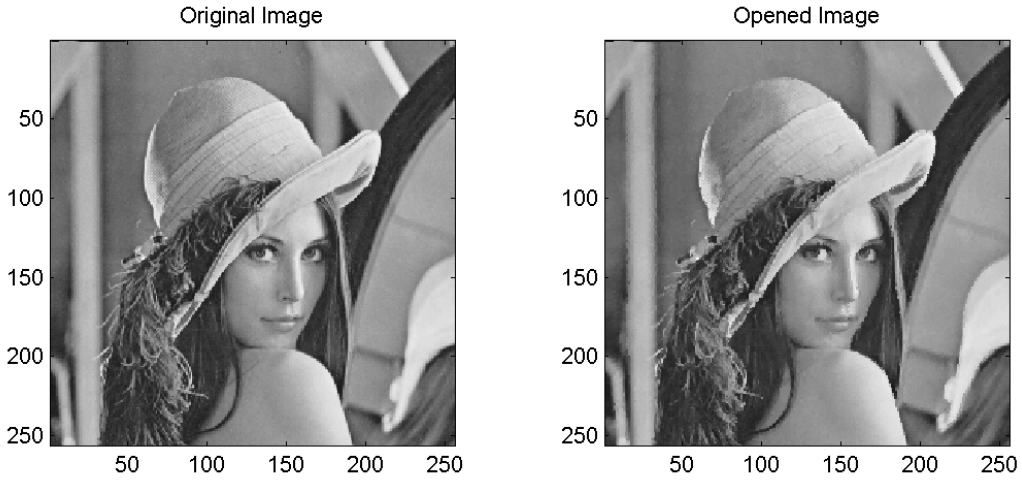
Figure 4.7: An example of filtering a dual-pixel noise : A black component - 3, is located on brighter background - 1 and 2. After erosion with erosion depth limit of 1, the upper pixel of the 3-rd component is untouched.

This problem can be solved if we use a greater filtering depth limit. As an example, in Fig. 4.11, the adaptive structuring element method succeeds to filter more salt and pepper noise, but some of the “trenches” remain. Another solution is presented in section 4.3.



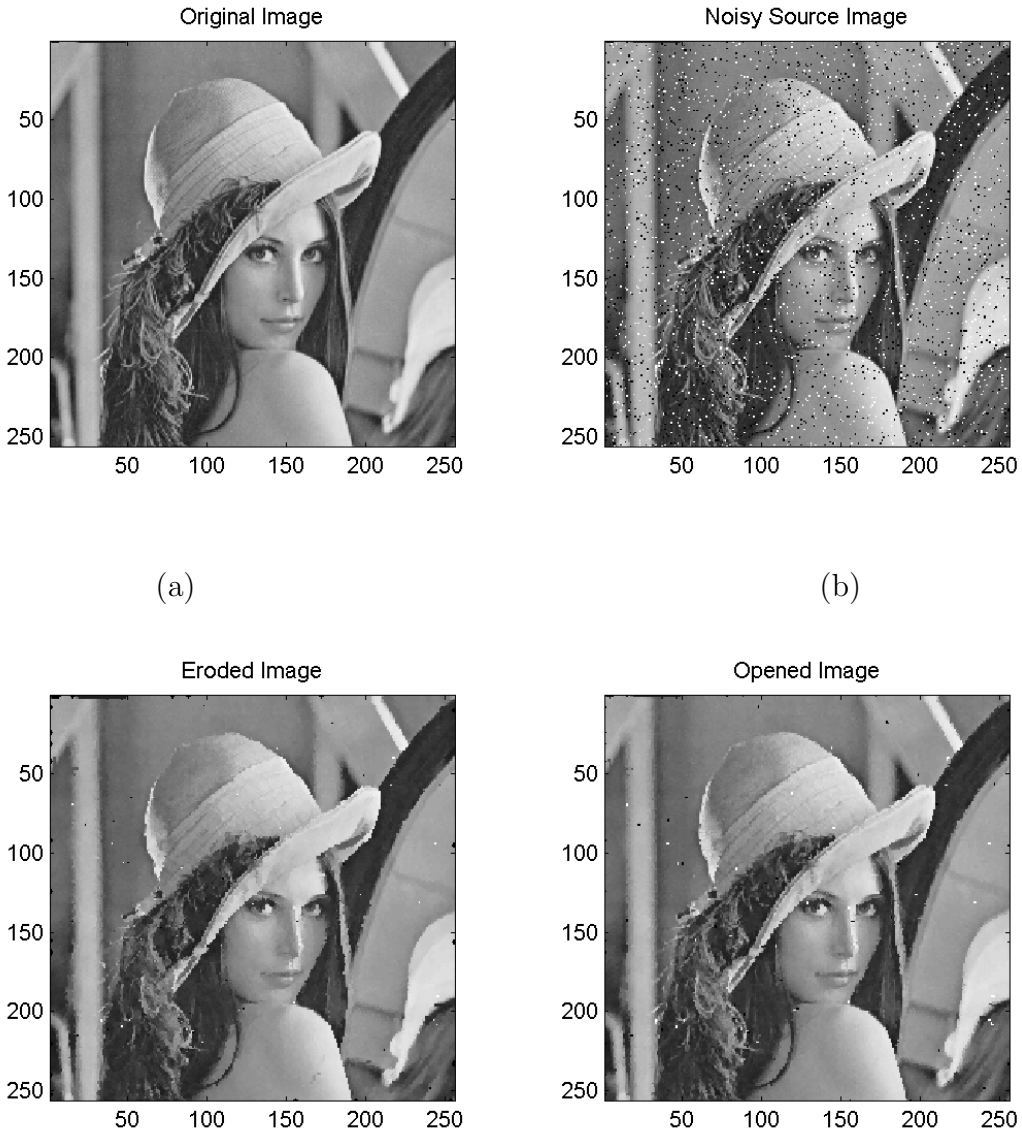
(a) (b)

Figure 4.8: Results of BTV-based erosion $\hat{\varepsilon}_B$, using adaptive SE. (a) Original Natural Image, and (b) erosion of Natural Image for a 2×2 structuring element.



(a) (b)

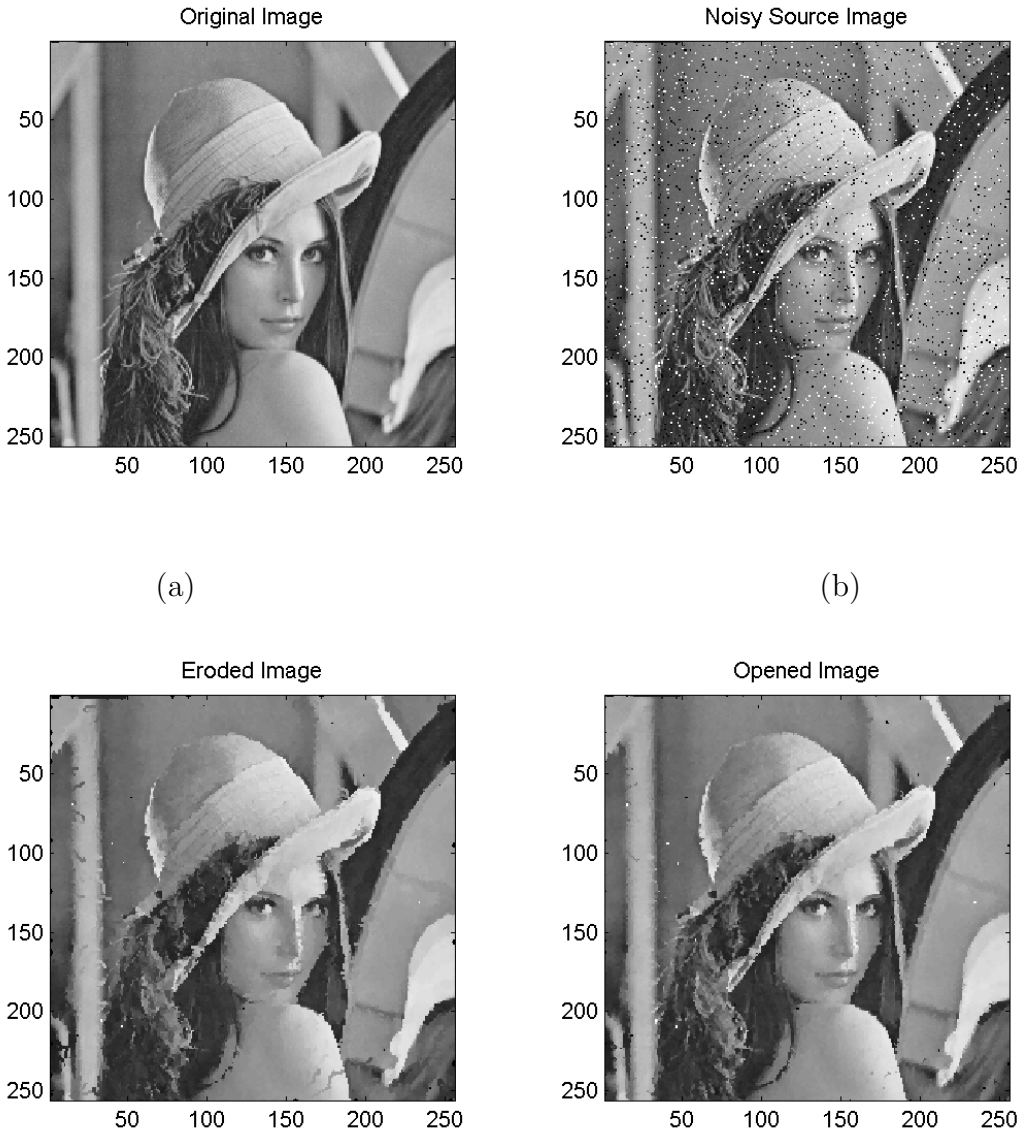
Figure 4.9: Results of BTV-based opening $\hat{\gamma}_B$, using adaptive SE. (a) Original Natural Image, and (b) opening of Natural Image for a 2×2 structuring element.



(a) (b)

(c) (d)

Figure 4.10: A Lena image with Salt&Pepper noise cleaned with an adaptive SE filter using filtering depth 1. Maximal size of structuring element here is 2×2. Notice that there are still part of the noise pixels. (a) Original Image (b) Image corrupted by the noise (c) erosion of Noisy Image (d) opening of Noisy Image



(a) (b)

(c) (d)

Figure 4.11: A Lena image with Salt&Pepper noise cleaned with an adaptive SE filter using filtering depth 3. Maximal size of structuring element here is 2×2. Notice that there are still part of the noise pixels. (a) Original Image (b) Image corrupted by the noise (c) erosion of Noisy Image (d) opening of Noisy Image

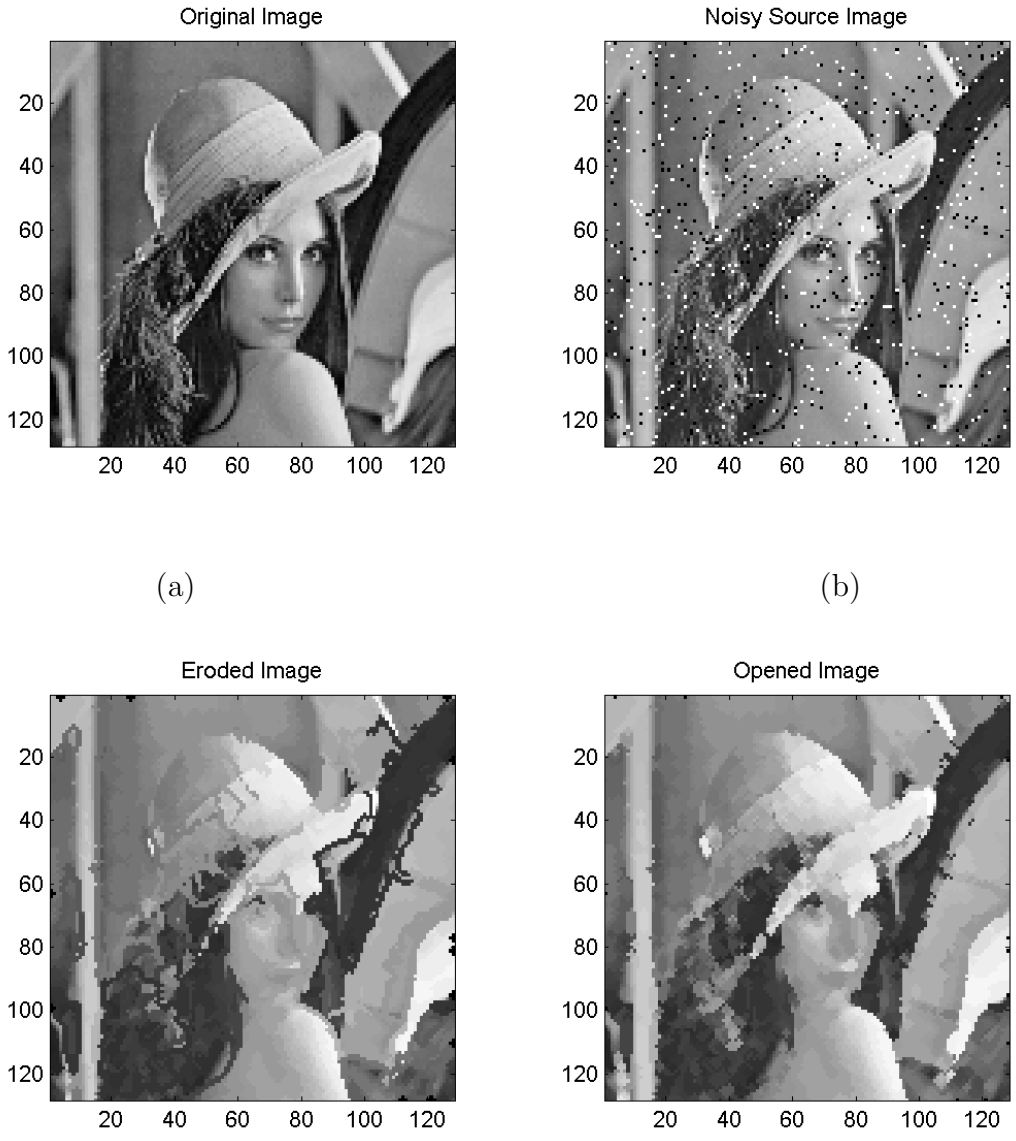
4.2 Filtering using multiple minimal paths

An alternative way to avoid the trenches is to store, for each flat zone, all possible combinations of the path-variations, and to choose the most suitable one during filtering. The suitable path-variation is one that has a maximal common-part length. For the example given in Table 2.1, we can store, for the skeleton point $x = 5$, both BTV transform possibilities $\{-2, 13, -1\}$ and $\{8, -3, 5\}$. Graphically, it can be seen in Fig. 4.6 that the vertex 5, $(x = 5)$, has 2 possible origins - one from the vertex 4 and another from vertex 6. For $x = 4$, the alternating sequence is $\{-2, 13\}$, thus its father is vertex 3. Therefore, the infimum path-variation of the two neighbors $x = 4$ and $x = 5$ is either $\{0\}$ or $\{-2, 13\}$, depending on which option we will choose. We choose the path-variation that has a maximal common part length - the option where vertex 4 is the father of vertex 5, that is, $\{-2, 13\}$. Of course, this process is more complex and more memory consuming.

Another possible implementation is to store the information about multiple possibilities for minimal paths, by enabling connections of the nodes to the multiple fathers in the TD-Tree. This saves the memory usage, but the price is computational overhead. In this case, the path-variation will be created a number of times for each pixel during the filtering, instead of to create it once for each flat zone - before the filtering. Our implementation uses the first option of storing multiple paths for each vertex.

A drawback of this approach is that it does not fit all cases, but only those where the topographic distance is equal in the different paths. Obviously, it ignores a large number of cases, because the topographic distance does not have to be equal for multiple paths. Therefore, some trenches remain after the filtering. See the example given in Fig. 4.12.

This method has another significant drawback - the performance issue. For real-world images, the number of combinations become very large, and the computation time and memory requirements are exponentially dependent on the number of flat zones in the image. Probably, it is possible, by using some heuristic methods, to reduce significantly the computation requirements, but this issue is out of the scope of this thesis.



(a) (b)

(c) (d)

Figure 4.12: A Lena image with Salt&Pepper noise cleaned with a multiple minimal paths filter. Size of structuring element here is 2×2. Notice that there are still some craters. (a) Original Image (b) Noisy Lena Image (c) erosion of Noisy Image (d) opening of Noisy Image

4.3 Filtering using a combined method

It is possible to combine the features of the above two methods into a single method. We can still look at all multiple path combinations and find such a combination that results in the largest adapted SE and a longest alternating sequence. An example is given in Fig. 4.13.

4.4 Results Comparison

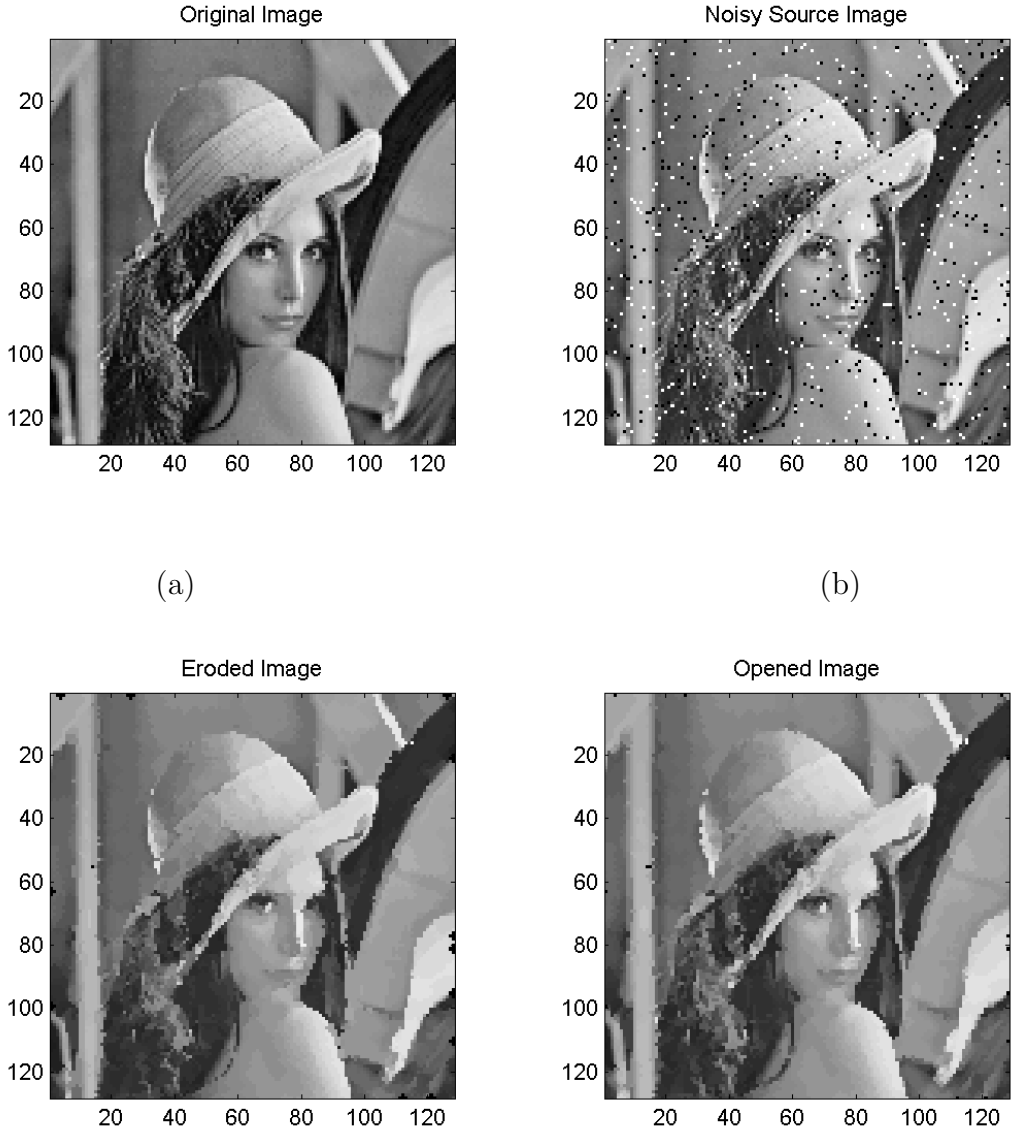
4.4.1 Comparison of different methods for avoiding trenches

In this section, we summarize the comparison between different methods for solving the trench problem. Figs. 4.14(c),(d) show the trench problem in BTV-based erosion $\hat{\varepsilon}_B$ and opening $\hat{\gamma}_B$ of the noisy image in Fig. 4.14(b), respectively. Figures 4.15, 4.16 and 4.17 show the results of the three proposed methods. The multiple minimal paths method is unacceptable, because it leaves trenches. The adaptive structure element method and combined method results are very similar. The combined method results are better than those of the adaptive structuring element method, but it is more complex and more memory consuming.

