Algorithmic patterns for morphological image processing

d' Ornellas, M.C.

Citation for published version (APA):
Chapter 1

Introduction

The white rabbit put on his spectacles. Where shall I begin, please your Majesty? he asked.

Begin at the beginning, the King said gravely, and go on till you come to the end: then stop.

*in Alice’s Adventures in Wonderland, by Lewis Caroll.*

Software for image processing is known to be difficult to write. Only one or just a few algorithms working on a specific type of image are relatively easy to code. Difficulties arise when one aims at an image processing package that tries to meet the theory.

Software developers in image processing are often confronted with questions concerning the design and implementation of algorithms. The reason for the questions is that developers as well as end users rarely share the same knowledge. In turn, what an algorithm precisely does remains often unclear. Specific implementation details and different pixel types, not taken into account in the initial phases of the design, are examples of the problems that often occur. The answers for such questions are not straightforward. What software developers often do in image processing is to show how to *program around* the implementation details.

No doubt much of these problems show up because in the past no one paid attention at specifying the *modus operandi* of the algorithms. An algorithm implemented this way will lack a precise definition of its run-time behavior. One of the reasons is that software developers still do not make the most of the functionality provided by the available programming tools.

Great efforts have gone into building scientific libraries that are dedicated to a particular application. The main issue is to manage large numbers of data types involved in a given domain. For instance, in image processing, an algorithm implementation should indistinctly accept two-dimensional and three-dimensional images (isotropic or not), regions, region adjacency graphs, image and graph pyramids, sequences, collections, etc. Hence, the types of data contained in these structures are scalar (Boolean, integer, or float), complex, composed (e.g. color RGB), and so forth. Existing libraries
for image processing are usually dedicated to a particular data structure (mostly two-
dimensional images). The implementations of algorithms are commonly restricted to
few data types (mostly unsigned 8 bit integers) [16].

The first research problem addressed in this thesis within the scope of image
processing arises from these software development pitfalls: *Is it possible to combine
both theory and practice in such a way as to produce reliable, precise, and generic
algorithm representations in order to overcome the combinatorial explosion of code
needed to deal with all possible kinds of data structures and image types?* If this is true,
how can the algorithms be devised with the guarantee that the desired implementation
should be written once, how can it process data in an abstract way, and how can it,
at the same time, be efficient? Consequently, what is needed to address these issues
is a precise framework for the development of image processing software. If properly
constructed with a high level programming language, this framework should greatly
increase programmer’s productivity as programming tasks are considerably improved
due to replacement of large pieces of code by short algebraic statements.

Some scientific libraries have been using a generic programming approach to this
problem [17] [30] and a few of them are freely available on the Internet for various
domains: containers, graphs, linear algebra, computational geometry, differential
algorithms, neural networks, visualization, etc. These libraries enforce the use of
genericity for software architectural purposes and especially for algorithm design.

The basic concept of generic programming is termed as follows: given $X$ data
types, $Y$ containers (data structures), and $Z$ algorithms as components of a software
system, the generic programming paradigm provides a mechanism to reduce the
possibly $X \times Y \times Z$ implementations to $X + Y + Z$ implementations. Original
libraries in this paradigm were implemented in Scheme and Ada. However, it has been
popularized only recently with a C++ implementation called the Standard Template
Library (STL). Thanks to parameterization, generic programming allows to abstractly
represent data structures [37].

On the mathematical side, the correctness of generic algorithms offers greater
challenges than conventional algorithms. Often, one must create the appropriate
abstract concepts in terms of which concepts can effectively express and reason about
the behavior of an algorithm or collection of algorithms. The nature of the problem of
developing generic algorithms should be attractive to researchers in computer science
and mathematics, whereas the problem for developing conventional algorithms is often
regarded as difficult to conceive, to implement, and to maintain.

The intention in this thesis is not to cover all aspects of software development for
image processing but to concentrate efforts on the underlying concepts of mathematical
morphology and present a new approach to morphological software development. Such
approach should lead to generic and efficient algorithmic implementations.