4.4.2 Filtering of AS images versus traditional morphological filtering

In this section, filtering results in the Alternating-Sequence inf-semilattice are compared to the traditional erosion, dilation, opening, closing, open-close, close-open operators in the complete lattice of gray level functions and a median filter.

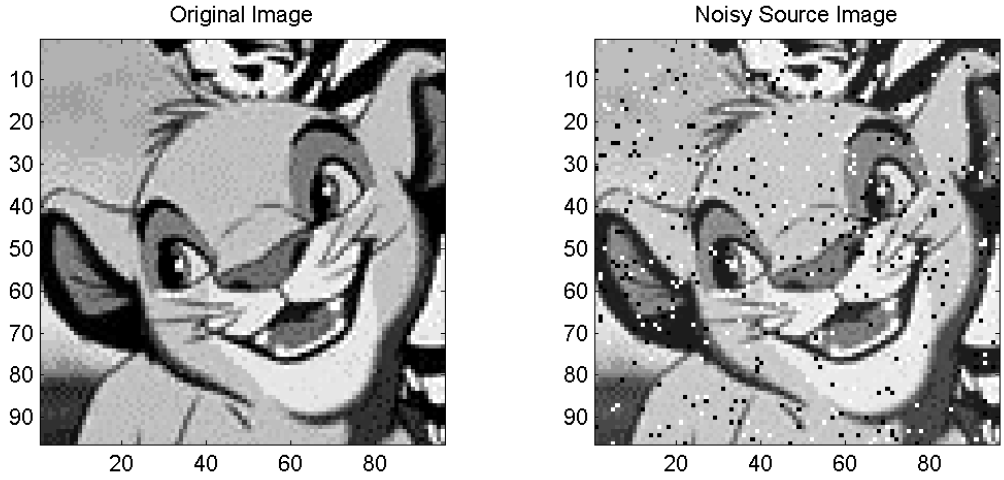
In the example given in Fig. 4.19, traditional closing removes dark noise pixels, but not the bright ones, and damages thin dark parts and edges. Traditional opening removes bright noise, but not dark noise. In contrast, opening that was defined in inf-semilattice of AS images is self-dual and removes both bright and dark noise. However, it is possible to perform pseudo self-dual filtering in complete lattice of gray level functions using traditional open-close or close-open operators, see Fig. 4.20. The drawback is that such filtering contains two consecutive operations. Opening removes bright noise, but damages thin bright parts or edges. Closing after opening



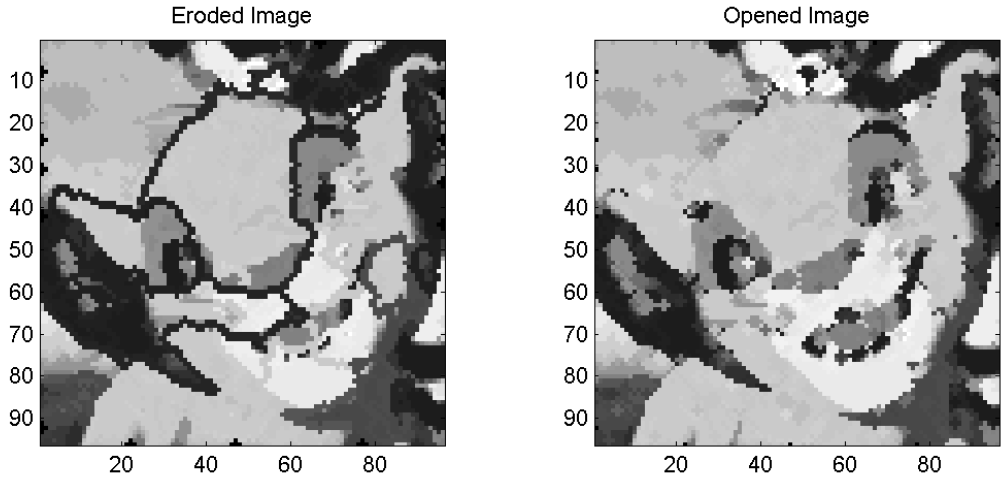
(a) (b)

(c) (d)

Figure 4.13: A Lena image with Salt&Pepper noise cleaned with a combined filter. Maximal size of structuring element here is 2×2. (a) Original Image (b) Noisy Lena Image (c) erosion of Noisy Image (d) opening of Noisy Image

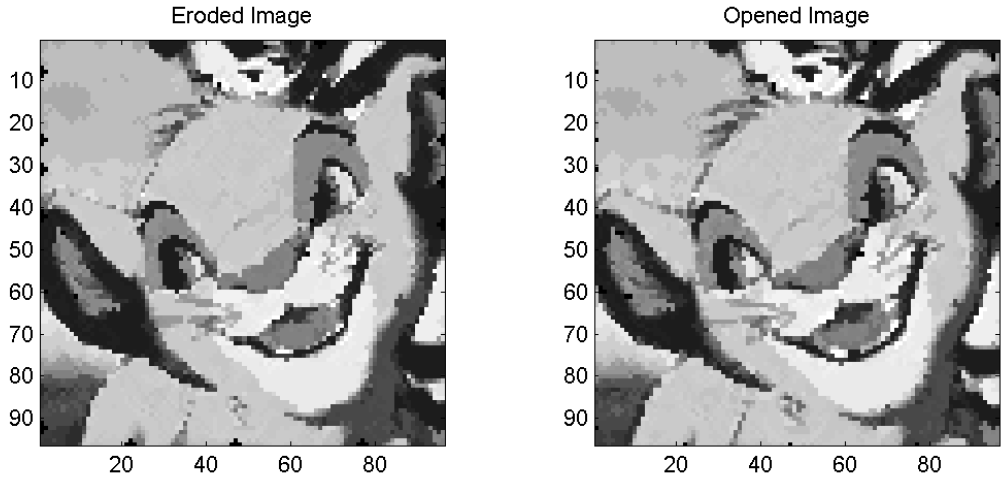


(a) (b)



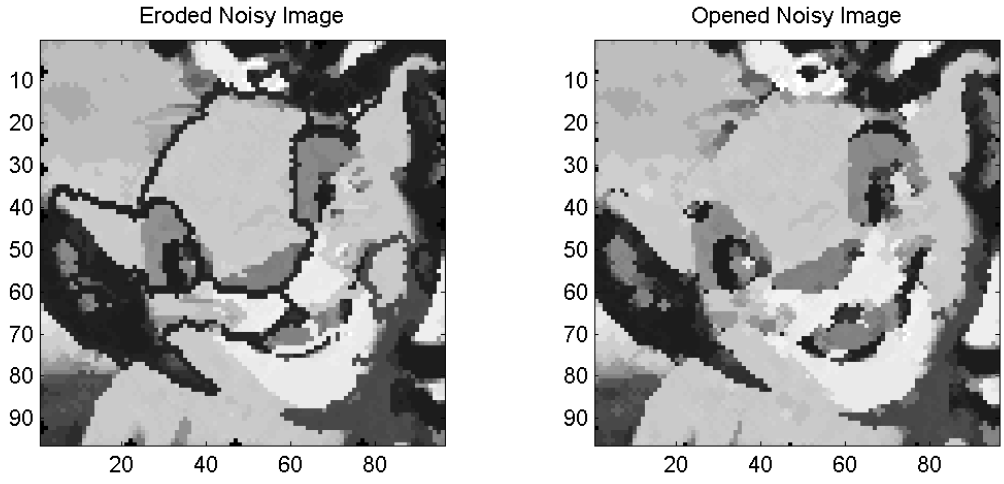
(c) (d)

Figure 4.14: Trench problem in results of BTV-based erosion $\hat{\varepsilon}_B$ and opening $\hat{\gamma}_B$. Maximal size of structuring element here is 2×2. (a) Original Image (b) Noisy Simba Image (c) erosion of Noisy Image (d) opening of Noisy Image



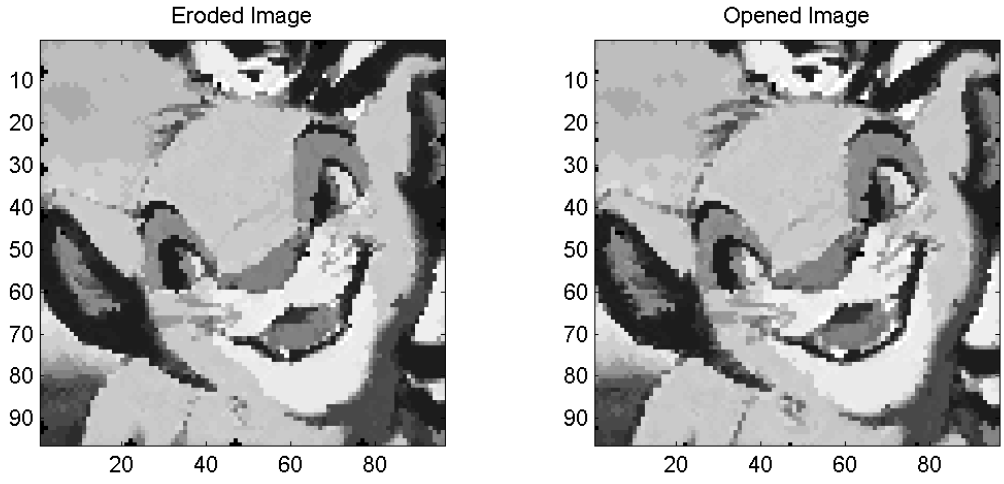
(a) (b)

Figure 4.15: A Simba image with Salt&Pepper noise cleaned with an adaptive SE filter. Maximal size of structuring element here is 2×2. (a) erosion of Noisy Image (b) opening of Noisy Image



(a) (b)

Figure 4.16: A Simba image with Salt&Pepper noise cleaned with an multiple minimal paths filter. Size of structuring element here is 2×2. Notice that there are still some craters. (a) erosion of Noisy Image (b) opening of Noisy Image



(a) (b)

Figure 4.17: A Simba image with Salt&Pepper noise cleaned with an combined filter. Maximal size of structuring element here is 2×2. (a) erosion of Noisy Image (b) opening of Noisy Image

removes dark noise, but cannot reconstruct the thin parts that where damaged by opening. Closing removes dark noise. Opening after closing removes bright noise, but cannot reconstruct the parts that where damaged by closing. Median filtering is a self-dual operator, see Fig. 4.21. Therefore, it is better than open-close and close-open operators. But, the major drawback of the median filter is that it is not idempotent, in contrast to opening defined in complete inf-semilattice of AS images.

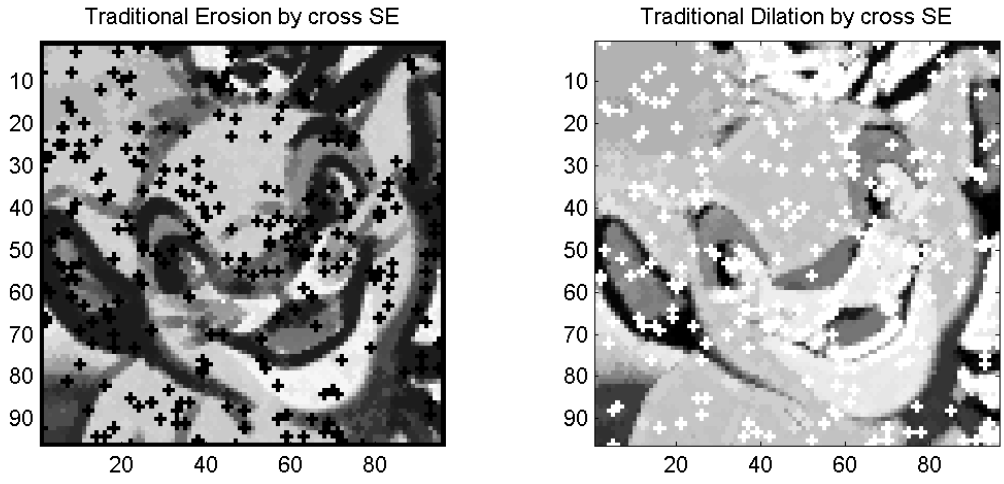


Figure 4.18: Traditional erosion and dilation of gray scale image

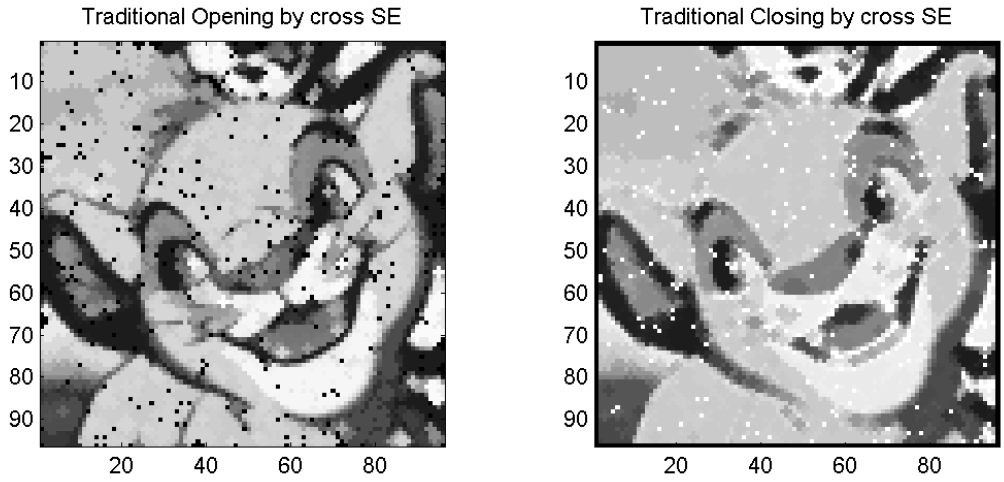


Figure 4.19: Traditional opening and closing of gray scale image.

Figure 4.20: Traditional pseudo-dual open-close and close-open operators.

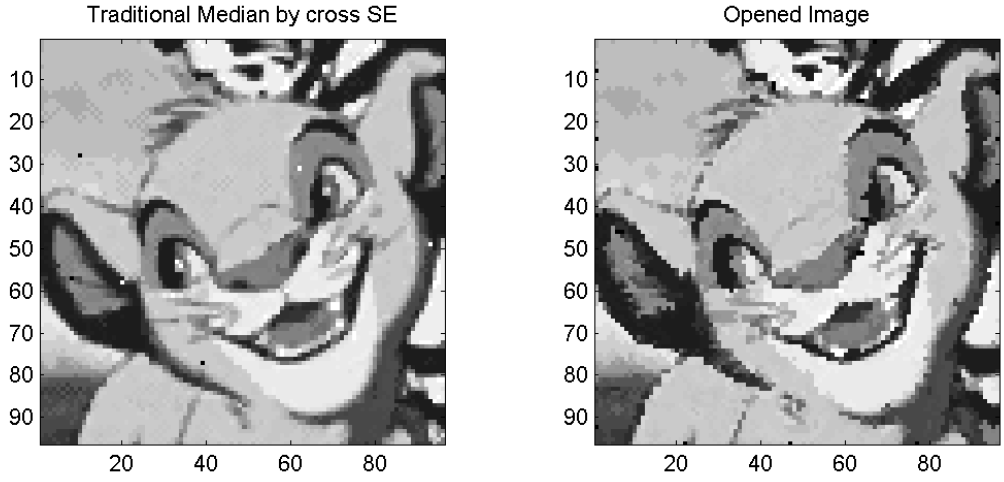


Figure 4.21: Median operator compared to the combined method of self-dual opening of AS image.

Chapter 5

Tree Semilattices

In this chapter, we propose a general framework for tree-based morphological image processing, which unifies the schemes presented in previous chapters, and enables the definition of new morphological operators that are based on tree representations. The heart of the proposed approach is a novel complete inf-semilattice of tree representations of images. The proposed image processing scheme is shown is figure 5.1.

This chapter is organized as follows. First, Section 5.1 introduces the complete inf-semilattice of tree representations. Then, in Section 5.2, morphological image processing using the complete inf-semilattice of tree representations is developed. In Section 5.3, we address the problem of whether the complete inf-semilattice of tree representations induces a complete inf-semilattice structure on the set of images as well. Finally, the chapter is summarized in Section 5.4.

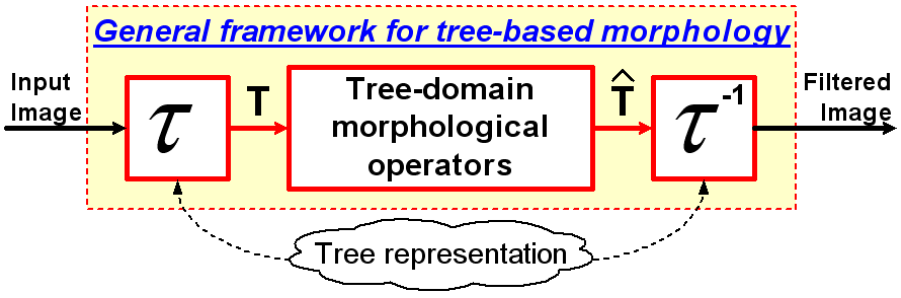


Figure 5.1: Tree-based morphology.

5.1 The Complete Inf-Semilattice of Tree Representations

Let L be an arbitrary set of "labels", and let $t = (V, E)$ be a rooted tree, with root r, such that $V \subseteq L$. Therefore t is a tree of labels. Moreover, let $M : \mathbb{E} \mapsto V$ be an image of vertices, mapping each point in $\mathbb{E}$ to a vertex of t. As before, $\mathbb{E}$ is either an Euclidean space or a discrete rectangular grid within the image area.

Definition 10. (Tree Representation) *The structure $T = (t, M)$ shall be called a* tree representation. *The set of all tree representations associated to the label set L and to the root r shall be denoted by $\mathcal{T}_r^L$.*

Figure 5.2 depicts an example of a tree representation.

Consider the following relation between tree representations: For all $T_1 = (t_1, M_1)$ and $T_2 = (t_2, M_2)$ in $\mathcal{T}_r^L$,

$$T_1 \leq T_2 \iff t_1 \subseteq t_2 \text{ and } M_1 \preceq_{t_2} M_2, \tag{5.1}$$

where $\subseteq$ is the usual graph inclusion, and $\preceq_{t_2}$ is the partial ordering of vertices (see

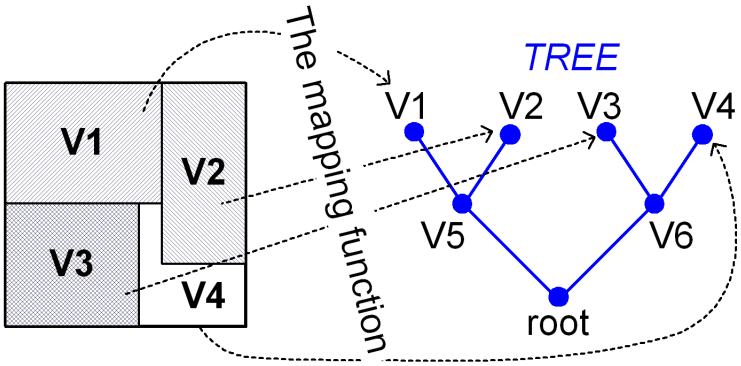


Figure 5.2: An example of image tree representation. An image with V1, V2, V3 and V4 zones is represented as a tree. Each pixel in this zone is mapped to a corresponding label.

section 2.1) within the tree t_2, taken point-wise, that is:

$$M_1 \preceq_{t_2} M_2 \iff \forall x \in \mathbb{E}, M_1(x) \preceq_{t_2} M_2(x). \tag{5.2}$$

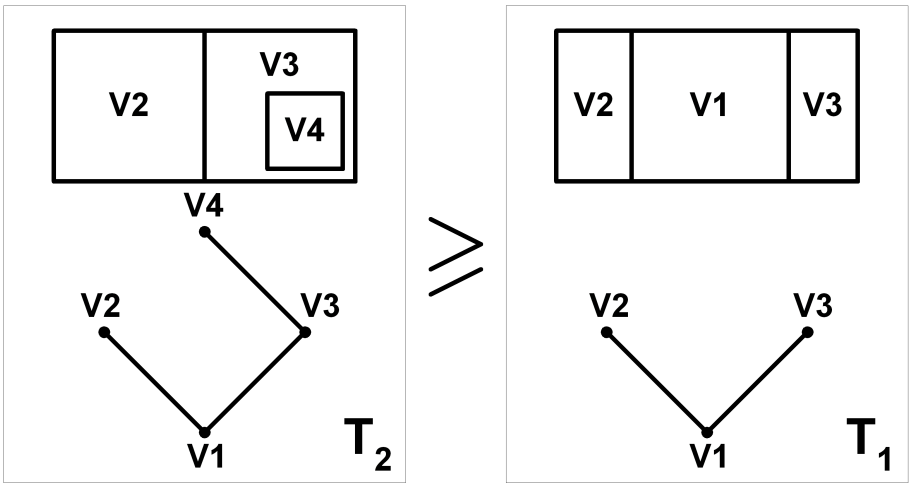
An example of order of trees is shown in Figure 5.3.

Proposition 1 (Tree order). *The above tree relation is a partial ordering on* $\mathcal{T}_r^L$.

Proof. Since both graph inclusion ($\subseteq$) and the intrinsic root tree partial ordering ($\preceq_{t_2}$) are partial orderings, so is the composed relation on the tree representation. □

A relevant question at this point is what are the infimum and supremum operators related to the above partial ordering and whether they are well defined for any subset of tree representations. In other words, is $(\mathcal{T}_r^L, \leq)$ a lattice or a semilattice, and if so, complete or not?

First notice that, for two graphs g_1 and g_2, the intersection $g_1 \cap g_2$ is the infimum graph. However, if g_1 and g_2 are trees, then unfortunately $g_1 \cap g_2$ is *not* necessarily a tree. This is because the resulting graph may be disconnected, in which case $g_1 \cap g_2$ is a forrest, but not a tree. For instance, consider the two trees $t_1 = (\{r, a, b, d\}, \{ra, ab, bd\})$ and $t_2 = (\{r, a, c, d\}, \{ra, ac, cd\})$ as shown in Figure 5.4.



(a) (b)

Figure 5.3: An example of order of trees. The tree representation (a) is bigger than (b), because of two reasons: tree (b) is included in tree (a), and there exists pixels in (b) that belong to label 1, while the same pixels belong to labels 2 and 3 in (a).

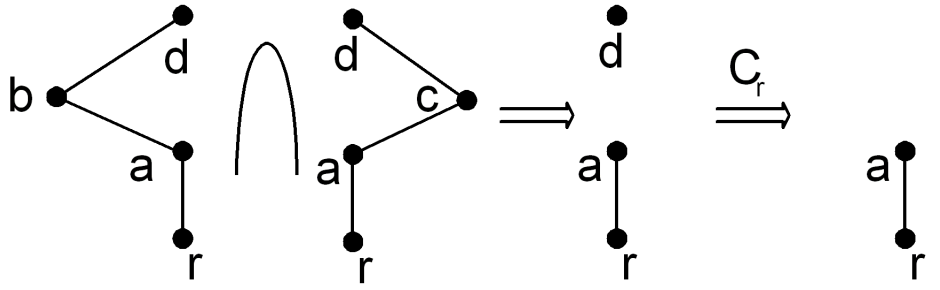


Figure 5.4: An example of trees intersection that is not a tree.

The intersection is given by $t_1 \cap t_2 = (\{r, a, d\}, \{ra\})$, which has two connected component: $(\{r, a\}, \{ra\})$ and $(\{d\}, \{\})$. Notice, however, that only one of these components contains the root r, and it is in fact the largest tree with root r that is included both in t_1 and in t_2. That is, the sub-tree containing r is the infimum between the two trees regarding the inclusion order.

This is true in general. That is, if we define $C_r(\cdot)$ to be the operator that extracts from a given graph its connected component containing the root r, then $C_r(t_1 \cap t_2)$ is the infimum $t_1 \wedge t_2$ between the trees.

Let us turn now to the infimum of mapping functions. It is natural to consider $M_1(x) \curlywedge M_2(x)$ as the natural candidate for the infimum of the two vertices $M_1(x)$ and $M_2(x)$. But, in which tree should this infimum be calculated? Should it be in t_1, t_2, $t_1 \wedge t_2$, $t_1 \cup t_2$? The infimum $t_1 \wedge t_2$ is correct, but it may occur that either $M_1(x)$ or $M_2(x)$ (or both) do not belong to its set of vertices $V(t_1 \wedge t_2)$. For instance, consider the trees t_1 and t_2 as in the example above, and assume that $M_1(x) = b$ and $M_2(x) = c$ as shown in Figure 5.5. Neither $M_1(x)$ nor $M_2(x)$ belongs to $V(t_1 \wedge t_2) = \{r, a\}$.

The solution to this problem is to first "project" each vertex $M_1(x)$ and $M_2(x)$ to $V(t_1 \wedge t_2)$, and then to take the infimum of the projected vertices on the tree $t_1 \wedge t_2$. By "projection" of a vertex v onto a subtree $t' \subseteq t$ we mean finding the vertex w in $V(t')$ that is the closest to v in the path connecting v to r inside t'. For instance, in the above example, the projection of both $M_1(x) = b$ and $M_2(x) = c$ onto $t_1 \wedge t_2$ is a. Recalling that vtr represents the path linking v to r in the tree t, the projection $P_{t'}(v)$ of v onto the subtree t' could be also defined as the leaf of $C_r(vtr \cap t')$.

In summary, we get the following proposition.

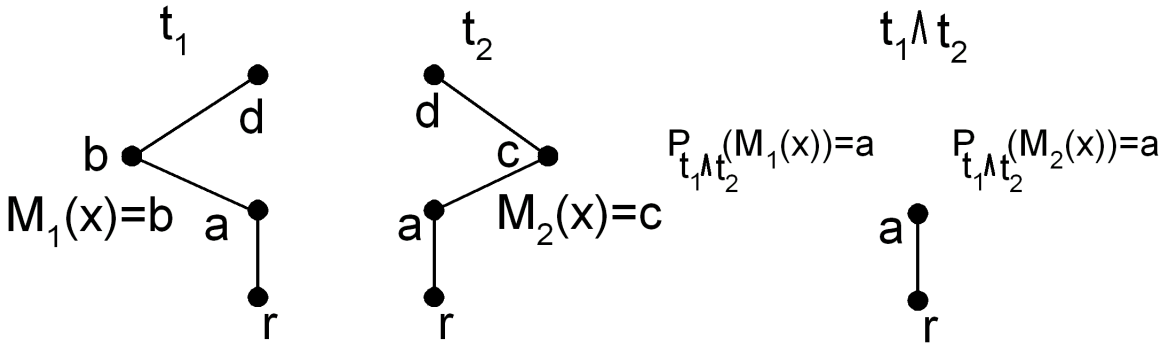


Figure 5.5: An example of a projection of the mapping function. Mapping functions $M_1(x)$ and $M_2(x)$ of point x are projected to a subtree $t_1 \wedge t_2$.

Proposition 2. *The tree representation infimum is given by:*

$$T_1 \wedge T_2 = (t_1 \wedge t_2, P_{t_1 \wedge t_2}(M_1) \curlywedge_{t_1 \wedge t_2} P_{t_1 \wedge t_2}(M_2)), \tag{5.3}$$

where $t_1 \wedge t_2 = C_r(t_1 \cap t_2)$, $C_r(\cdot)$ *stands for the connected graph containing* r, *and* $P_{t'}(\cdot)$ *means the projection onto* t', *i.e., the closest vertex in the path linking the operand to* r, *which belongs to* $V(t')$.

In words, the tree representation infimum is built in the following way: The intersection of the trees is calculated, and the sub-tree containing r is extracted. This is the infimum of the trees. For each point in $\mathbb{E}$, the mapping function is obtained by calculating the infimum vertex of the projections of the original mapping functions onto the infimum tree.

Proof. The infimum must have the following properties:
$T = T_1 \wedge T_2$ is infimum iff :

1. $T \preceq T_1$ and $T \preceq T_2$.

2. If there exists T_3 such that $T \preceq T_3$, $T_3 \preceq T_1$ and $T_3 \preceq T_2$, then $T_3 = T$.

The graph part of the tree transform is created by a regular set intersection and $C_r(\cdot)$ operator, thus $t \subseteq t_1$ and $t \subseteq t_2$. The mapping function of T is calculated from infimum of vertices in the tree $t_1 \wedge t_2$, so $M \preceq M_1$ and $M \preceq M_2$.

In addition we must show that the infimum is unique. $t_1 \wedge t_2$ is created by a set intersection that is a known infimum, and then by operator $C_r(\cdot)$ that is a univalent operator. Any other graph, that is a subset of $t_1 \cap t_2$ but is larger than $t_1 \wedge t_2$, could not be a tree, because it must contain a vertex not connected to the root vertex.

Another question is about the infimum mapping function M. It is obtained in 2 steps: a projections of source functions to $t_1 \wedge t_2$ and then infimum between them. Let's assume that another infimum mapping function exists M_3, such that at least one mapping - $M(x) \preceq M_3(x)$ and $M(x) \neq M_3(x)$. For this to be true, $M_3(x)$ must be a member of $t_1 \wedge t_2$ and smaller in tree sense then M_1 and M_2, but we recall that M is calculated from the tree infimum, therefore such M_3 could not exist.

□

Now, let us find out what the supremum of the tree representations is. First, notice that the union $t_1 \cup t_2$ of two trees t_1 and t_2 is always connected, but one can not assure that it does not contain loops. Therefore, the union is not necessarily a tree. Worse, there does not necessarily exist a smallest tree that is larger than the union. For instance, if $t_1 = (\{r, a, c\}, \{ra, ac\})$ and $t_2 = (\{r, b, c\}, \{rb, bc\})$, then $t_1 \cup t_2$ is equal to the graph $(\{r, a, b, c\}, \{ra, ac, rb, bc\})$, which is not a tree, and there does not exist a tree that contains it. Therefore, there does not exist a supremum of tree representations if the union of their trees is not a tree. Now, suppose that the union is a tree; let us focus on the mapping function. Now both $M_1(x)$ and $M_2(x)$ do belong to $V(t_1 \cup t_2)$, but their supremum in $t_1 \cup t_2$ may not exist. In summary, we obtain the following.

Proposition 3. *The supremum $T_1 \vee T_2$ of two tree representations $T_1 = (t_1, M_1)$ and $T_2 = (t_2, M_2)$ is given by:*

$$T_1 \vee T_2 = \begin{cases} (t_1 \cup t_2, M_1 \curlyvee_{t_1 \cup t_2} M_2), & \text{if } t_1 \cup t_2 \text{ is a tree, and} \\ & M_1 \curlyvee_{t_1 \cup t_2} M_2 \text{ exists,} \\ \not\exists, & \text{otherwise.} \end{cases} \tag{5.4}$$

In conclusion, the set of tree representations together with the partial ordering defined in (5.1) is a complete inf-semilattice. The least element is $T_0 \stackrel{\triangle}{=} ((\{r\}, \{\}), M_0(x) \equiv r))$.

As indicated by the following proposition, calculations become much simpler when all tree presentations involved in an infimum or supremum operation have a common

tree associated to them.

Proposition 4. *Let $T_1 = (t, M_1)$ and $T_2 = (t, M_2)$. In this case,*

$$T_1 \wedge T_2 = (t, M_1 \curlywedge_t M_2)\,, \tag{5.5}$$

and

$$T_1 \vee T_2 = (t, M_1 \curlyvee_t M_2)\,, \tag{5.6}$$

where $M_1 \curlyvee_t M_2$ may not exist.

Proof. The general infimum definition in equation 5.3 can be easily reduced to a simpler form if $t_1 = t_2$. Then $t_1 \wedge t_2$ becomes just t and the projection operator $P_{t_1 \wedge t_2}$ has no effect because the trees are equal. For that reason the equation may be rewritten as above.

In the similar way, the general supremum definition at equation 5.4 is reduced to the above form, because $t \cup t = t$.

□

The above is in fact the situation that occurs when one defines flat erosion and dilations on the complete inf-semilattice of tree representations. The flat erosion is the operator ε defined as:

$$\varepsilon(T) = \bigwedge_{b \in B} T_{-b}, \tag{5.7}$$

where B is a structuring element, and T_{-b} is the tree representation obtained by translating the mapping function of T by the vector $-b$. That is, if $T = (t, M)$, then $T_{-b} = (t, M_{-b})$. It is easy to verify that the above operator is an erosion on $\mathcal{T}_r^L$. Indeed, it is distributive with respect to the tree representation infimum.

Using Proposition 4, one obtains that

$$\varepsilon(T) = (t, \curlywedge_t \{M_{-b} | b \in B\})\,. \tag{5.8}$$

According to the morphological semilattice theory [21], on a complete inf-semilattice, one can associate to any given erosion an opening γ and any morphological operator that is derived from compositions of erosions and openings, such as the internal gradient, dark top-hat transform, and skeletons. Furthermore, the adjoint dilation δ

exists, and, even though it is not well defined for all complete inf-semilattice elements, it is always well-defined for elements that are mapped by the erosion, and $\gamma = \delta\varepsilon$.

In the case of the above tree-representation flat erosion, the adjoint dilation is given by:

$$\delta(T) = (t, \curlyvee_t \{M_b | b \in B\}) . \tag{5.9}$$

5.2 Image Processing on Tree Semilattices

5.2.1 Proposed Approach

Our goal is to process a given grayscale image f, which we assume to be an integer-valued function on $\mathbb{E}$, i.e., $f \in \mathrm{Fun}(\mathbb{E}, \mathbb{Z})$. We propose the following approach for processing f, using the complete inf-semilattice of tree representations.

1. Transform f into a pair (T, ℓ), where $T = (t, M) \in \mathcal{T}_r^L$ is a tree representation, and $\ell : L \mapsto \mathbb{Z}$ is a function that maps labels into graylevels,

2. Perform one or more morphological operations on T to obtain a processed tree representation $\hat{T} = (t, \hat{M})$.

3. Transform $(\hat{T}, \ell)$ back into a new image $\hat{f} \in \mathrm{Fun}(\mathbb{E}, \mathbb{Z})$, using:

$$\hat{f}(x) = \ell\left(\hat{M}(x)\right) . \tag{5.10}$$

Notice that the format for the inverse transform (item 3) is fixed, and does not depend on the exact definition of the tree transform τ. The user however has a great deal of freedom for choosing τ itself. A number of tree transforms are discussed in sections 2.2 and in section 6.1. Because there are many possible alternatives for defining τ, designing the forward transform is the main issue for applied research in that field, where the goal is to find the best tree representation of an image in some sense. In this chapter, however, this issue is not addressed; instead, a tree transform is assumed given, and we focus on the corresponding morphological operators and the methodology to obtain them.

If the morphological operation in item 2 above is the erosion ε, then, according to next theorem, all three steps can be collapsed into one equation.

Proposition 5.

$$\hat{f}(x) = \ell\left(\curlywedge_t \{M_{-b}(x) | b \in B\}\right). \tag{5.11}$$

Let the elements of the label set L be indices to the flat zones of f; each flat zone is represented by a label in L, so that $V = L$. Let r be the label to one of the flat zones, which is set in advance as a root for a spanning tree of the RAG. Now assume that τ maps an image f into a structure (T, ℓ), where $T = (t, M)$ is a tree representation, with $t = (L, E)$ being a spanning tree of the RAG.

Proposition 6. *Let $R(v)$ be the flat zone associated to a node v of the spanning tree t, i.e.,*

$$R(v) = \{x \in \mathbb{E} | M(x) = v\}. \tag{5.12}$$

Now consider the eroded tree representation $\hat{T} = \varepsilon_B(T) = (t, \hat{M})$, by the structuring element B, and let $\hat{R}(v)$ be the flat zone associated to v in the eroded tree representation, i.e.,

$$\hat{R}(v) = \{x \in \mathbb{E} | \hat{M}(x) = v\}. \tag{5.13}$$

Then, the following holds:

$$\bigcup_{v' \succeq v} \hat{R}(v') = \left(\bigcup_{v' \succeq v} R(v')\right) \ominus B. \tag{5.14}$$

Proof. First let's pay attention to a feature of infimum of the mapping function. If one mapping function $M_1(x)$ belongs to a certain subtree of v : t_v and another $M_2(x)$ doesn't belong to that subtree, the infimum of $M_1(x)$ and $M_2(x)$ will not be belong to that subtree t_v. If both $M_1(x)$ and $M_2(x)$ are belong to same subtree, the infimum will belong to the same subtree.

Let's assume that the above equation 5.14 is wrong, and reach a contradiction. The above equation may be wrong either if there exist $x \in \mathbb{E}$ that belongs to the left side but not to the right side, or the opposite.

In the first case, if x belongs to $\bigcup_{v' \succeq v} \hat{R}(v')$, it means that $\hat{M}(x)$ belongs to a subtree of v. Where $\hat{M}$ was created from $\curlywedge_t \{M_{-b} | b \in B\}$. It means that all source

mappings belong to that subtree. But x lies outside a shape left by an erosion operation on $\bigcup_{v' \geq v} R(v')$ using the structure element B. It means that when B was applied to x, it included $M_{-B}(x)$ vertices, and one of that vertices was outside the subtree of v. So, $\hat{M}(x)$ could not belong to $\bigcup_{v' \geq v} \hat{R}(v')$.

In the second case, x belongs to $\bigcup_{v' \geq v} R(v')$ but doesn't belong to $\bigcup_{v' \geq v} \hat{R}(v')$. In a similar way as described above, all source $M_{-B}(x)$ vertices belong to $\bigcup_{v' \geq v} R(v')$, so that $\hat{M}(x)$ must belong to the same subtree - $\bigcup_{v' \geq v} \hat{R}(v')$.

□

The above proposition suggests that one way of implementing the tree-representation erosion (assume the list of all flat zones of f and a spanning tree t are given) is by:

1. Associating to each item v of the list the set of points $\bar{R}(v) \stackrel{\triangle}{=} \bigcup_{v' \geq v} R(v')$, i.e., the union of all flat zones that are equal or greater than v in the tree t.
2. Eroding $\bar{R}(v)$ for each v in the list.
3. Assigning the gray level of v for all pixels in $\hat{R}(v) \stackrel{\triangle}{=} \bar{R}(v) \setminus \bigcup_{v' \geq v} \bar{R}(v')$.

5.2.2 Examples and Particular Cases

RAG's and Spanning Trees

In order for the tree transform to be invertible, τ should be such that it assigns a common label to each flat zone of f. This is because τ^{-1} maps each label to a single gray value. This suggests that special attention should be paid to the flat zones of f.

One way of addressing the flat zones of a given image is by considering its Regional Adjacency Graph (RAG). The RAG is a graph, where V is the set of all flat zones of the image, and E contains all pairs of flat zones that are adjacent to each other.

A spanning tree is a subgraph of a RAG that should, obviously, be a tree, and have the same vertex set V as the RAG. A spanning tree creates a hierarchy in the RAG, defining father/son relationships between adjacent flat zones.

BTV and Shape Trees

The BTV tree described in chapter 3, is built from the RAG using minimal topographic distance criteria for building the tree t. The vertex set of the BTV tree is the

same vertex set of the RAG - a set of all flat zones in the image. In that sense, the BTV tree is a particular case of a spanning tree.

The shape tree is defined in [20] and the resulting semilattice is defined in [17]. The shape tree differs from the BTV tree in one main aspect: its vertex set is not the flat zones of the image. Its vertex set is built from the flat zones during the tree generation. Each father vertex area includes all sons in addition to an area of itself. For each point in the image there may be more than one vertex in the shape tree that includes that point. In order to preserve the single value of the mapping function we define a mapping to the biggest vertex of all possible vertices in the tree.

Max and Min Trees

Another group of trees is Max and Min trees. Those trees, as described in section 2.2.1, are created from the RAG, sorted by the gray level of the flat zones. Similar to the above shape tree, each vertex region in the picture contains all children vertices. In Max Tree the father gray level is always bigger than the son. And, of course, the opposite in a Min tree. When a father is always brighter/darker then a son, the infimum operation always changes the gray level to the darker/brighter side. This has the same effect as a regular erosion on a gray-level picture. For this reason those trees are of no special interest for the proposed approach.

5.3 Semilattice of Images

5.3.1 Structure Induction

What we would really like is the complete inf-semilattice of tree representation (using a tree τ) to induce a complete inf-semilattice in the image domain. That is, we would like, for instance, that the composite operation of $\tau^{-1}\varepsilon\tau$ be an erosion in the image domain. However, that is not guaranteed; for some tree transforms τ an image semilattice is obtained and for others not. In fact, the partial ordering in the tree-representation domain does induce a partial ordering for images, for any τ; however, the infimum is not guaranteed to exist.

Let f be an image, $\tau(f)$ is a tree representation. $T_\varepsilon = \varepsilon(\tau(f))$ is the tree representation of an erosion operator result. Let $f_\varepsilon = \tau^{-1}(\varepsilon(\tau(f)))$ be the image obtained

by the inverse tree transform. There is no guarantee that $\tau(f_\varepsilon)$ will be equal to T_ε. In fact, T_ε often lies outside the range of all possible tree representations of images, as shown in Fig. 5.6. Therefore, we cannot assure that $\tau(f_\varepsilon) < \tau(f)$. In words, we cannot be sure that operators in a tree semilattice preserve their features in the image domain.