During the last few years, morphological image processing has evolved from a
specialized imaging subject, to an important area of study within image processing
[9] [10] [11]. It is presented in many industrial applications that have been, and are
currently being developed. Mathematical morphology has become an essential tool
for image processing experts due to a vast number of tasks suitable for solution by
morphological methods.
Mathematical morphology aims at analyzing geometrical properties of objects. The analysis is quantitative in order to provide a complete framework for describing spatial organizations. So far, the use of such a framework has allowed for the development of a class of morphological algorithms to deal with binary, grey-scale, and recently also for color images.

The first study on a general framework underlying mathematical morphology was proposed in [50]. It was shown that such framework could be achieved if one starts from the assumption that the object space is a complete lattice. This idea has been carried further by various people [1] [26] [35] [44] [45]. Despite this accredited theoretical evolution and a large number of applications, software development for morphological image processing still lacks of a standardized, mathematical rigorous algebraic structure that is specifically designed for image handling [12] [13] [43].

Recently, much of the morphological software-related practice has evolved from merely a set of fundamental operators, such as erosions and dilations, to the concept of an entire software package. Morphological tools within these packages usually range from classical filters [33] [34] to watersheds [5]. Problems emerge when one tries to fulfill the specific morphological algorithm requirements based on the complete lattice without a well-founded software framework.

For instance, currently released morphological software packages as if Micromorph and the MMac toolbox for the Khoros environment are still facing these problems. This has led to an entire reformulation in such systems and in some cases, to the development of complete new software, in order to continue standing as a useful tool. Sometimes, slightly different formulations among systems yield a different algorithm representation which, in turn, leads to a different implementation. Moreover, having distinct implementations in practice for the same morphological operator in theory, generates more documentation and makes software difficult to program.

Given the obstacles in morphological software development, the following problem statement emerges naturally: How can the algebraic theory of mathematical morphology be used to build a general framework for the development of morphological algorithms?

What is required is that the algorithms should appear as a sequence of operators and images into a well-designed framework where each morphological operator should be expressed as a composition of elementary operators. Such framework should be able to produce statements in terms of low-level operations that are tied to the algebraic representation of the fundamental structures upon which images and images operators are designed. Particularly, these algorithms should be following the generic programming paradigm. That is, they should be independent of both data types and data structures.

A general framework for morphological image processing will be very handy since it formalizes the entities like images, pixel values, operators, etc. Such a framework will be connected to the level of details provided by the theory in morphological image processing.

This chapter is organized as follows: section 1.1 discusses about the reasons why mathematical morphology is so important for non-linear image processing and why morphological software is needed in several applications. Then section 1.2 describes
and characterizes morphological algorithms in terms of structure, implementation techniques, and mathematical notation. Section 1.3 reviews some of the most used software packages and image libraries with respect to mathematical morphology. The aspects considered include general information, technical requirements, maintenance, documentation, and support. Section 1.4 refers to the state of art in morphological software, concentrating on features relevant to software design in mathematical morphology.

1.1 The Need for Morphological Software

Mathematical morphology is the science of shape and structure, based on set-theoretical, topological and geometrical concepts [20] [48] [49] [55]. Over the past years, mathematical morphology has grown as a powerful method for image processing and analysis, receiving considerable attention due to its solid theory. It provides a set of tools for image processing and analysis that has been used to a wide number of applications. As a nonlinear branch of image and signal processing, mathematical morphology represents a rupture with classical linear processing and facilitates the application of various mathematical concepts and evolving disciplines to image processing [21] [22] [24] [25].

These mathematical concepts and disciplines include medical image segmentation [18] [27] [28], nonlinear statistics [33] [34], logic [56], geometry [25], scale-space [8] [29] [57], topology [39] and various algebraic systems such as lattice and group theories. While these disciplines have been used in the classical image-processing techniques, they appear more naturally within the context of mathematical morphology. They are central to its development and application.