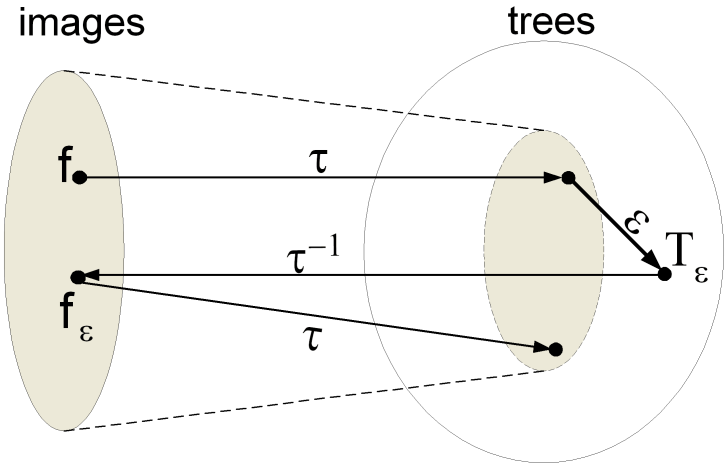


Figure 5.6: Problem of existence of image semilattice, based on tree representation

The derivation of necessary and sufficient conditions for the tree transform to preserve the semilattice feature is not in the scope of the current thesis. As stated in the list of future directions in Chapter 7, we propose the investigation of this topic as open future research

5.4 Summary and discussion

This chapter introduced a general framework for image processing using tree representation. For this purpose, a semilattice of tree representations was developed. Morphological operators, such as erosion, opening and opening by reconstruction were defined in this semilattice. This general framework allows introduction of new, mostly self-dual, filtering techniques. One example of such technique is presented in the next chapter.

However, there are still issues waiting to be studied in this subject. The most important of them is the existence of an image semilattice based on a given tree

representation. The current definition of tree transform and operator definition does not assure the existence of the image semilattice (see section 5.3).

Chapter 6

Extrema-Watershed Tree example

Chapter 5 presents a general framework for producing morphological operators that are compatible to a given tree representations. For every tree representation, a complete inf-semilattice in the tree-representation domain is derived, and a set of morphological operators on that inf-semilattice is obtained.

As for any general framework, the strength and usefulness of the proposed morphological tree-based framework is measured by its ability to:
1) unify existing methods as much as possible, in a simple way, and
2) generate new and useful methods.

The purpose if this chapter is to present an example of a new method that is obtained from the general framework, and to investigate its usefulness. This exemplifies the strength of the general framework, as a tool for generating new, useful sets of morphological operators.

Based on the general framework of Chapter 5, all that is needed in order to obtain a new set of morphological operators is a given tree representation. The more this tree representation is useful, the more useful these morphological operators are likely to be, since many of the properties of the tree are inherited by the morphological operators (like self-duality).

The tree representation selected for the method presented in this chapter is a particular case of a "Binary Partitioning Tree", which is a state-of-the-art, general framework for tree generation, developed by Salembier in [10]. The proposed representation is built using an iterative merging process as presented by Salembier, Garrido and Garcia in [23].

The selected tree-representation was designed with the purpose of achieving good properties for tasks such as denoising and segmentation. The resulting tree-representation is called Extrema-Watershed Tree (EWT), and it yields a new set of morphological operators, derived from the Extrema Watershed Tree, by the general framework of Chapter 5. Fig. 6.1 shows how the given tree representation is used in general framework.

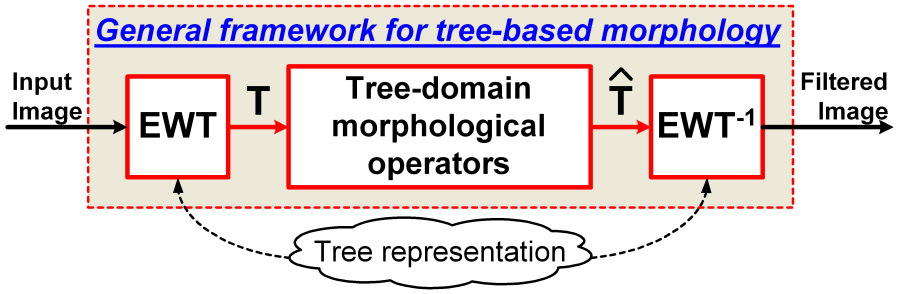


Figure 6.1: EWT-based morphology.

This representation is self-dual: dark and light pixels are treated evenly. The tree representation is built by symmetrical flooding of extrema zones to merge flat zones. In the tree building process, first the smallest extremum points are merged with their neighbors. This approach represents an image in such a way that smaller and brighter or darker objects are located further from a root of the tree, while larger objects are located closer to the root. This tree structure enables meaningful morphological operators to be defined, as described in section 6.2. Later, in section 6.3, the results are compared to a similar tree - the shape tree.

6.1 Extrema watershed tree description

Assume all extrema in the images, i.e., all regions associated to a local minimum or a local maximum. An algorithm for creating the watershed tree sorts all regions of extremum by size, and if sizes are equal, by gray level differences between the extremum and a closest adjacent region. Let's denote by V_{extr} a label corresponding to an extremum region. Then, e will be an edge in a G, the RAG (Region Adjacency Graph) that contains V_{extr}. For simplicity, we assume that V_{extr} is always the first vertex in e ($e = (V_{extr}, V_2)$). Each vertex V has a number of properties: $Lbl(V)$ is

its label; $GL(V)$ is the gray level of region corresponding to V; $Father(V)$ is the father vertex of V in the tree; and $Size(V)$ is the number of pixels in V. An exact comparison function is shown in Algorithm 2 below.

In every algorithm cycle, the smallest edge is removed from the edges queue and merged to create a new flat zone. In the merge process, a newly created flat zone receives a gray level of an edge vertex other than extremum and is derived by a region that is a union of both components. Then, all neighbor edges are extracted from the queue and new edges with a new vertex are added only if they still include extremum vertices. A pseudo code of the algorithm we developed is listed in Algorithm 3 below.

Using notions of **merging order**, **merging criterion** and **region model** introduced in [23], the proposed representation has a **merging order** according to a queue sorted by an order function shown in algorithm 2. Its **merging criterion** is whether the edge in question is the first in the queue of all edges. And the **region model** as described in the algorithm, is to take a gray level of the vertex other than the extremum and use a union of their regions.

This algorithm resembles an algorithm introduced by Vachier and Vincent in [24]. One difference is in the merging order: our main merge criteria is the size of an extremum, which is not the case in Vachier and Vincent's algorithm. Another difference is that the resulting tree is used here to define morphological operators, whereas Vachier and Vincent use it as an intermediate step for computing symmetrical dynamics (see [24] for details).

Let's take a 1-dimensional example to illustrate the tree structure. Figure 6.2 shows the first 3 steps in creating the extrema watershed tree. A source image includes 2 flat zones with extremum gray levels - v3 and v5. The first step in the algorithm is to merge v3 and v2, because v3 is a small extremum (only 1 point) and because it's gray level is closer to v2 then v5 with v6. The merge creates a new flat zone - v7 with gray level 9. A gray level is taken from a flat zone other than the extremum. The region of a new flat zone is a merge of the two source regions. The newly created v7 is a new extremum in the image. In the next step another extremum - v5 is merged with v6 to create v8, because v5 is the smallest extremum in the image. In the third step v7 is merged with v4 to create v9. At that stage all remaining flat zones are extremum. The next merge steps are to merge v8 and v9 with v10 and finally v10 is merged with v1 a root. The choice between merging of v8 with v9 or v1 with v9 is

arbitrary, because v1 and v8 have identical sizes and same gray level difference.

The resulting tree is depicted in Fig. 6.3.

Algorithm 2 Compare 2 extremum regions for a merge candidate selection

function CompareExtrema(e1,e2) ▷ e1 = (V^1_{extr}, V^1_2), e2 = (V^2_{extr}, V^2_2)
▷ return value 0 means e1 is smaller,value 1 means e2 is smaller
  if $Size(V^1_{extr}) < Size(V^2_{extr})$ **then**
    return 0
  else if $Size(V^2_{extr}) < Size(V^1_{extr})$ **then**
    return 1
  else ▷ sizes are equal
    $\Delta^1_{GL} = |GL(V^1_{extr}) - GL(V^1_2)|$
    $\Delta^2_{GL} = |GL(V^2_{extr}) - GL(V^2_2)|$
    if $\Delta^1_{GL} < \Delta^2_{GL}$ **then**
      return 0
    else
      return 1
    end if
  end if
end function

6.2 Morphological operations on the extrema-watershed tree

As described in the Chapter 5, once a tree transform is defined, morphological operations (such as erosion and opening) in the tree-domain can be obtained. In this section, we investigate the results of applying this technique to the extrema-watershed tree.

6.2.1 Erosion and opening

Consider the erosion and opening operators derived from the extrema-watershed tree, using the general framework developed in Chapter 5. In the following examples the two operators: erosion and opening are applied on synthetic and natural noisy and not noisy images.

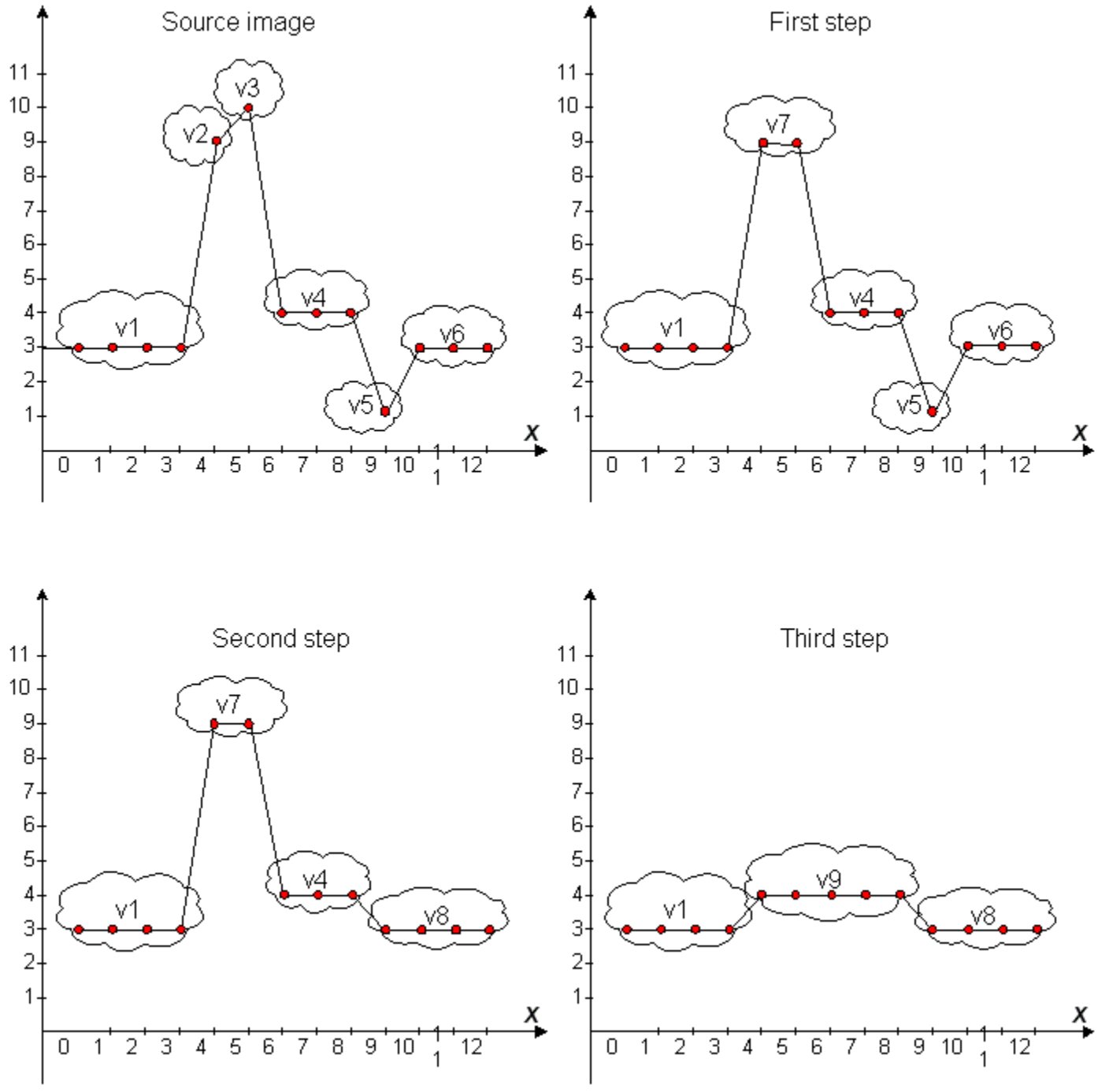


Figure 6.2: Example of extrema watershed tree creation.

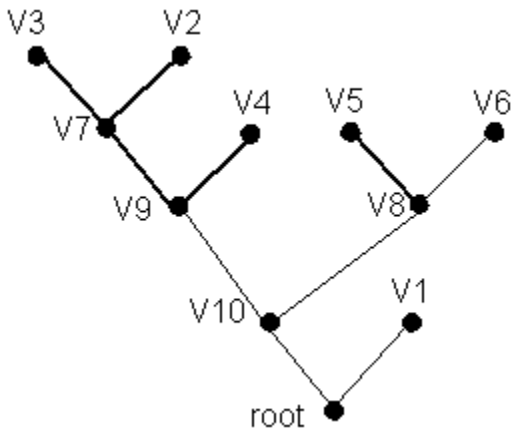


Figure 6.3: The Extrema Watershed Tree associated to the example in Fig. 6.2.

Algorithm 3 Create maxima watershed tree

A list of symbols :

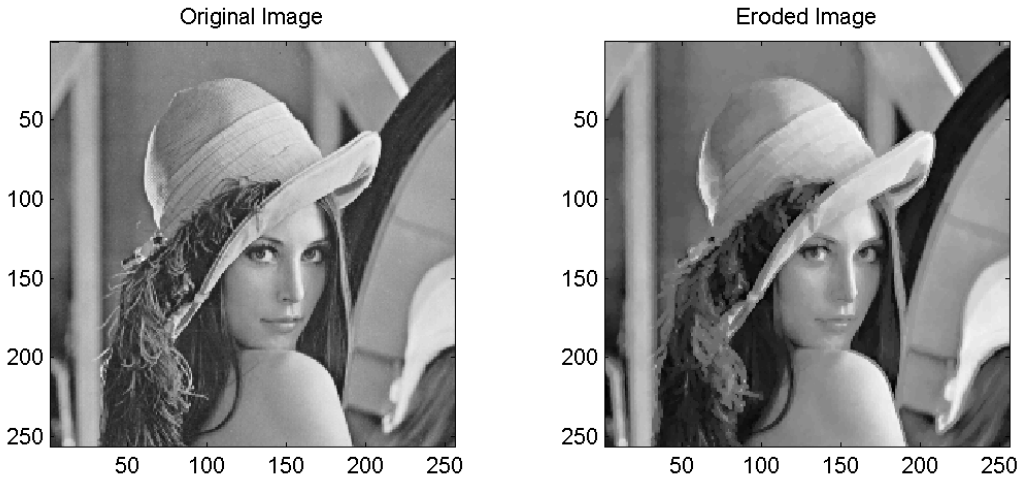
Eq A sorted queue of edges in G containing extremum region.

Ngh A sorted queue of edges, a subset of Eq, neighbors of V_{extr}, V_2.

Lbl The current label number, initialized to zero.

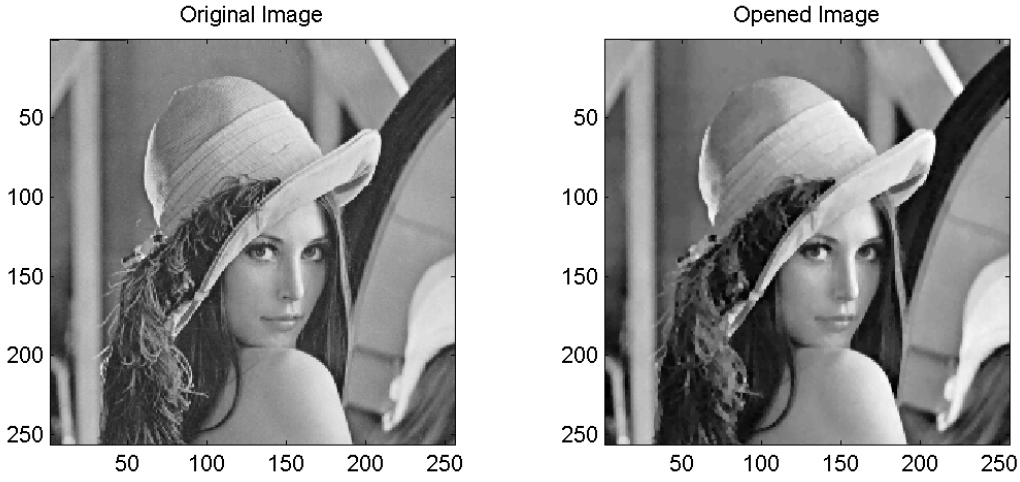
1: Find all extremum region ,V_{extr}, in G.
2: Find all edges connected to the extremum regions and put them in Eq.
3: Sort Eq according to the comparison function in alg. 2.
4: **while** Eq not empty **do**
5: $(V_{extr}, V_2) \leftarrow First(Eq)$
6: $Lbl = Lbl + 1$
7: Create new vertex V
8: $Lbl(V) \leftarrow Lbl$
9: $GL(V) \leftarrow GL(V_2)$ ▷ The new vertex always gets GL other then extremum
10: $Size(V) \leftarrow Size(V_{extr}) + Size(V_2)$
11: $Father(V_{extr}) \leftarrow V$; $Father(V_2) \leftarrow V$
12: Ngh ← all pairs in Eq containing V_{extr} or V_2
13: Substitute V_{extr} and V_2 by V in Ngh.
14: Remove all pairs in Ngh where the first vertex is not extremum anymore.
15: Sort Ngh according to the comparison function in alg. 2.
16: Merge Eq and Ngh preserving the order.
17: **end while**
18: $Root \leftarrow Lbl$ ▷ the last added label is defined as the root

The result of the erosion and the opening operations on the image Lena are shown in Figs. 6.4 and 6.5, respectively. As one can see, these operators create no significant artifacts (unlike the the trenches that are added by boundary topographic operators, as shown in Chapter 4). Very small components were removed, whereas the larger ones were preserved (by opening) or shrank (by erosion). Moreover, the average gray level of the picture did not change; in particular, the picture did not become darker, which is what usually happens after a standard erosion or opening. In addition, in Fig. 6.7 the filtering effect on a corrupted Lena image is shown. The image is corrupted with salt and pepper noise constituting of random black and white pixels. It is seen that after the filtering there is no sign of noise, and the resulted image is very similar to a filtered image without the noise. A comparison between the two results (filtered with and without the noise) is shown in Fig. 6.8. The image shown is the gray level difference between the filtered original and filtered noisy images, both obtained by opening and erosion. An additional example is shown in Figs. 6.11 and 6.10, with respect to the synthetic image "Simba". The Fig. 6.12 results of filtering a binary image, first introduced in Fig. 1.1, are presented. All the noise, except for noise on the edges, has been removed.



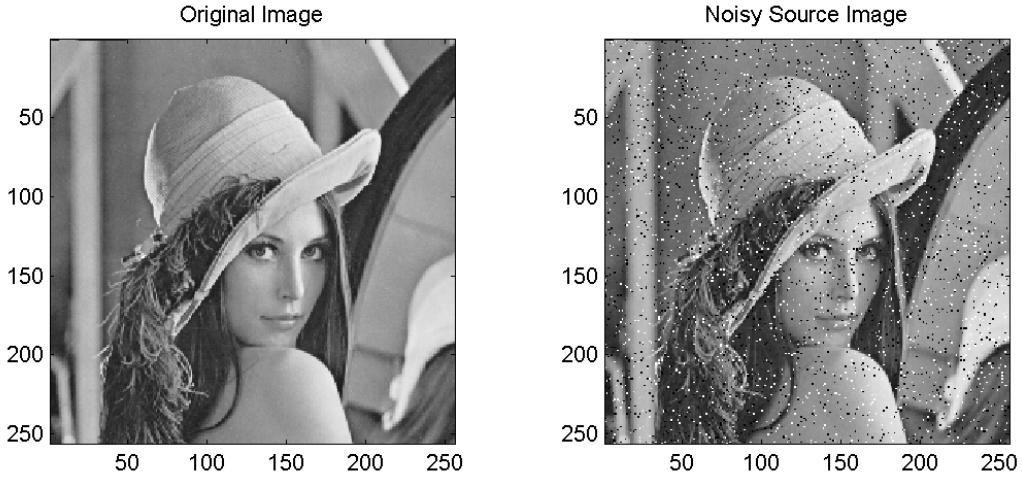
(a) (b)

Figure 6.4: Inverse transform of Extrema watershed tree of Lena image eroded by structuring element 2×2. (a) Original Image (b) Eroded Image



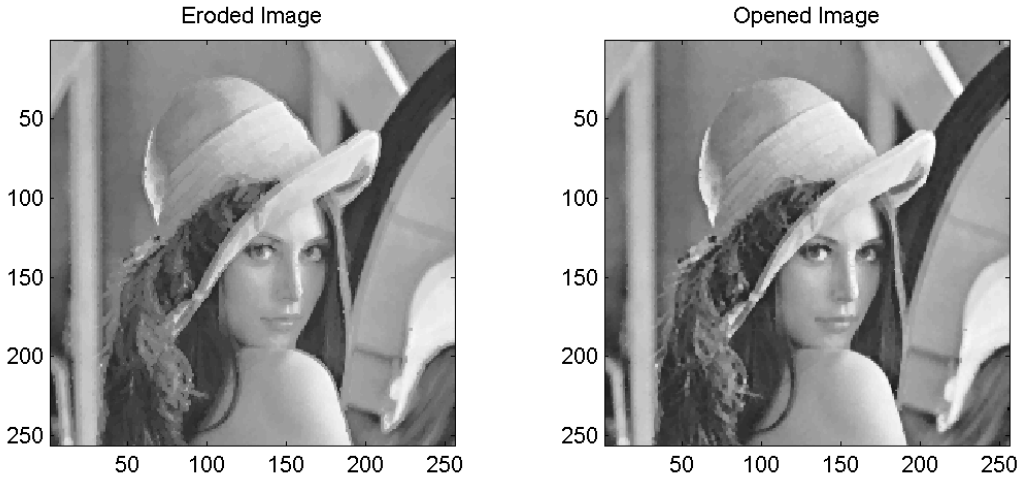
(a) (b)

Figure 6.5: Inverse transform of Extrema watershed tree of Lena image opened by structuring element 2×2. (a) Original Image (b) Opened Image



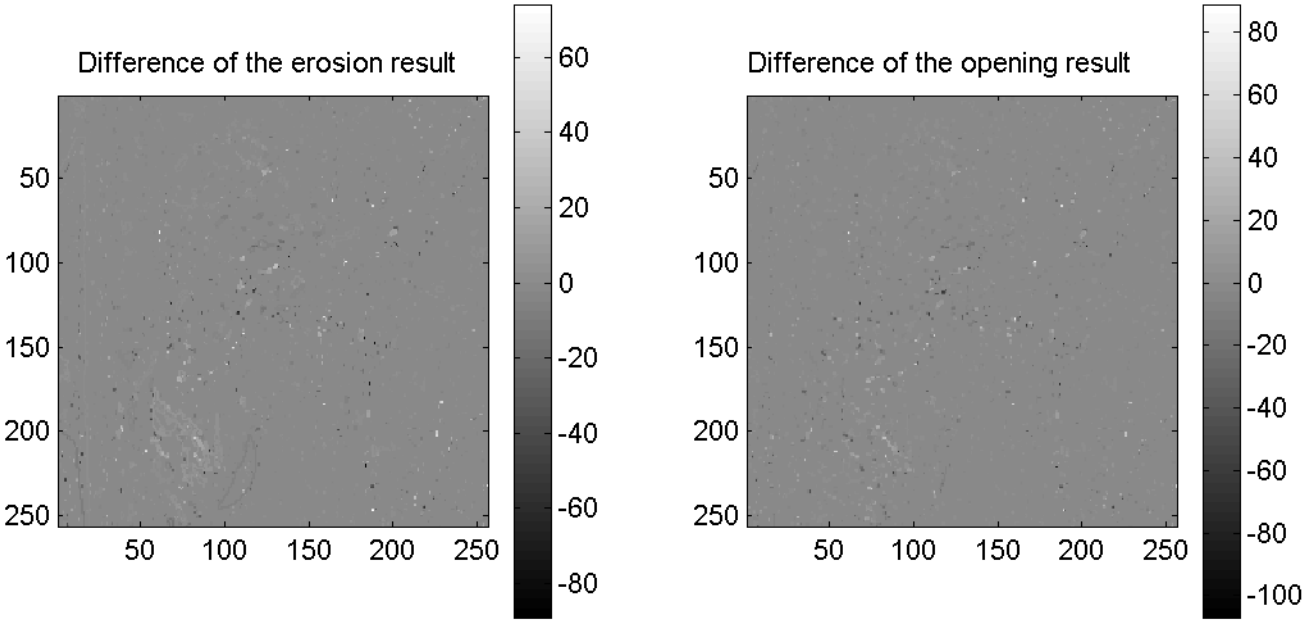
(a) (b)

Figure 6.6: A Lena image corrupted by the Salt&Pepper noise. (a) Original Image (b) Noisy Lena Image



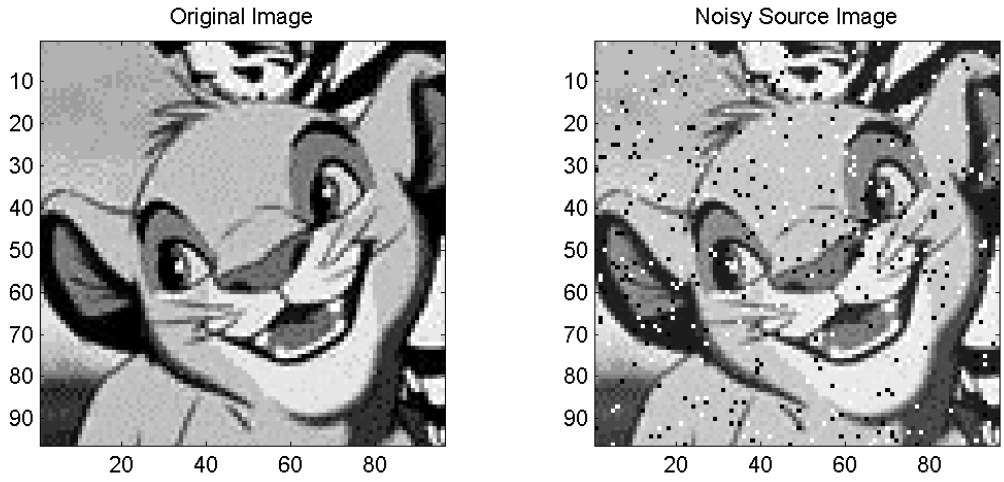
(a) (b)

Figure 6.7: Filtering Noisy Lena Image using t Extrema watershed tree. (a) Inverse transform of Extrema watershed tree eroded by structuring element 2×2 (b) Inverse transform of Extrema watershed tree opened by structuring element 2×2



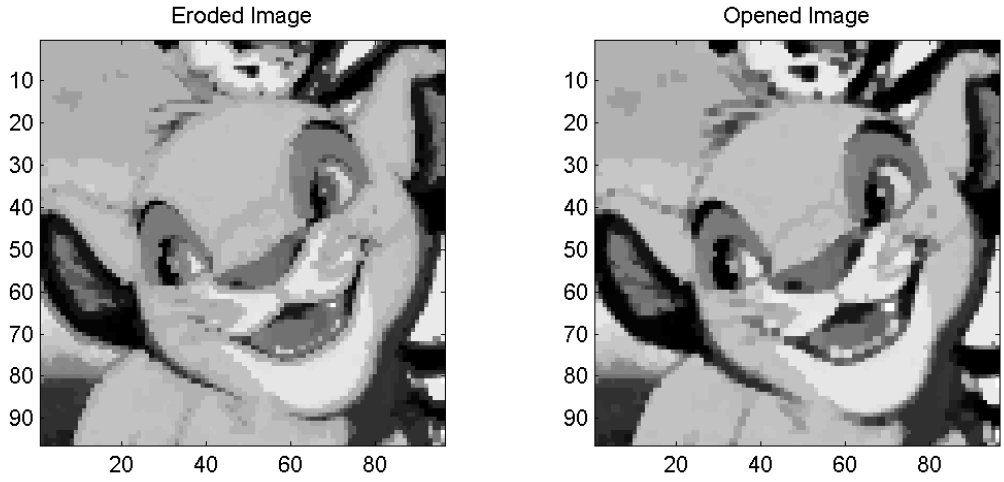
(a) (b)

Figure 6.8: A comparison between filtering results of noisy and clean image. (a) Erosion result difference. (b) Opening result difference.



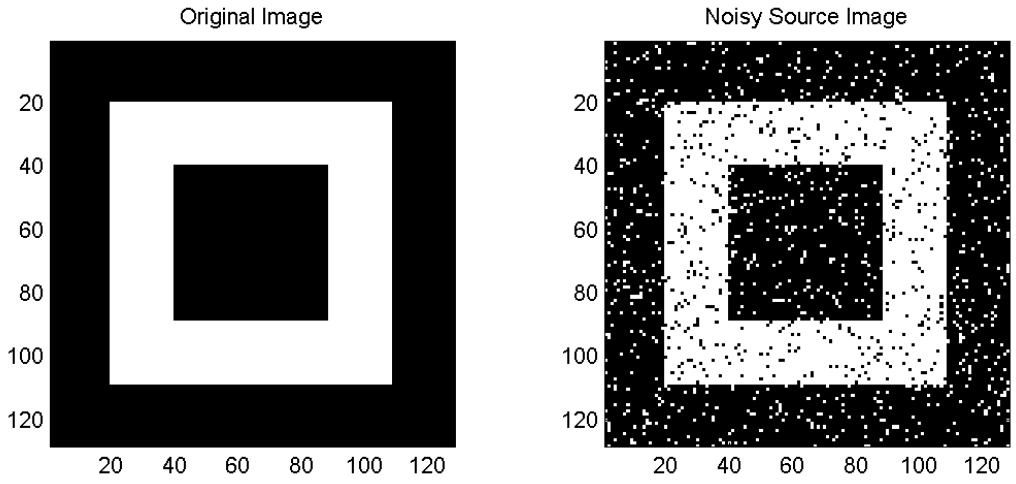
(a) (b)

Figure 6.9: A Simba image corrupted by the Salt&Pepper noise. (a) Original Image (b) Noisy Simba Image



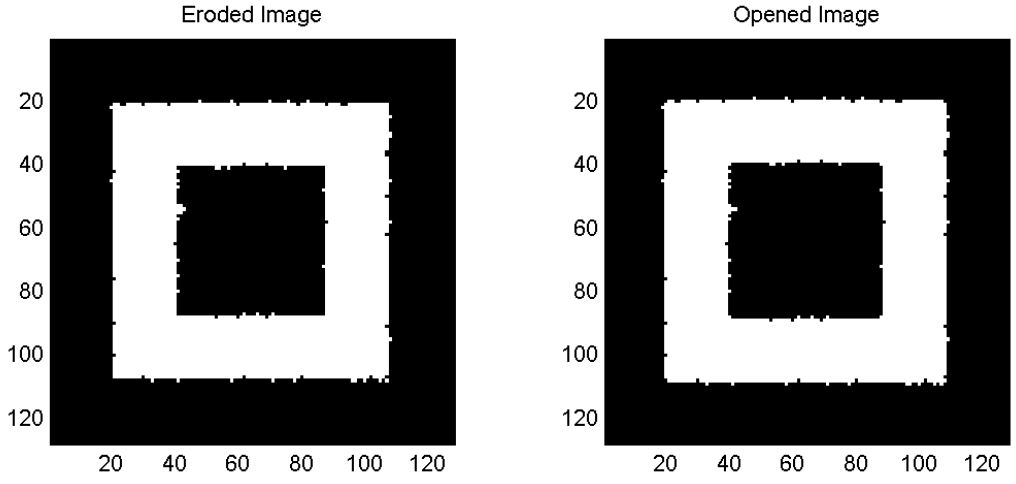
(a) (b)

Figure 6.10: Filtering Noisy Simba Image using Extrema watershed tree. (a) Inverse transform of Extrema watershed tree eroded by structuring element 2×2 (b) Inverse transform of Extrema watershed tree opened by structuring element 2×2



(a) (b)

Figure 6.11: A Binary image corrupted by the Salt&Pepper noise. (a) Original Image (b) Noisy Binary Image



(a) (b)

Figure 6.12: Filtering Noisy Binary Image using the Extrema watershed tree. (a) Inverse transform of the Extrema watershed tree eroded by cross structuring element (b) Inverse transform of Extrema watershed tree opened by cross structuring element.

6.2.2 Opening by reconstruction

We can define an *opening by reconstruction* operator in the extrema-watershed tree. The classical version of this morphology operator is defined in [3] as a single erosion operation followed by a sequence of conditional dilation operations. The sequence is stopped when stability is reached - meaning that a dilation followed by the minimum operation does not change the resulting image.

Using an opening operator defined on a tree, an *opening by reconstruction* operator is equivalent to tree pruning, where the pruning criteria is whether the vertex in question exists in the opened image. The tree pruning for *opening by reconstruction* is done using the following steps:

1. Apply an opening operator on the source image.

2. In a tree, for every label remaining in the tree, mark all vertices of the path starting from this label until the tree root, as still existing in the tree.

3. Replace every label in the input image by the closest parent label that was marked as still existing in the tree.

Let's demonstrate the method using an example illustrated in Fig. 6.13. In this example a source image consists of three rectangles located on a background. In the corresponding tree, those shapes have indexes V1, V2 and V3. In the example, a regular opening operator and an opening by reconstruction are applied on the image. The structuring element is a square that is bigger than V2, but smaller than the other shapes. The regular opening, shown in the figure, will leave the root label V5 shown in the image instead of V2. However, the opening by reconstruction will remove V2, as shown, and then reconstruct label V4 instead.

Additional examples can be seen on Figs. 6.14 and 6.15.

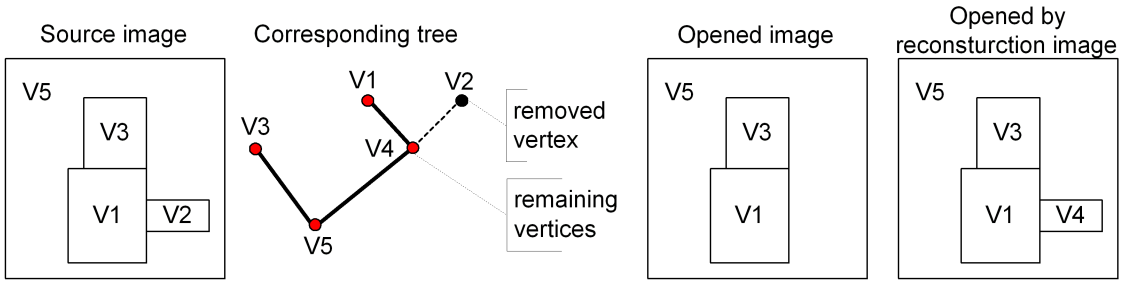
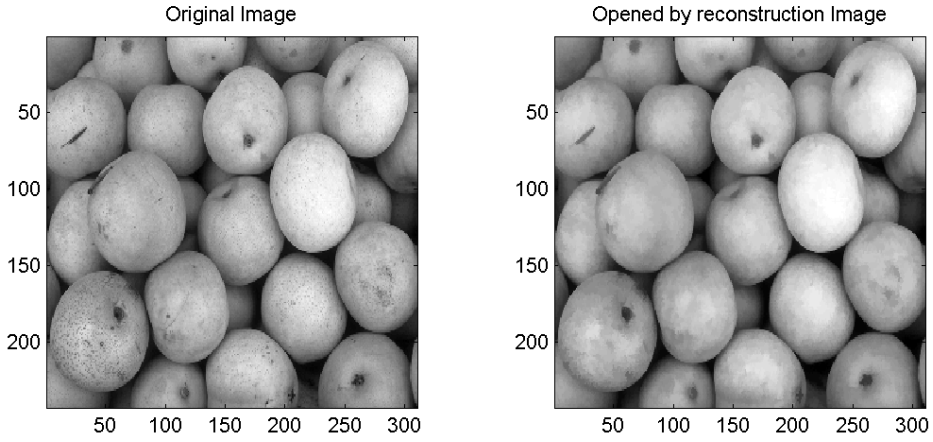


Figure 6.13: Opening by reconstruction of simple image.



(a) (b)

Figure 6.14: (a) Original pears image. (b) Opened by reconstruction image using SE of 3x3.

(a) (b)

Figure 6.15: (a) Original Lena image. (b) Opened by reconstruction image using SE of 3x3.

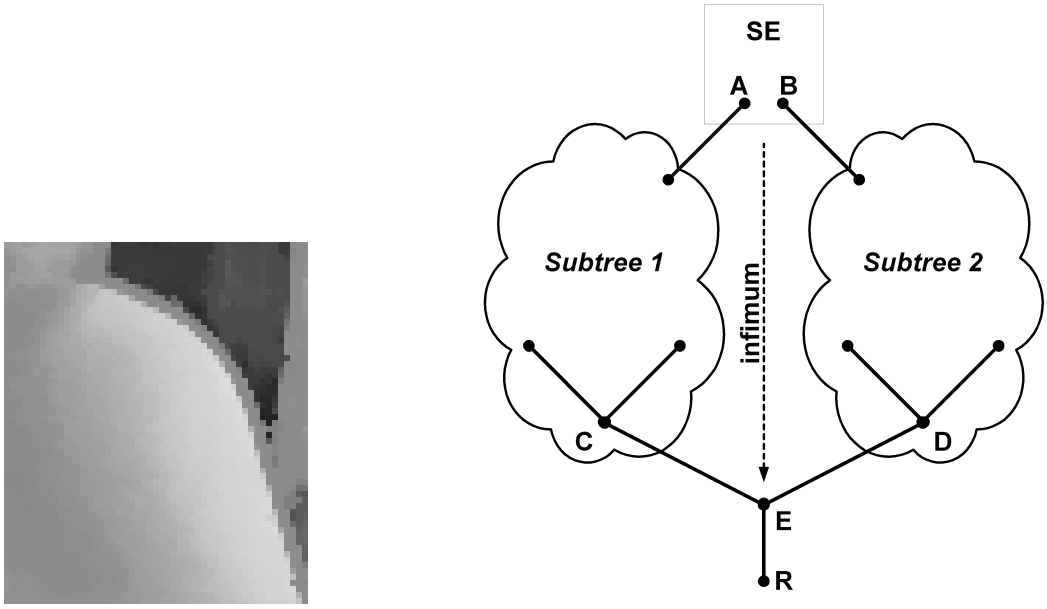
6.3 Study of EWT properties

What are the properties of extrema watershed tree and why use it? The EWT has two main properties, listed below:

1. This tree representation is self-dual.
2. The Extrema watershed tree provides implicit hierarchical segmentation due to the following reasons:
   - A tree is created in watershed-like process.
   - Small area extrema are leaves.
   - Bigger flat zones are close to the root.
   - Vertices connected in the tree usually have similar gray levels.

6.3.1 Implicit segmentation

Recalling that the extrema-watershed tree is built in a process that is very similar to flooding, it is natural to investigate the potential of using the resulting tree for



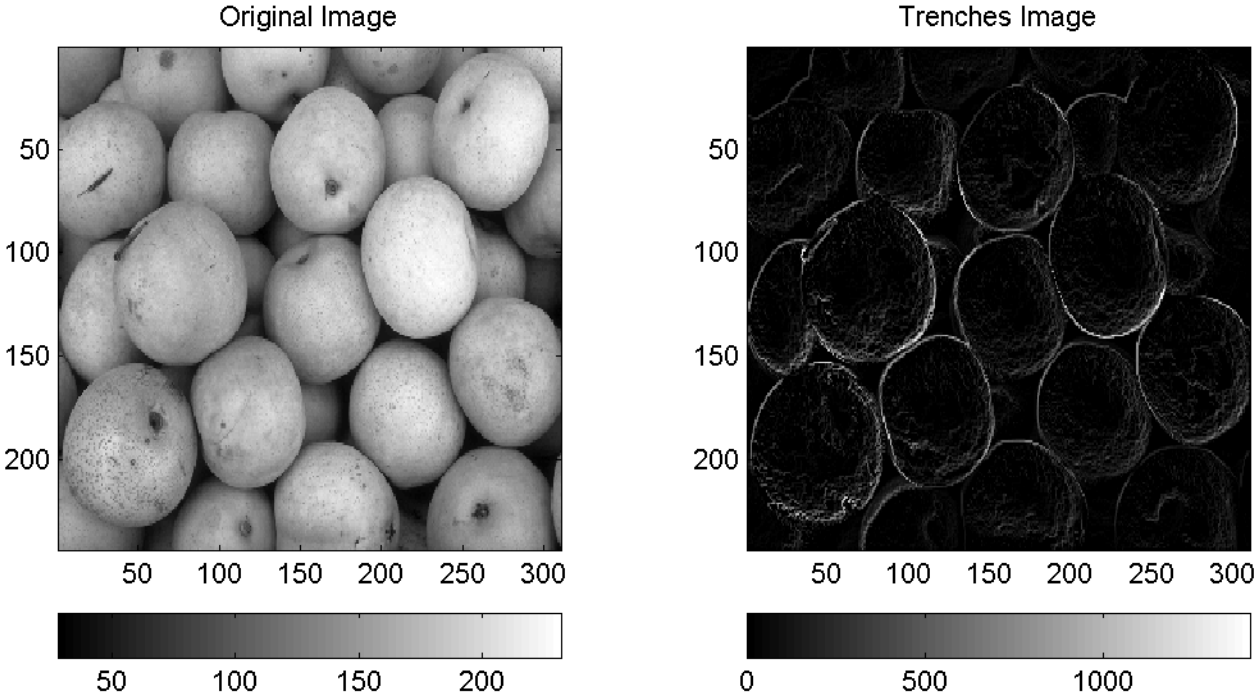
(a) (b)

Figure 6.16: Trenches meaning. (a) Trench example (b) An infimum of A and B, is a common father E. Its sons, C and D are the roots of subtree1 and subtree2. Pruning the tree to C and D vertices, will create the required segmentation.

segmentation. The idea behind the presented method is to prune the tree, leaving only the main objects. The tree is self-dual, and so is the resulting segmentation.