The recent success of mathematical morphology in noise removal [47], background normalization [63], shape recognition [23] [53] [58], object flaw detection [19], object feature extraction [15] [54], and segmentation [36] [51] [52] [60] has made a number of researchers take notice, further explore, and use morphological techniques. Most of the software packages for image processing have now included in their operation set the morphological operations. In addition, a considerable effort has been done in the development of new specialized morphological libraries and toolboxes.

Several algorithms have been designed and implemented to identify characteristic topographic features on images, such as large and deep valleys, sharp crests, high summits, and watersheds; these features are then used to find an edge, segment the image, remove some artifacts, or localize an object [6] [7] [32] [38]. However, the original implementation derived directly from the definition is particularly inefficient to compute. In addition, its implementation requires large computer resources [61]. Consequently, new design techniques may be applied to produce effective algorithm implementations for morphological image operators.

In view of this, the most important strategic goal of morphological software is to serve the needs of application developers and end users. Obviously, the key to this is programmability, which allows them to control their own tools by writing the software they need. Key solutions for particular applications must be developed easily.
By providing both a solid theoretical infrastructure with a rich set of morphological tools, and coordinating the development of broad application frameworks for specific areas, a morphological package will achieve its goals for programmability.

1.2 Characterization of Morphological Algorithms

This section describes and characterizes morphological algorithms in terms of structure, implementation techniques, and mathematical notation. Algorithms are usually designed to solve a particular problem efficiently and with a minimum of effort. The significance of an algorithm is generally determined by two factors: how it solves the problem it aims for and how it implements the solution.

For instance, an erosion algorithm must implement a local comparison between a pixel \( p \in f \) and all other pixels within the neighborhood defined by the structuring function \( g \). As another example, some morphological algorithms (e.g. skeletons), should be implemented to produce results as close as possible to the continuous framework when applied to a discrete one.

The main characteristic of the majority of morphological algorithms is a large number of iterations over elementary operations. It should be noted, however, that in most iterative operations, relatively few pixels really change value in the intermediate stages. This offers a chance for developing efficient algorithms.

The efficiency of a morphological algorithm may be defined using three main criteria [59]: speed, accuracy, and flexibility. Accuracy means that the algorithm should give valid results. Flexibility can be defined as the way an algorithm adapts to different image types and data structures. However, most algorithms can not achieve the criteria all at once.

In addition, software needs to be reusable. Implementing reusable algorithms seems to be more difficult in mathematical morphology than in other fields of computer science. In part, this stems from the fact that the vast amount of image data to be processed makes efficiency one of the highest priorities. Until recently, most programming had to comply with the efficiency of the underlying hardware. Since the hardware was continually changing, reusability was virtually impossible to achieve. Due to the increasing performance and widespread application of standard computers, reusable morphological algorithms have become a realistic option, but efficiency still pays an important role.

Therefore, reusable algorithms may be applied either for some basic morphological processing tasks, such as erosions and dilations, or as elementary building blocks for more complex transforms like thinnings, thickenings, skeletons, and reconstruction.

1.2.1 Morphological Implementation Techniques

The most common implementation techniques for morphological algorithmic implementations are based on parallel, sequential, and queue-based algorithms.

In a parallel algorithm, the result does not depend on any of the other image data. Erosions and dilations allow for a parallel implementation because they can be carried
out independently for every pixel and the result of the computation for each pixel is stored in a new image.

In a sequential algorithm, the data are run through in a predefined scan. Forward and backward scans are frequently used since they simplify memory access in computer implementations. In a sequential algorithm, the result of a computed pixel value has an influence on the value of all pixels processed after it, in contrast to parallel algorithms. Sequential and parallel algorithms are mathematically equivalent, and the former should be competitive in processing time required if a sequential computer is used [46].

Queue-based algorithms are provided with a mechanism to control the number of pixels in any iteration, which requires a specific scanning order of the pixels. It is made in such a way that every pixel, over a number of iterations, is treated only once, at the very moment when its neighborhood is sufficiently well known to determine its value. From a computational point of view, queue-based algorithms are very efficient.

These three classes of algorithms will be carefully studied in the sequel of this thesis.

1.2.2 The Role of Notation

Consider the erosion of an image \( f \) with a structuring function \( g \). A mathematical notation for this operator is usually represented by either \( f \circledast g \) or \( e_g(f) \). The first notation focuses on the perceived similar role of the image to be eroded and the structuring function that encodes the spatial dependency of the operation. The second notation focuses on the operator nature of the erosion: it transforms one image \( f \) into a new image \( e_g(f) \), where \( g \) just serves as a parameterization of the operator. The second notation is the preferred way in the literature on morphological image processing. An option was made in this thesis to stick to the second notation.

In practice, there is a waist of notation and code to deal with the same generality. What all erosions have in common is more than just the name, which is an important conceptual observation on its own: the different operators share large parts of the algorithms. Indeed, for each new image type an entirely new version of the algorithm is written. The different names for essentially the same operator let alone the fact that not all functionality is available for all different types of images or different types of erosions.

1.3 Review of Morphological Software Packages

This section reviews some of the most used software packages and image libraries with respect to mathematical morphology. The aspects considered include general information, technical requirements, maintenance, documentation, and support.

Currently, there are several image processing software packages that include mathematical morphology in the set of available tools due to its relevance to image segmentation, non-linear filtering, pattern recognition, and image analysis. These tools range from a poor coverage, i.e. just a few morphological operators like erosions,
dilations, and set operations to a complete toolbox including not only fundamental operators but also specialized tools for filtering, reconstruction, and segmentation.

The software packages have different complexity levels depending on the needs of the researcher and end user. In the area of image processing, there are not only computer scientists but also physicians, mathematicians, biologists, and any kind of scientist who needs to process images in order to analyze and recognize their properties. Image processing software packages can be commercial, shareware, or even freeware. The majority of them are oriented to specific platforms like Unix, Linux, and Microsoft Windows since their related graphical user interfaces are tied with the operating system capabilities and functionalities.

It is not the intention in this section to cover all image processing software packages and their related morphological tools. The goal is to make an overview of some currently available morphological software packages like Mmacht toolbox for the Khoros system; Morph toolbox for Matlab; Micromorph; and Visilog. The aspects to be considered include general information, technical requirements, maintenance, and support.

1.3.1 Mmacht

Khoros is considered one of the most famous software environments designed for research in image processing [40] [41] [42]. Khoros consists of a collection of small, independent modules which can be plugged together to accomplish very complicated tasks. Since all modules are independent of each other, it is quite easy to write new modules or transform existing image processing algorithms into a Khoros conforming form. An intuitive and flexible user interface makes Khoros an ideal platform for rapid prototyping, solution exploration, and scientific visualization applications. In fact, the known applications cover a very broad spectrum: industrial inspection, medical diagnosis, optical measurement, remote sensing, semiconductor processing, optics, medical imaging, ecosystem analysis, cell biology, etc.

The use of a data flow visual programming environment called Cantata within the Khoros system allows users to easily bring together these large amount of applications and have them working as a cohesive whole. Programs from the Khoros system are represented as visual objects called glyphs. When accessed within the Cantata visual environment, a Khoros program is called an operator. To build a visual program, the user selects the desired programs, places the corresponding glyphs on the workspace, and connects them to indicate the flow of the data from program to program, in the workspace [64] [65].

Khoros was designed to be portable and extensible. It relies on existing standards (X11 Windows and Unix), incorporates tools for software development and maintenance, contains a flexible data exchange format, provides tools to export and import

*©São Paulo State University (USP), National Institute for Space Research (INPE), and Campinas State University (UNICAMP)
†©SDC Information Systems
‡©Centre de Morphologie Mathématique, École Nationale Supérieure des Mines de Paris.
§©NOESIS VISION Inc.
standard data formats, and includes an algorithm library. Programs can also be organized as independent subsystems, called toolboxes, which can easily be integrated to the system [31].

Following a component based approach, a toolbox for the *Khoros* software environment has been developed in order to enhance its potentiality by adding a new set of morphological tools. *Mmach* is designed to deal with one-dimensional signals, binary, grey-scale, and multiple band images. Each program has a specialized algorithm for grey-scale and binary images that is automatically chosen according to the input data type [3] [4].