The implicit segmentation is based on the "trench" phenomenon. This phenomenon is described in Chapter 4. The trench is created, during erosion, on the border of two very different zones. The difference is defined by the path-distance to the common parent. The trench opens a gap with the gray level of the common parent. The sub-trees (the descendents of the common parent) are image segments. We are going to use this feature to find basic components of an image as shown in the example in Fig. 6.16(b). As one can see, common parent of two different subtrees, has two sons, where each of them is the root of the corresponding subtree. An example of a trench that opens between two zones in the Lena image is shown in Fig. 6.16(a). If we take only those son labels, and prune every other label beneath, a segmented image is created.

The tree-depth image, is the map of depths in the tree associated to each label in the image. In order to obtain trenches, another definition is required. The trench



(a) (b)

Figure 6.17: An example of image a trench image. (a) Original pears image. (b) Trenches map image.

image is the difference between images of depth of the original and the corresponding eroded image. The number of segments is controlled by setting a threshold for this trench image at an arbitrary value. In this way, we determine, what is a trench and what is a regular point. An example of trench image is shown in Fig. 6.17.

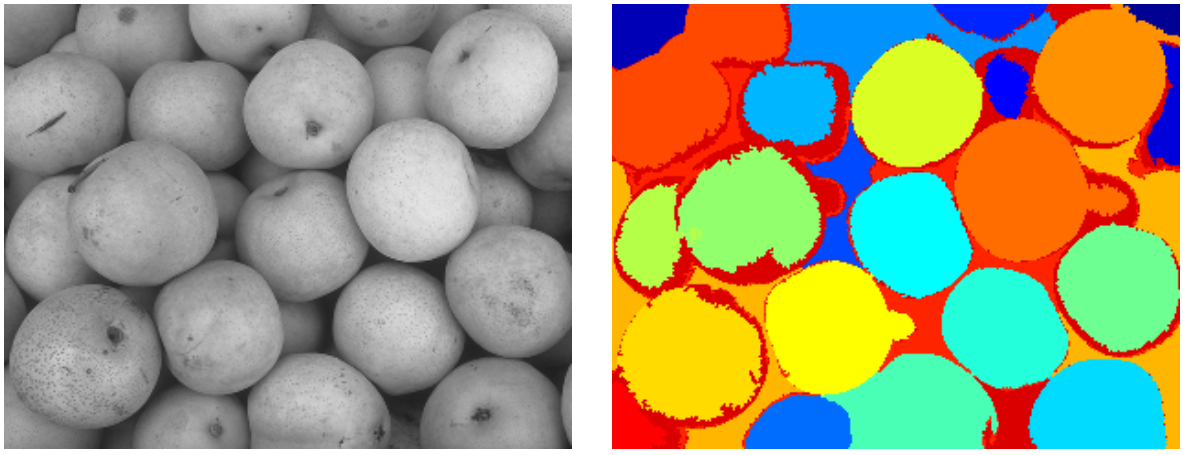
We can get the sub-tree labels using the following process :

1. Calculate the image erosion.

2. Extract the trenches from the depth-difference image.

3. Prune the tree, leaving only sons of vertices exposed in trenches.

4. Perform additional pruning by size and similarity according to the parent criterion.



(a) (b)

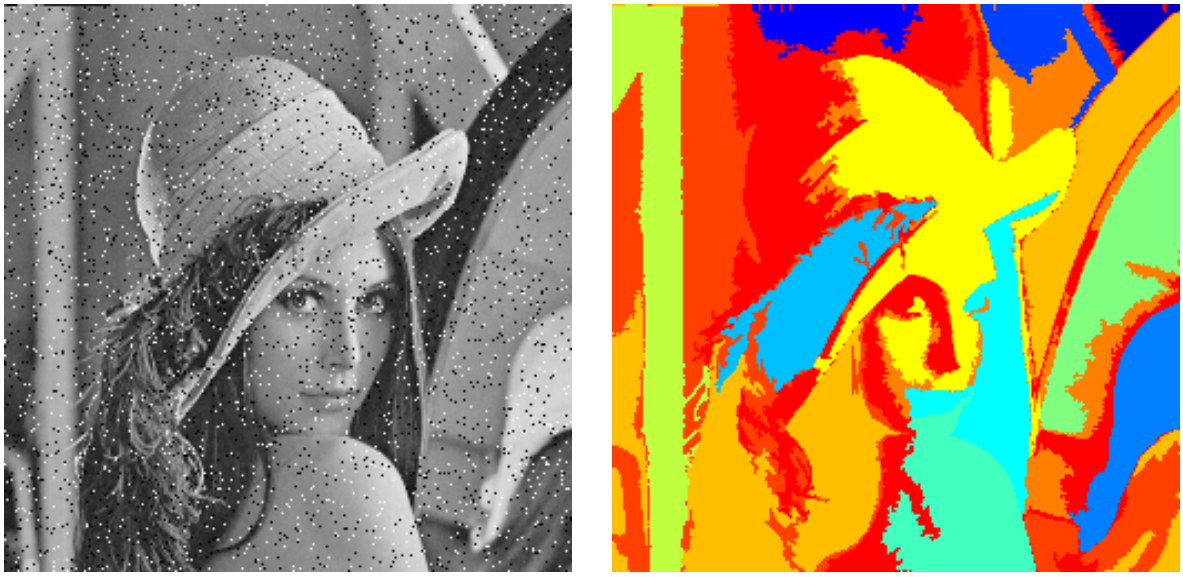
Figure 6.18: Segmentation example. (a) Original pears image. (b) Sub-tree labels. Trenches depth threshold 450.

The last operation in the algorithm is needed to deal with two problems. One problem is that small regions are left sometimes near object borders. Those regions are removed by the size criterion. It is important to notice that this size criterion does not cause the algorithm to be a regular size-based pruning, because the size criteria should be an order of magnitude smaller than the required objects. Another problem is that in many cases a number of similar zones remain in the tree. Those zones have a son-father relationship, and are stacked one above the other, with a slight difference in size. The proposed solution of this problem is to prune vertices that have a slightly bigger father. A threshold of 10% size difference was found to be a good choice.

An example of this segmentation is shown in Fig. 6.18. In this image the segmentation roughly follows the pears boundary. Another example of segmentation can be seen in Fig. 6.19.

6.3.2 Comparison to the Shape Tree

A natural alternative to the proposed representation is the shape tree, discussed in Chapter 2.2.3. Monasse and Guichard list in [13] the properties of an opening operator based on the shape tree. It may be a good idea to check which properties the two approaches have in common and which they do not. Let us denote an opening operator on an image u using the extrema watershed tree as $\gamma_B(u)$, where B is a structuring



(a) (b)

Figure 6.19: Segmented Lena image. (a) Original Lena image. (b) Sub-tree labels. Trenches depth threshold 800.

element. We shall address only properties that are more general and not specific to [13].

Contrast invariance. If g is a contrast change (an increasing real valued function), then:

$$\gamma_B(g(u)) = g(\gamma_B(u)) \tag{6.1}$$

Negative invariance. Like the shape tree based operator, $\gamma_B(u)$ has the feature of negative invariance:

$$\gamma_B(-u) = -\gamma_B(u) \tag{6.2}$$

Idempotent. As the shape tree operator, $\gamma_B(u)$ is idempotent:

$$\gamma_B(u) = \gamma_B(\gamma_B(u)) \tag{6.3}$$

This property is due to the fact that the proposed operator is a morphological opening in a complete inf-semilattice, and according to [21] such an operator is idempotent, increasing and anti-extensive.

Fig. 6.23 and 6.22 present effects of opening and erosion using extrema watershed

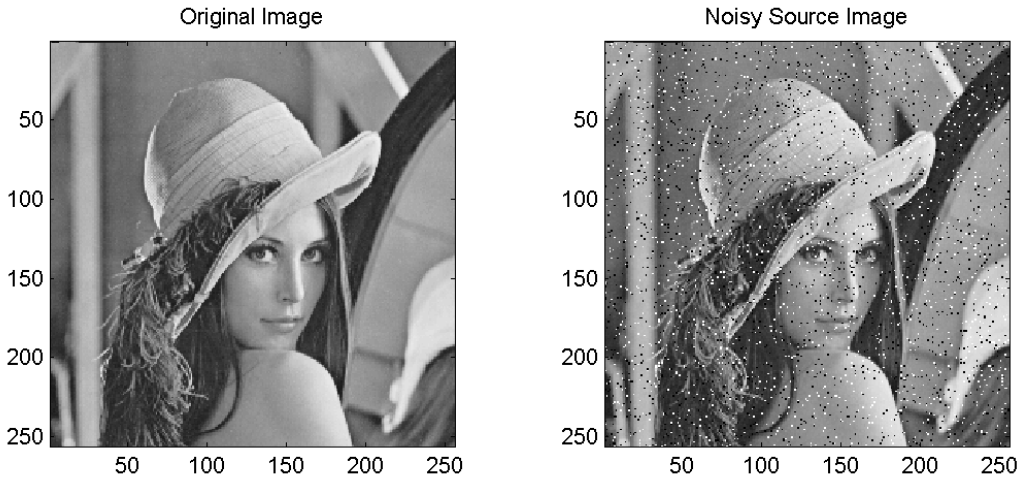
tree and shape tree on the corrupted image shown in Fig. 6.20. A difference between the two approaches is shown in Fig. 6.21. It is interesting that there is a major difference in the erosion result, but almost no difference in the opening result. This is because in the erosion picture of the extrema watershed tree trenches are revealed. The trenches in extrema watershed tree exposes flat zones located deep into the tree. There is no similar effect in a shape tree because its tree structure is different.

Note that noise connected to the image border is not filtered in the Tree of shapes and is filtered in the EWT tree. Noise is left by the shape tree filter because its "fillhole" operator does not consider this boundary noise as a hole that can be filled during tree creation process.

6.3.3 Filtering using Extrema watershed tree versus traditional morphological filtering

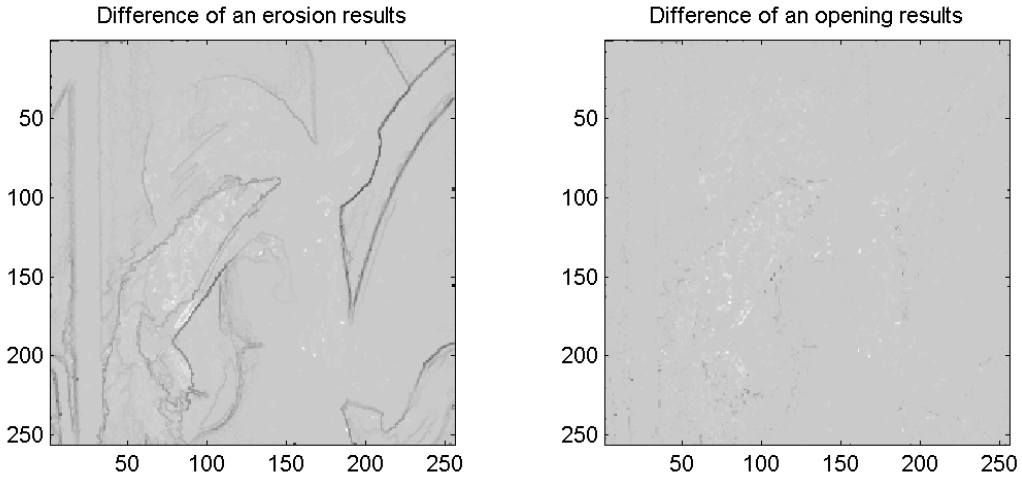
In this section, filtering results using the EWT are compared with traditional erosion, dilation, opening, closing, open-close, close-open operators and median filter in a complete lattice of gray level functions. In Figs. 6.24 and 6.24, results of conventional morphological filters (erosion, dilation, opening, closing) are shown. As expected, their results are not symmetric with respect to a gray and dark pixels. Because of that, those filters cannot be compared with the extrema-watershed tree.

An additional morphological filter that has a pseudo symmetrical effect is a combined open-close or close-open filters as described in Chapter 8 of [3]. Their results are shown in Fig. 6.26. In Fig. 6.27 a median filter result is shown and compared to the EWT result. Those two figures enable a comparison between the proposed filter and traditional approaches. It can be seen the the EWT filtered result is in general similar to an open-close and close-open results, but without a bias towards dark pixels as a close-open or towards a light pixels as an open-close filter. The median filter is unbiased, but a number of noisy pixels have survived the filtering by it, and the median filter lacks other important features. For example it is not an idempotent filter.



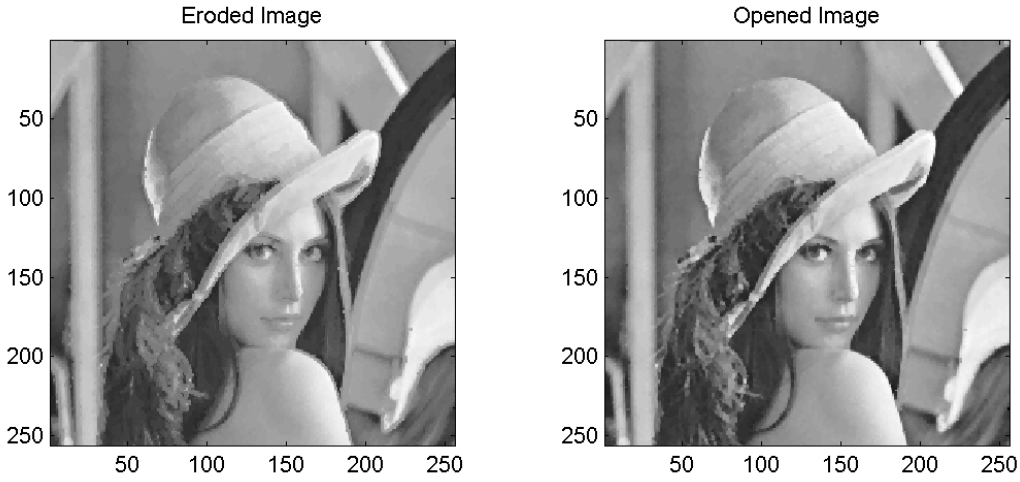
(a) (b)

Figure 6.20: A Lena image corrupted by the Salt&Pepper noise. (a) Original Image (b) Noisy Lena Image.



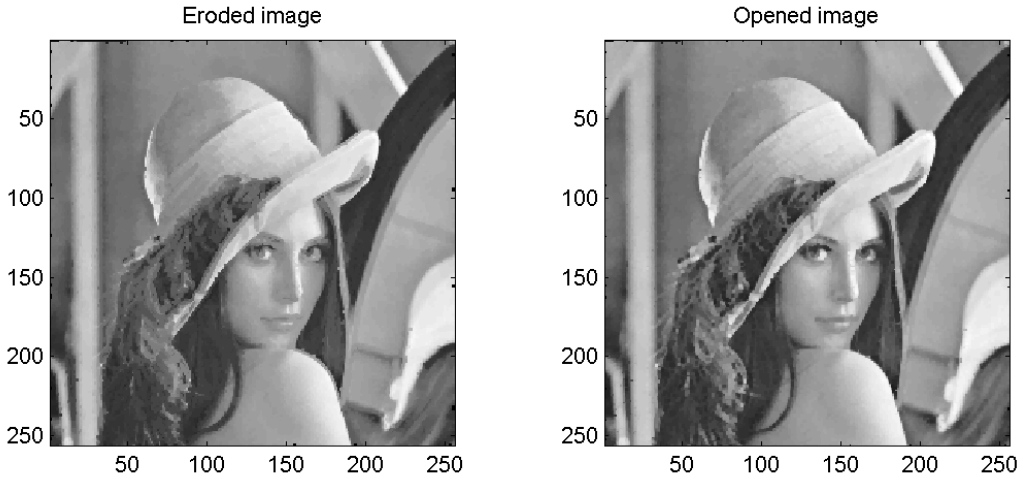
(a) (b)

Figure 6.21: Difference between filtering results of Tree of shapes minus filtering results of Extrema watershed tree of Noisy Lena Image. In both filtering approaches we have used structuring element 2×2. (a) Difference between erosion results of Tree of shapes minus erosion results of Extrema watershed tree (b) Difference between opening results of Tree of shapes minus opening results of Extrema watershed tree.



(a) (b)

Figure 6.22: Filtering Noisy Lena Image using Extrema watershed tree. (a) Inverse transform of Extrema watershed tree eroded by structuring element 2×2 (b) Inverse transform of Extrema watershed tree opened by structuring element 2×2.



(a) (b)

Figure 6.23: Filtering Noisy Lena Image using Tree of shapes. (a) Inverse transform of Tree of shapes eroded by structuring element 2×2 (b) Inverse transform of Tree of shapes opened by structuring element 2×2.

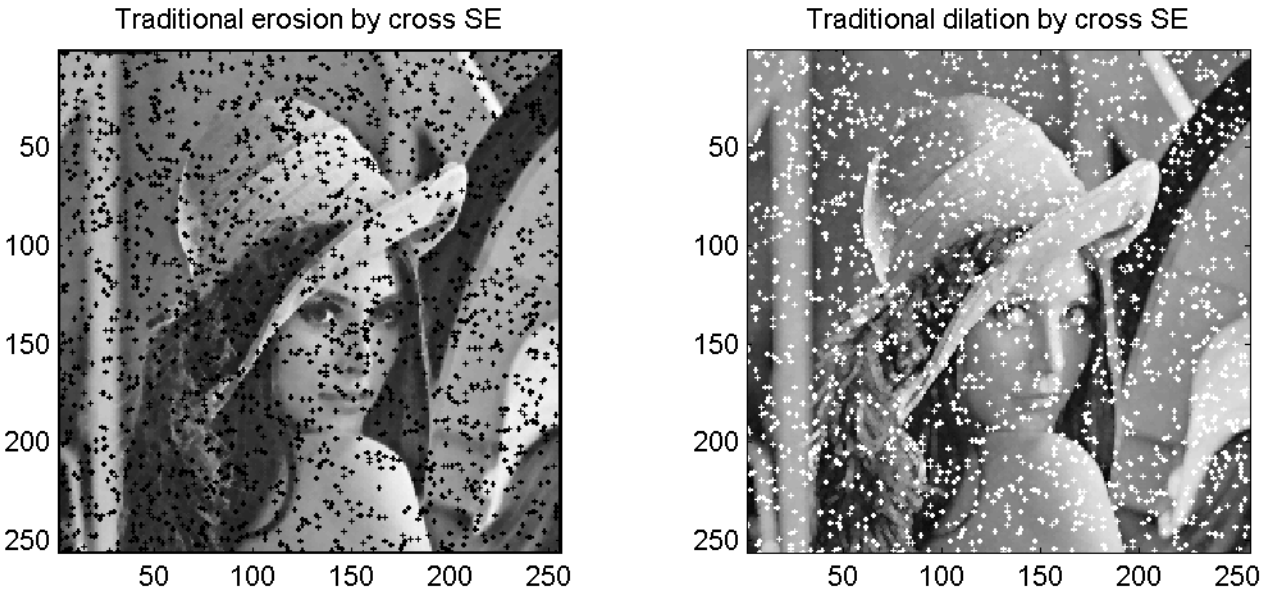


Figure 6.24: Traditional erosion and dilation of a gray scale image.