*Cantata* suits quite well with mathematical morphology and brings an excellent environment for algorithm development. The right combination of operators gives the solution for a specific problem.

The architecture of the toolbox is organized in four groups of programs:

**Basic level:** Holds the elementary operations and operators of mathematical morphology like addition, dilation, distance transform, erosion, intersection, negation, subtraction, toggle and union;

**First level:** Holds operators that use one basic operator at least. These operators are: anti-dilation, anti-erosion, closing, conditional dilation, conditional erosion, conditional thickening, conditional thinning, inf-generating, morphological gradient, opening, sup-generating, thickening, thinning and threshold;

**Second level:** Contains operators that use each basic operator more than once. These operators are: close by segment, dilation by segment, erosion by segment, inf-canonical, extended intersection, extended union, N-closing, N-conditional dilation, N-conditional erosion, N-conditional thickening, N-conditional thinning, N-thickening, N-thinning, opening by segment, sup-canonical, alternate sequential filters, and symmetrical difference;

**Third Level:** Contains operators that uses an *a priori* undefined number of basic operators. These operators are: center filter, homotopy changing, close holes, closing by reconstruction, conditional exoskeleton by thickening, conditional skeleton by thinning, exoskeleton by thickening, last erosion, labeling, morphological skeleton, N-conditional bisector, objects on frame-off, opening by reconstruction, regional maximum, regional minimum, skeleton by thinning with multiple parameters, skeleton by thinning, skeleton by influence zones (SKIZ), and watershed.

*Mmach* provides complementary functions like an interface for the support, generation and rotation subsets of the $3 \times 3$ structuring element, an interface for the generation of binary or gray scale images that represent disks, the equality operator and the less than operator [2]. It also includes code optimizations for structuring function decomposition in order to speed up the elementary operators.

The *Mmach* toolbox is freely available. It has a web site on the Internet at http://www.dca.fee.unicamp.br/projects/Khoros/mmach/tutor/mmach.html with additional information, tutorials, examples, and extra help. *Mmach* 1.4 is the latest
version. Source code is also available in C language as well as compiled binary versions. It depends on Khoros environment and runs on Unix and Unix-like platforms. MMach also has a mailing list and an increasing number of users. With respect to documentation, each program of the toolbox has an on-line help associated that gives the definition of the operator and a set of well known parameters to extract useful image information.

Another driving force behind its success within the scientific community was the inclusion of MMach in the Digital Image Processing courseware, known as DIP. DIP’s purpose is to give users a hands-on approach to image processing through an extensive number of experiments including mathematical morphology. It can also be used as a self-study guide or for any type of training format.

Despite of the worldwide acceptance of MMach as a software tool for morphological image processing, it presents some inherent drawbacks as follows:

- elementary operators for binary images have more properties than the corresponding ones for grey-scale images. Additional algorithms are also provided for the distance transform, opening and closing by reconstruction, labeling, and watersheds. This characteristic shows that the MMach makes a clear distinction in terms of image semantics besides having specialized algorithm implementations to deal with different semantics;

- MMach is data type oriented, which means that additional algorithms must be implemented for every single data type. Therefore, it has a large set of programs dealing with the same morphological operator. When executing a given operation or operator that make use of different types, MMach needs to choose the appropriate one, making use of data type conversion operators and yielding an extra overhead in terms of processing time. This is not only a characteristic of MMach itself but the Khoros system as well;

- the large set of morphological operators derived by semantics and data type distinction, generates code proliferation, which makes code reutilization a very difficult issue.

In addition, MMach is not very easy to use. Time needs to be invested to acquire some kind of expertise before getting the most of the toolbox.

1.3.2 Mmoph

MATLAB is an integrated technical computing environment that combines numeric computation, advanced graphics and visualization, and a high-level programming language. It is an outgrowth of the LINPACK and EISPACK projects, and has been evolving for a number of years to the current system.

The name MATLAB is an abbreviation for MATrix LABoratory. As the name suggests, it is especially designed for matrix computations: solving systems of linear equations, computing eigenvalues and eigenvectors, factoring matrices, and so forth. It contains

©KRI, ISTEC, Ramiro Jordan, Roberto Lotufo.
a variety of graphical capabilities, and extends through programs written in its own programming language. In addition, it is used in a variety of application areas including signal and image processing, engineering, control system design, and medical research. The open architecture makes it easy to use MATLAB and companion products to explore data and create custom tools that provide early insights and competitive advantages.

A significant value of this environment is that it is extensible, since the user can create new functions and toolboxes for the environment. These functions and toolboxes are plain text files containing code and are called M-files. MATLAB also provides a compiler that automatically converts M-files to C and C++ code.

Morph toolbox is a software for image analysis and signal processing for MATLAB. It is composed by a family of morphological operators and operations based on lattice operations. These operators and operations are quite useful for restoration, segmentation and quantitative analysis of images and signals. The toolbox contains state-of-the-art morphological operators, implemented by the most efficiently known algorithms. The available operators range from the classical morphological filters, to reconstruction, connected filters, and watersheds.

Morphological operators are written hierarchically based on elementary operators, called dilation and erosion. The toolbox includes implementations for dilation and erosion and permits the creation of any other morphological operator through its constructive approach. Some operators have specialized and hardcoded implementations geared to efficiency like distance transform, watershed, reconstruction, labeling and area opening.

Apart from functions to deal with data type conversion, image handling and visualization, measurements, relations and operations between images, structuring elements, and intervals, Morph contains a vast number of other functions organized according to their applicability:

**Dilations and Erosions:** Contain elementary functions like erosion, dilation, conditional erosion and dilation, distance transform, and geodesic distance transform;

**Morphological Filters:** Include the classical morphological filters like opening, closing, center filter, and the alternate sequential filtering (ASF);

**Connected Operators:** Hold area opening and area closing, opening and closing by reconstruction, inf-reconstruction and sup-reconstruction, regional minima and maxima, valley and peak removal, ASF by reconstruction, minima imposition, and hole filling;

**Residues:** Hold morphological gradient, opening and closing top-hat, open and close by reconstruction top-hat, ultimate erosion, and morphological skeleton;

**Sup-generating and Inf-generating:** Contain functions like sup-generating (hit-miss), inf-generating, intersection of inf-generating operators, union of sup-generating operators, and adaptive threshold;

**Thinning and Thickening:** Hold thinning and thickening, conditional thinning and thickening, skeleton by influence zones (SKIZ), and watersheds.
Mmorph toolbox supports flat and non-flat structuring elements and represents them using decomposition and separability methods, which increases the performance of the corresponding dilation and erosion.

With respect to the documentation, Mmorph has an on-line reference manual with information for each function of the toolbox including a formal definition, command syntax, parameters, a short description, and examples. Mmorph is also illustrated by several demonstrations, which show the morphological solution of real-life image processing problems. Some of the application areas covered are machine vision, medical imaging, and desktop publishing, document processing, food industry and agriculture.

Mmorph is a commercial product and can be seen as a collection of functions implemented as mex files that depend on a dynamic library, called libmorph. It has a web site on the Internet at http://www.mmorph.com with additional information, extensive examples, tutorials, recommended literature, and mathematical morphology related links. The toolbox has a 30 days evaluation license that is freely available in the web site. Mmorph 0.13 is the latest version. It depends on MATLAB environment and is supported for three platforms: Win95/98/WinNT, Linux and Unix/Solaris.

Mmorph functions can be embedded into larger software projects or standalone applications. By implementing and compiling new algorithms with the MATLAB compiler, one can make use of Mmorph capabilities the foundation for useful image processing applications.