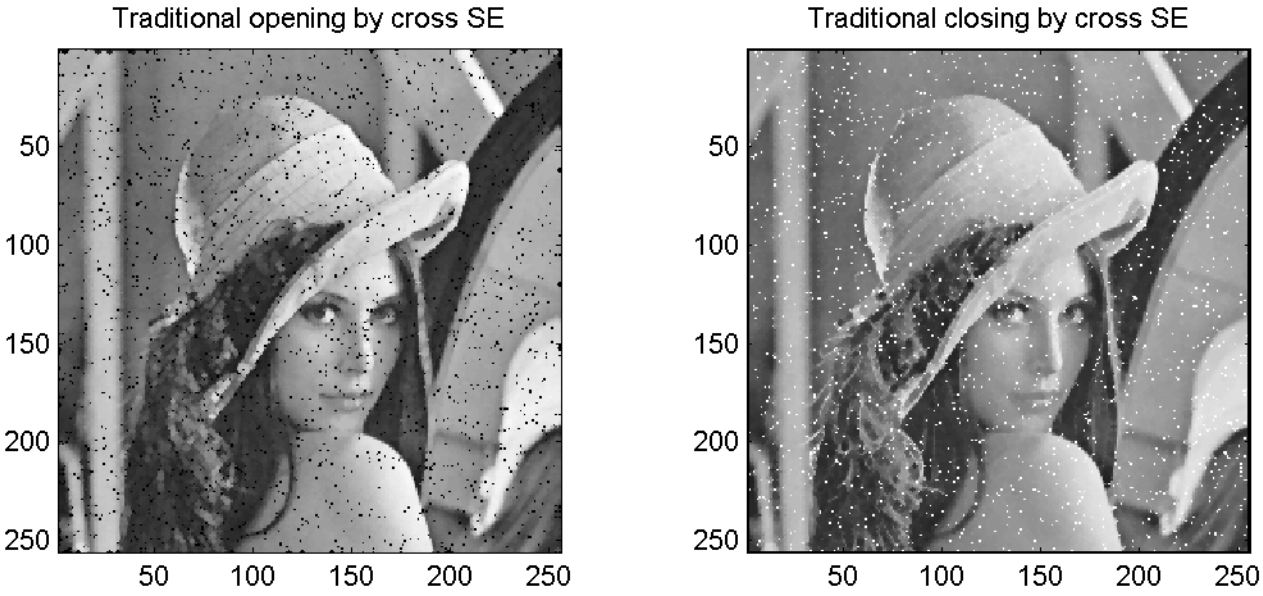


Figure 6.25: Traditional opening and closing of a gray scale image.

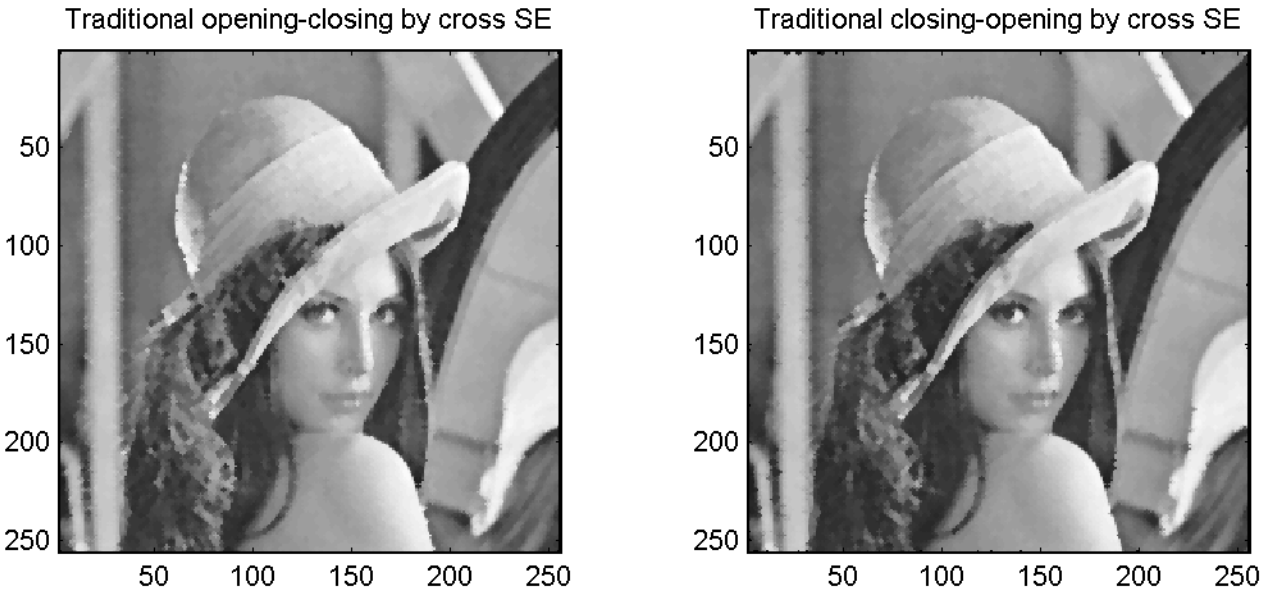


Figure 6.26: Traditional pseudo-dual open-close and close-open operators.

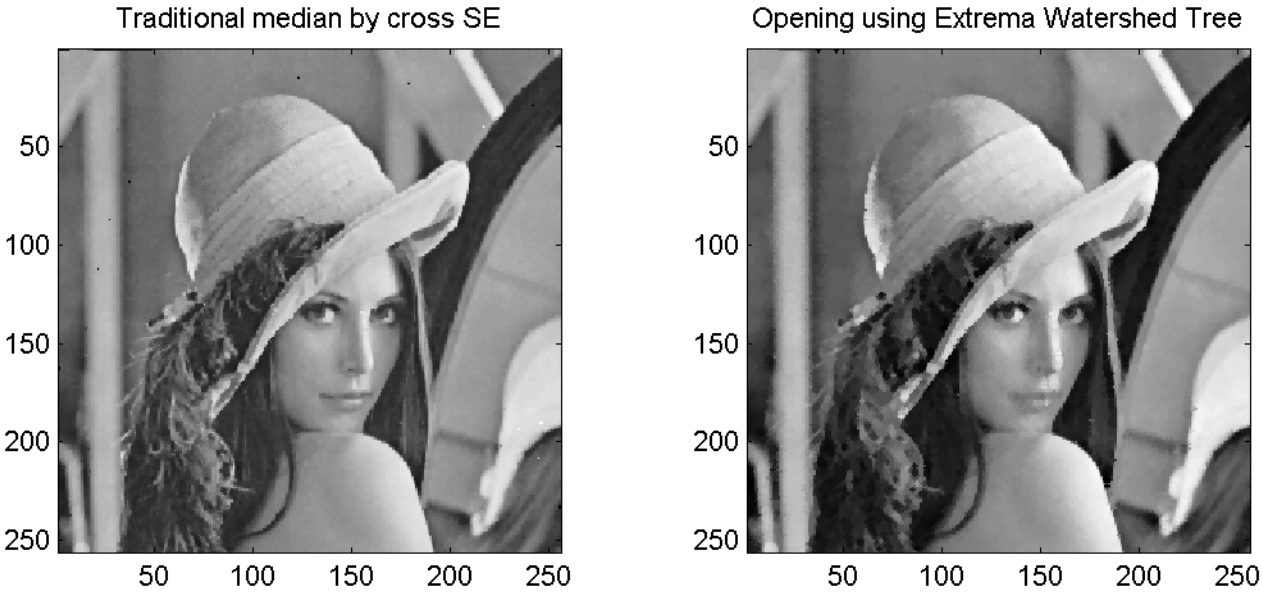


Figure 6.27: Median operator compared to the self-dual opening using Extrema watershed tree.

6.4 Application examples

A common usage of opening and opening by reconstruction operators is cleaning noisy images as an intermediate step in various applications. Our operators, defined on extrema watershed tree, have an important feature: duality. In the next paragraphs we will show a number of possible applications that may use the EWT-based filtering as an intermediate step. The presented applications are OCR (Optical Character Recognition) algorithms and dust and scratch removal from images.

6.4.1 Pre-processing for car license plate number recognition

Morphological operators are a common choice for applications that require image simplification. One kind of such an application is pre-processing for OCR (Optical Character Recognition) algorithms. We have chosen a specific OCR algorithm, used for recognition of license plate numbers, that was developed in [25]. This algorithm uses a mask for each digit and looks for the best correlation among these masks with an image. Any noise that exists in the image degrades the correlation value and interferes with the recognition.

This particular algorithm has an important benefit: it calculates a coefficient of recognition quality. This coefficient enables us to measure recognition improvements, when using different pre-processing algorithms. The exact definition of this coefficient is presented in [25].

Consider an example license plate shown in Fig. 6.29. This image is corrupted with different kinds of noise, see Fig. 6.30. The correlation is done on the gray scale image shown in Fig. 6.31. It is difficult to automatically recognize the numbers without pre-processing, because of the noise level. We have done opening by reconstruction, using the structuring element shown in Fig. 6.28. The diameter of this SE is equal to the width of digits lines. This size of SE assures that we filter as much noise as possible without damage to the numbers. The resulting image can be seen in Fig. 6.32. In this figure, almost all the noise is gone, and the image is much clearer and suitable for OCR. The effect of the proposed pre-processing can be seen in table 6.1. Without the pre-processing, the OCR failed to recognize correctly the plate number. The pre-processing, not only removes the noise, but also simplifies the image. Therefore, when

Method used	Recognition result	Quality coeff.
Image without noise, without pre-processing	70-587-07	3.75
Noisy image, without pre-processing	20-687-07	3.47
Averaging filter	70-587-02	3.65
Median filter	**70-587-07**	3.75
Regular open by reconstruct	79-687-07	3.65
Regular quasi dual open by reconstruct	**70-587-07**	3.79
EWT filter	**70-587-07**	**3.84**

Table 6.1: OCR results comparison with different pre-processing methods.

the pre-processing is applied, the quality coefficient increases and correct recognition is achieved. Sometimes OCR results are even better for pre-processed noisy image, than for not noisy image.

In order to measure the effectiveness of the proposed approach, a number of other pre-processing filters were used. A simple averaging filter, using the same structuring element was used to filter the image. The result is shown in Fig. 6.33. As can be seen in Table 6.1, this particular pre-processing gives a very poor result. A median filter with the same structuring element, which is shown in Fig. 6.34, gives better results than averaging. Opening-by-reconstruction results, using regular gray scale image morphology, are shown in Fig. 6.35. The image looks much better, although it is apparent that the filter is not self dual. The OCR, as seen in Table 6.1, successfully recognizes the plate number and the quality coefficient goes higher. In order to handle the lack of duality of regular morphology, we took the following approach: Two opening by reconstruction operations were performed using the scheme shown in equation 6.4.

$$I_{quasi-self-dual} = 256 - OR(256 - OR(I_{original})) \tag{6.4}$$

Where OR is opening by reconstruction operator, 256 is the maximum gray level and $I_{original}$ is the input image.

The filter result is shown in Fig. 6.36. The result looks fine but is a bit more corrupted than the EWT filtering result in Fig. 6.32. The OCR results, shown in Table 6.1, are slightly worst than the EWT filtering.

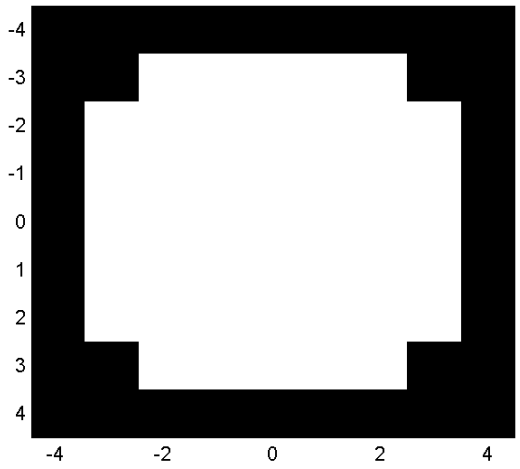


Figure 6.28: The structuring element used in opening by reconstruction.

6.4.2 Noise filtering for text OCR

Another example of pre-processing for OCR can be seen in Fig. 6.37(a). The figure shows text taken from a National Geographic journal. This text is printed on non-uniform background, and worse, part of the text is white and part of the text is black. In addition the picture is corrupted by added random white and black pixels. If we want to use this image for OCR, the image must be cleaned. We used an EWT-based opening filter for this purpose. As can be seen in Fig. 6.37(b), both white and black text was cleaned in the same way. Both white and black noise pixels were removed, and the letters were almost undamaged. Although we didn't run an actual OCR, we believe that the filtered image would have a higher success rate in OCR. For comparison, we filtered the image with the median filter and opening filter based on shape-tree. It can be seen in Fig. 6.38 that the median filter leaves some noisy pixels, and the shape-tree opening is almost identical to the EWT opening.

Figure 6.29: License plate image without noise.

Figure 6.30: License plate image corrupted with noise.

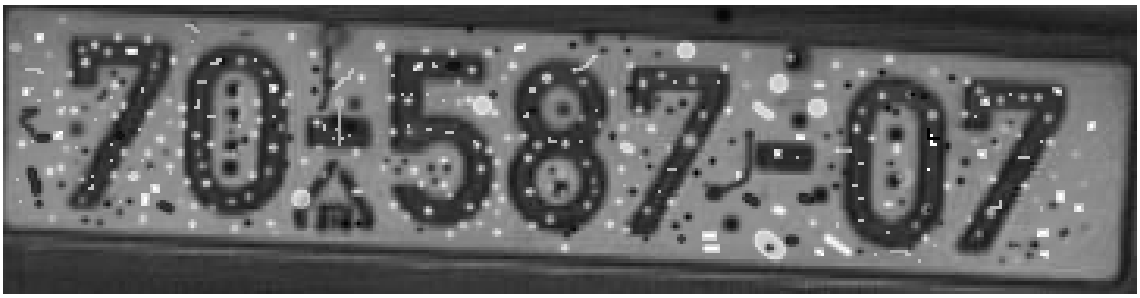


Figure 6.31: License plate image in gray scale, as used by the OCR.

Figure 6.32: License plate image filtered by EWT-based opening by reconstruction.

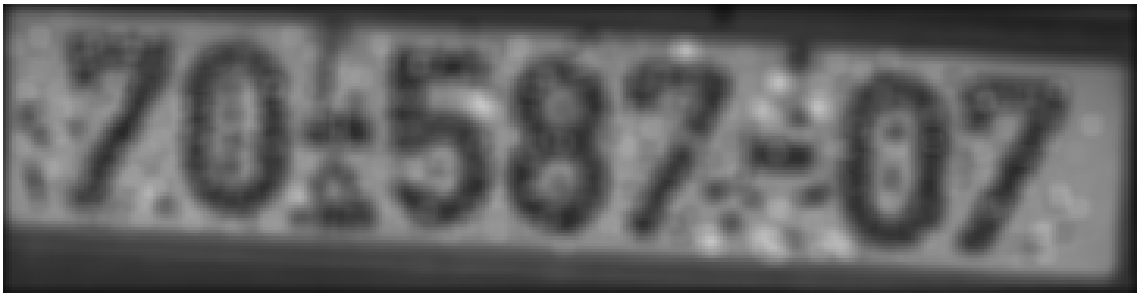


Figure 6.33: License plate image filtered with an averaging filter.

Figure 6.34: License plate image filtered with a median filter.

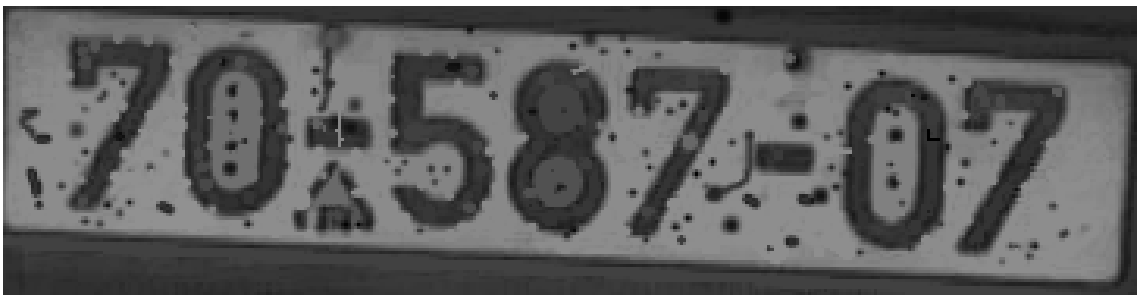
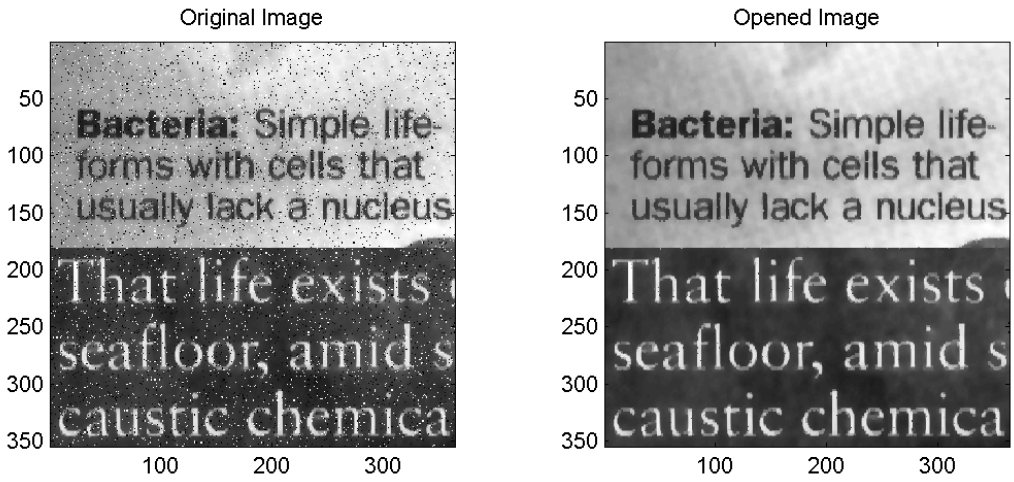


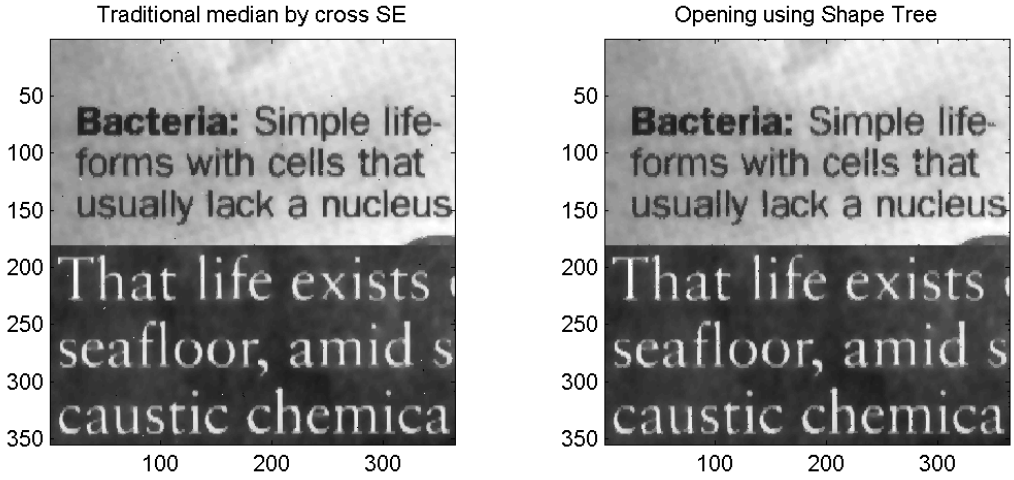
Figure 6.35: License plate image filtered with regular opening by reconstruction.

Figure 6.36: License plate image filtered with regular self dual opening by reconstruction.



(a) (b)

Figure 6.37: (a) A text image corrupted by Salt&Pepper noise. (b) Image restored by an opening on the watershed tree with 2x2 SE.



(a) (b)

Figure 6.38: (a) Image restored by a median filter. (b) Image restored by an opening on the shape tree with 2x2 SE.

6.4.3 Initial step for dust and scratch removal

Another example uses opening by reconstruction as an initial step for an application that removes dust and scratches from images. The elements that remain after filtering by the opening by reconstruction are completely preserved, including their edges. The elements that are removed by the opening by reconstruction are completely removed, including their edges. This feature enables one to catch candidates for dust and scratch during the initial step. The output of the initial step is a top-hat (TH) filter that includes all details that were filtered out by opening by reconstruction (OR) from the original image $I_{original}$, and is given by: $TH = I_{original} - OR(I_{original})$. Later steps of the application can make further analysis of that image in order to decide which object is dust or scratch and which is not. Afterward, objects that are dust or scratch, according to decision rule, are removed from the image.

We have compared different methods for this application. The comparison was done using the energy level of the top-hat image and qualitative evaluation of the extent of dust and scratch removal. As the energy level of the top hat image is lower, and as the dust and scratch removal is better, so the filter is declared to be more efficient. The energy is defined by equation (6.5).

$$Enrg = \|f\|_2 = \sqrt{\sum_{i,j \in f} f(i,j)^2} \tag{6.5}$$

Where f is the image and $Enrg$ is the image energy level.

We used averaging and median filters for comparison. Table 6.2 summarizes the energy levels of the top-hat images. Figures from 6.39 until 6.42 show top-hat images for different structuring elements and image magnifications. The objects in the top-hat images are the potential candidates for dust and scratch for the later steps of the application.

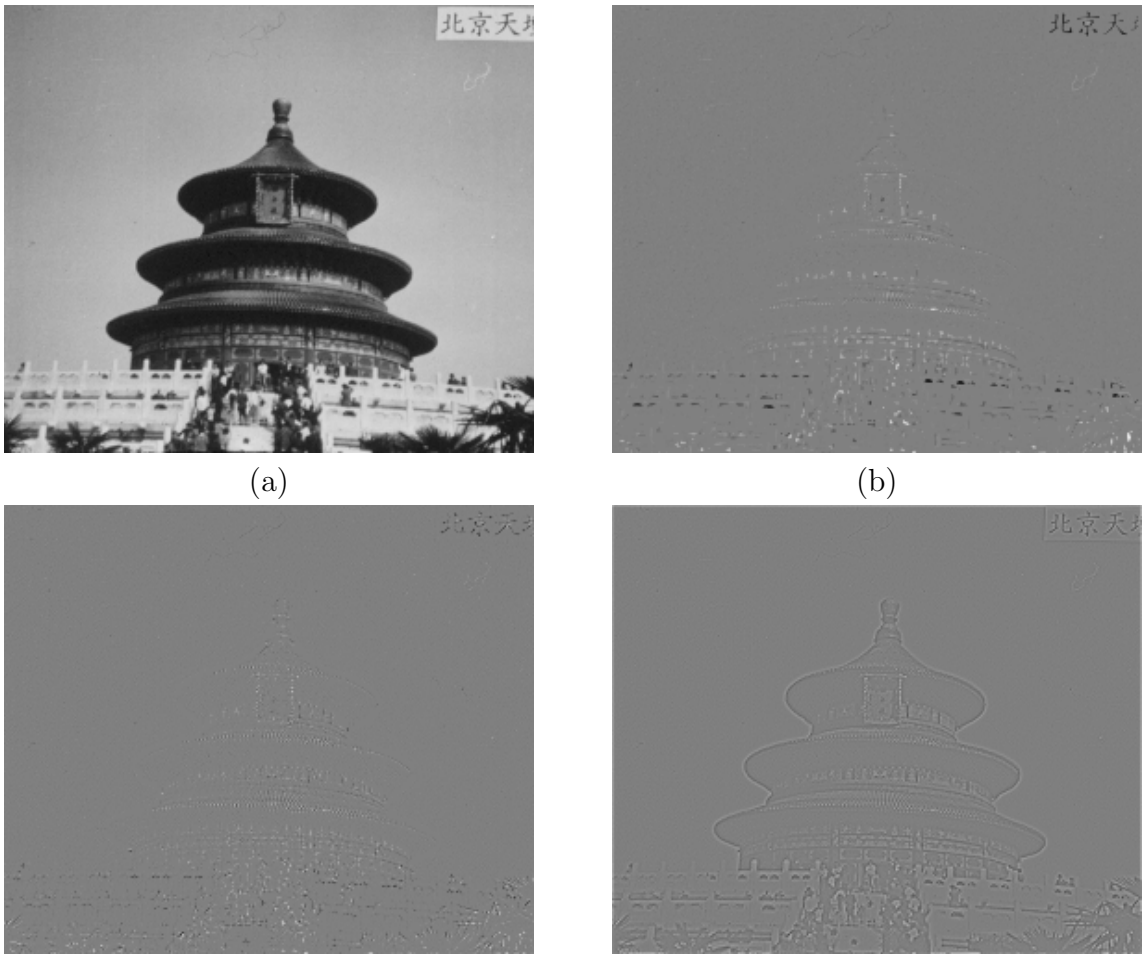
For all structuring elements, the averaging filter catches all the edges of the image. In addition, the energy of the top-hat image is higher for the averaging filter than for the other filters listed in Table 6.2. This means that the averaging filter catches too many candidates for dust and scratch. Therefore, it is less suitable for dust and scratch removal application, than the other compared filters.

For a given image, the energy is higher for the median than for the filter based

Method used	**Energy (cross SE)**	**Energy (SE 3x3)**	**Energy (SE 5x5)**
Averaging filter	1799	2643	3979
Median filter	**1233**	1971	3162
EWT filter	1376	**1737**	**2895**

Table 6.2: Energy results comparison, with different pre-processing methods.

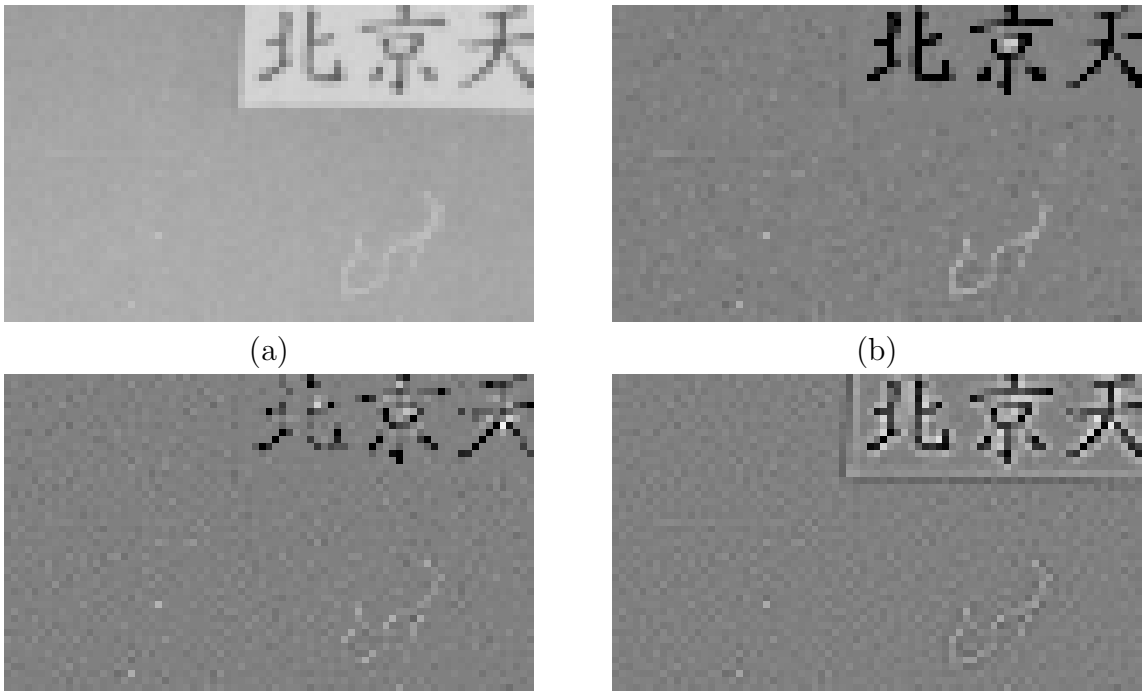
on EWT, for all structuring elements, except for "cross". For "cross" structuring elements, median does not catch the entire real dust and scratch, as shown in Figure 6.40. So we conclude that the filter based on EWT is better for dust and scratch removal applications than the other compared filters.



(a) (b)

(c) (d)

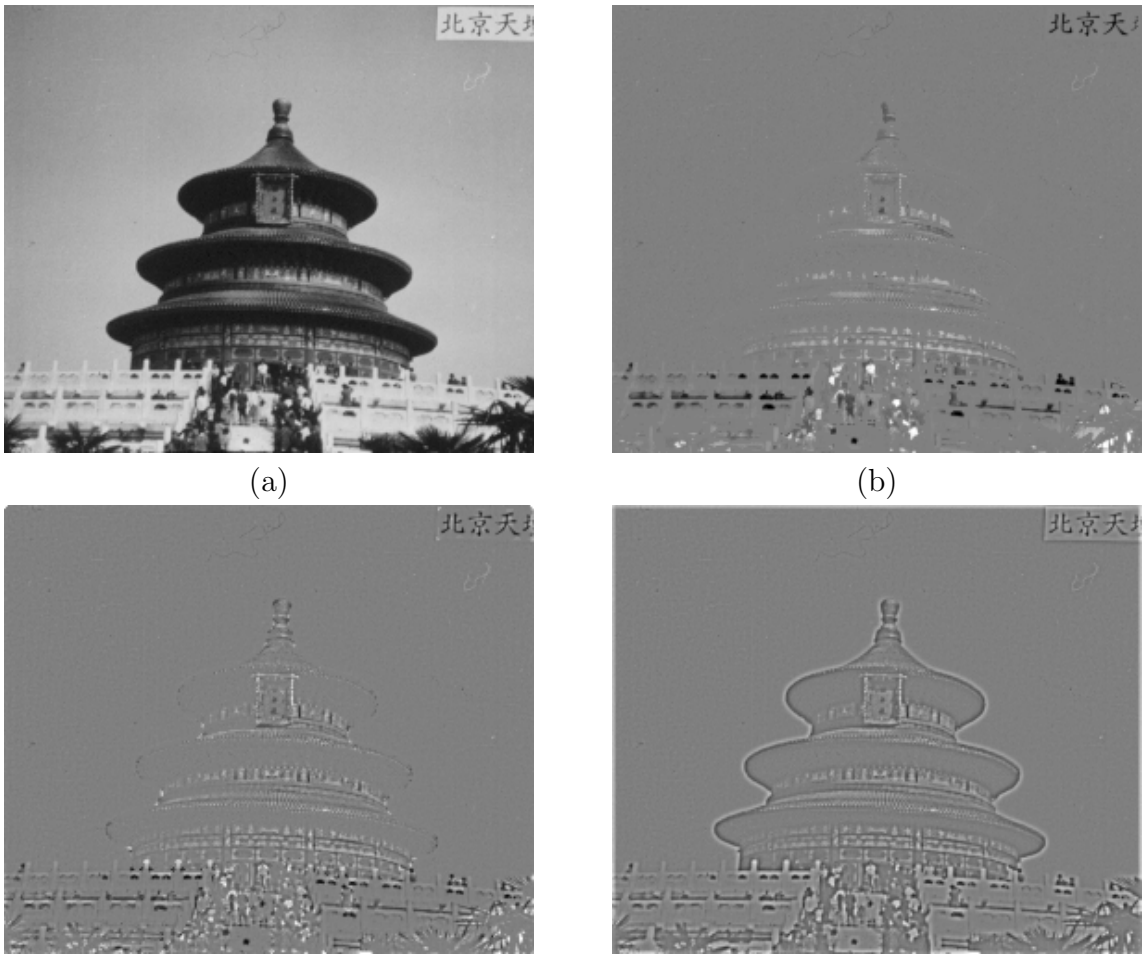
Figure 6.39: Top hat, using cross SE, as a pre-processing for dust and scratch removal. (a) Original image (b) Top hat by reconstruction based on EWT (c) Top hat using median (d) Top hat using an averaging filter.



(a) (b)

(c) (d)

Figure 6.40: Zoom in. Top hat, using cross SE, as a pre-processing for dust and scratch removal. (a) Original image (b) Top hat by reconstruction based on EWT (c) Top hat using median (d) Top hat using an averaging filter.



(a) (b)

(c) (d)

Figure 6.41: Top hat, using SE 5x5, as a pre-processing for dust and scratch removal. (a) Original image (b) Top hat by reconstruction based on EWT (c) Top hat using median (d) Top hat using an averaging filter.

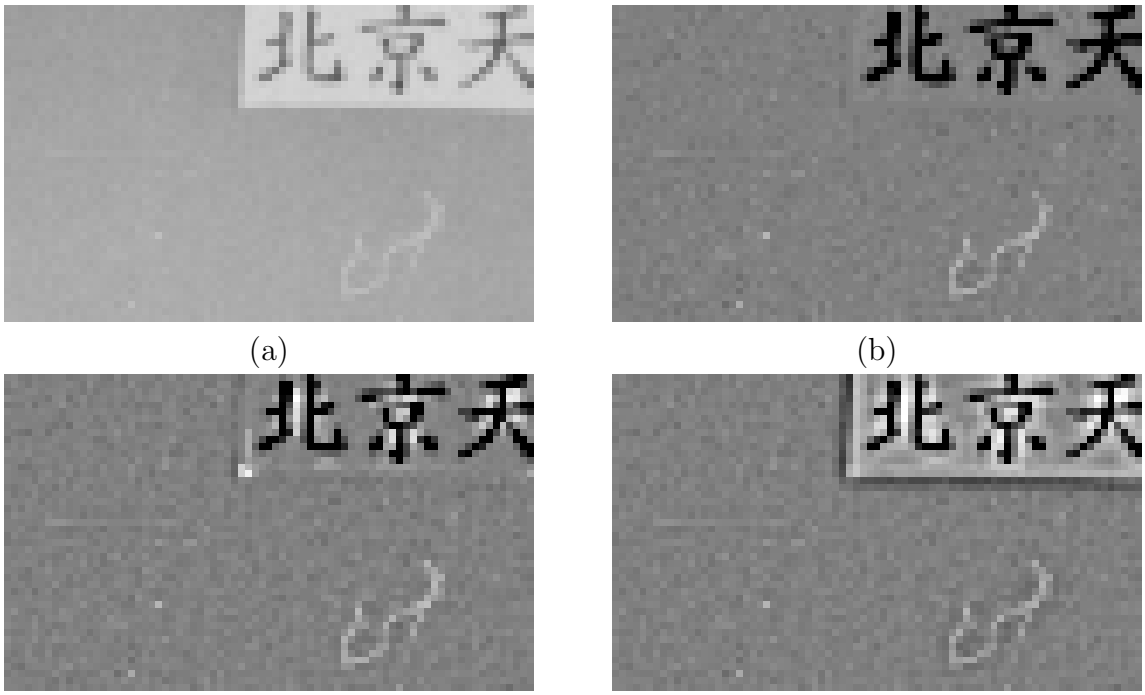


Figure 6.42: Zoom in. Top hat, using SE 5x5, as a pre-processing for dust and scratch removal. (a) Original image (b) Top hat by reconstruction based on EWT (c) Top hat using median (d) Top hat using an averaging filter.

6.5 EWT conclusions

A new tree representation is proposed and studied. In addition, a number of applications are proposed and examined. Those applications include pre-processing for OCR, noise filtering, and initial step for dust and scratch removal. It is shown that EWT-based filtering gives better results for those applications, as compared to other filtering techniques. The EWT-based filtering produces results similar to the classical morphological filtering, but achieves a better results when the self-duality is required.

Chapter 7

Conclusions and further research

7.1 Conclusions

In this work, we present a general framework for producing new morphological operators that are compatible to given tree representations. This framework was shown to be useful by presenting an example of a new tree representation, the Extrema Watershed Tree, with the set of morphological operators based on this tree, and giving a number of applications examples. Those examples were pre-processing for OCR, noise filtering and initial step for dust and scratch removal. EWT-based filtering performs well on tasks suitable for classical morphological filtering. When self-duality is required, EWT-based filtering achieves better filtering results as compared to other filtering techniques, including classical morphological filtering.

Moreover, a study of the trench problem was performed, and a number of possible solutions were presented, along with finding a potential in using those trenches for segmentation purposes.

7.2 Further research topics

A number of further research directions can be stated on the basis of this work:

- The developed general framework opens a gate for many possible sets of new morphological operators. Each set is based on a specific tree representation. Therefore, finding and exploring additional tree representations can bring new

useful operators. Moreover, one may look for more applications for the additional tree representations and for the developed one. For instance, further exploration of segmentation capability using the trench phenomenon in EWT is needed.

- The current definition of tree transform does not assure the existence of an image semilattice. Further research is needed in order to find necessary conditions for tree representation, so that the inf-semilattice structure is carried from the tree-representation domain to the image domain.

- Although a number of optimizations were done to the developed algorithms, they are still not suited for real time implementations. Therefore, improvements of the efficiency of the presented algorithms can be performed.

Appendix A

Algorithm 4 RAG image representation

The gray-level image can be represented as a graph, where each node stands for the image flat zones and the edges connect all the neighboring nodes. The algorithm, that we have developed for graph building, involves the following stages:

1: Initialize *current label* to zero. Mark all pixels as not processed. Start arbitrary from the image border. Insert all the pixels of the image border into *fifo1*.
2: **if** *fifo1* is not empty **then**
3: Get pixel *current p1* from *fifo1*.
4: **if** this pixel was not processed yet **then**
5: Increment emcurrent label, give this new *current label* to the current pixel *current p1* and insert this pixel into the *fifo2*. Go to 12.
6: **else**
7: Go to 2.
8: **end if**
9: **else**
10: End algorithm.
11: **end if**
12: **if** *fifo2* is not empty **then**
13: Get pixel *current p2* from *fifo2*.
14: **else**
15: Go to 2.
16: **end if**
17: **for** every neighbor of the current pixel *current p2* **do**
18: **if** it was not processed yet **then**
19: **if** it belongs to the same flat zone as *current p2* **then**
20: Insert it into *fifo2*.
21: **else**
22: Insert it into *fifo1*.
23: **end if**
24: **else**
25: **if** it belongs to another flat zone **then**
26: Then connect this flat zone in the graph to the current flat zone.
27: Go to 12.
28: **end if**
29: **end if**
30: **end for**

Bibliography

[1] J. Serra, *Image Analysis and Mathematical Morphology*, vol. 1. London: Academic Press, 1982.

[2] J. Serra, *Image Analysis and Mathematical Morphology*, vol. 2 : theoretical advances. London: Academic Press, 1988.

[3] Pierre Soille, *Morphological Image Analysis*. Springer-Verlag Berlin Heidelberg New York, 1999.

[4] Renato Keshet (Kresch), "Mathematical morphology on complete semilattices and its applications to image processing," *Fundamenta Informaticæ*, vol. 41, pp. 33–56, January 2000.

[5] H.J.A.M. Heijmans and R. Keshet, "Inf-semilattice approach to self-dual morphology," *Journal of Mathematical Imaging and Vision*, vol. 17, pp. 55–80, July 2002.

[6] H.J.A.M. Heijmans, "Self-dual morphological operators and filters," *Journal of Mathematical Imaging Vision*, vol. 6, no. 1, pp. 15–36, 1996.

[7] A.J.H. Mehnert, P.T. Jackway, "Folding induced self-dual filters," in *Mathematical Morphology and its Applications to Image and Signal Processing* (J. Goutsias, L. Vincent, and D. S. Bloomberg, ed.), pp. 99–108, Kluwer Academic Publishers, Boston, 2000.

[8] H.J.A.M. Heijmans, "Connected morphological operators for binary images," *Computer Vision and Image Understanding*, vol. 73, no. 1, pp. 99–120, 1999.

[9] P. Salembier and J. Serra, "Flat zones filtering, connected operators and filters by reconstruction," *IEEE Transactions on Image Processing*, vol. 8, no. 3, pp. 1153–1160, August 1995.

[10] Philippe Salembier, "Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval," *IEEE Transactions on Image Processing*, vol. 9, pp. 561–576, April 2000.

[11] P. Salembier, J. Llach, L. Garrido, "Visual segment tree creation for mpeg-7 description schemes," *Pattern Recognition*, vol. 35, pp. 563–579, March 2002.

[12] P. Monasse and F. Guichard, "Fast computation of a contrast-invariant image representation," *IEEE Transactions on Image Processing*, vol. 9, pp. 860–872, 2000.

[13] P. Monasse and F. Guichard, "Scale-space from a level lines tree," *Visual Communication and Image Representation*, vol. 11, pp. 224–236, 2000.

[14] V. Casellas and P. Monasse, "Grain filters," *Journal of Mathematical Imaging and Vision*, vol. 17, no. 3, pp. 249–270, November 2002.

[15] C. Ballester, V. Casellas and P. Monasse, "The tree of shapes of an image," *ESAIM: Control, Optimization and Calculus of Variations*, vol. 9, pp. 1–18, 2003.

[16] Renato Keshet, "Homotopy semilattice." HP Technical Report, 2004. www.hpl.hp.com/techreports/2004/HPL-2004-1.pdf.

[17] Renato Keshet, "Shape-tree semilattice," *Journal of mathematical imaging and vision*, vol. 22, pp. 309 – 331, May 2005.

[18] Reinhard Diestel, *Graph Theory*. Springer-Verlag New York, Electronic Edition ed., 2000.

[19] Philippe Salembier and Luis Garrido, "Connected operators based on region-tree prunning," vol. 3, pp. 367 – 370, September 2000.

[20] Pascal Monasse, *Morphological representation of digital images and application to registration*. PhD thesis, Universite Paris IX-Dauphine, 2000.

[21] Renato Keshet (Kresch), "Extension of morphological operations to complete semilattices and its applications to image and video processing," *Mathematical Morphology and its Applications to Image and Signal Processing (Proc. of ISMM'98)*, pp. 35–42, June 1998.

[22] F. Meyer, "Topographic distance and watershed lines," *Signal Processing*, vol. 38, pp. 113–125, 1994.

[23] Philippe Salembier, Luis Garrido and David Garcia, "Auto-dual connected operators based on iterative merging algorithms," *IEEE Transactions on Image Processing*, vol. 7, no. 4, April 1998.

[24] C. Vachier, L. Vincent, "Valuation of image extrema using alternating filters by reconstruction," *Neural, Morphological, and Stochastic Methods in Image ans Signal Processing*, pp. 94–103, July 1995.

[25] S. Vichik , R. Sandler , A. Rosen , "Moving car license plate recognition." www.cs.technion.ac.il/Labs/Isl/Project/Projects_done/cars_plates/finalreport.htm, 1999.