Despite of its state-of-art operators and the use of software engineering tools for its development, it exhibits some deficiencies as follows:

- like in the Mmach toolbox for the Khoros environment, Mmorph needs additional algorithms for both binary and grey-scale images. This particularly shows that it is semantically oriented, which means that different implementations must be developed for all kinds of semantics supported by the toolbox;

- Mmorph deals with grey-scale and binary images and is data type oriented. Consequently, more code needs to be written for every single data type. In other words, type oriented algorithms need to be implemented and included in the toolbox. When a Mmorph function is called, it needs to automatically choose the right one based on the data type, producing an execution time overhead. Like in the Mmach for the Khoros environment, this is not only a characteristic of Mmorph itself but the MATLAB environment as well;

- due to its semantic and data type oriented characteristic, Mmorph suffers from code proliferation and lack of reuse.

### 1.3.3 Micromorph

Micromorph is an image processing and analysis software package that implements several algorithmic innovations of mathematical morphology. The software package is not only educational with a great number of exercises and applications described in detail, but a tool for application development. Developed by the Centre de Morphologie Mathématique, École Nationale Supérieure des Mines de Paris, it is perhaps
the first software package entirely devoted to morphological image processing and analysis.

The software package provides solutions for scientific problems; a programming environment for problem solving; a software development environment and tools; and libraries for portability, scalability, data access, data visualization, and visualization applications.

Micromorph contains a dictionary that holds a list of all available words (functions and procedures). The primitive words constitute the basic core of the system. A very interesting feature is that the system allows the user to define new routines and add these routines to the dictionary, by using procedures and functions already available. The dictionary can be customized with new words in a very simple way. The programming structure of Micromorph language is also rather simple. Once an application has been programmed, compiled and added to the dictionary, it can be used to define tools to process images of increasing complexity.

Apart from primitive operators and functions to deal with image handling and visualization, measurements, rank operators, convolutions, simulation, neighborhood operators, arithmetical and logical operations with images, image sequences, structuring elements, segmentation, graphs, etc., Micromorph contains a vast number of other procedures and functions organized according to their applicability:

**Erosions and Dilations:** Include elementary erosions and dilations, erosion and dilation on square or hexagonal grid, erosion and dilation by a segment, erosion and dilation by an elementary square or hexagon, distance square or hexagonal distance function, erosion and dilation by a pair of points, contour of a set, Sobel and morphological gradients, isotropic erosion and dilation, isotropic distance function, cylindrical or conic dilation, dilation of a set by another set, erosion and dilation by a rhombododecaedron, erosion and dilation by a diamond, dilation by a ring or by the summits of an hexagon, polygonal dilations;

**Openings and Closings:** Hold elementary openings and closings, openings and closings using a linear structuring element, open and close top-hats, isotropic opening and closing, sup-close and inf-opening intersection, regularized gradients, infimum of directional openings;

**Geodesy and Connectivity:** Contain geodesic erosion and dilation, binary reconstruction from a marker set, opening and closing by reconstruction, area opening, reconstruction by a ring, geodesic distance, binary reconstruction, leveling transform;

**Applications of Geodesy:** Hold removal of particles touching the image border, extraction of the first particle internal contour of an image, hole filling, extrema of a function, extended extrema, modification of the homotopy by markers, extrema of an image inside a mask, grain smoothing and closing, individual closing of particles, dynamics of the maxima or minima of an image;

**Filters:** Include alternate filter, alternate sequential filter, alternate sequential filter (ASF) using linear openings, center auto-median operator, morphological
center, contrast using the top-hat, isotropic alternate filter, alternate filter by reconstruction, toggle between the opening and the closing, binary median set, grey-scale median image;

**Skeletons and Maximal Balls:** Contain skeleton by maximal squares or hexagons, ultimate erosion, centroid, conditional bisector;

**Thinnings, Thickennings:** Hold binary thinnings and thickenings, elementary cycle of geodesic thinning or thickening, homotopic thickenings, homotopic thinnings, endpoints of the skeleton, multiple points, geodesic center, grey-scale thinnings and thickenings, erosion and dilation by a general structuring element, elementary grey-scale thinning and thickening, directional gradient, complete vector gradient, enhanced vector gradient, thinning and clipping, grey-scale hit-or-miss, grey-scale basic thickening, grey-scale thinning;

**Segmentation and Watersheds:** Contain clipping, skeleton by influence zones (SKIZ), marker-based and simple watersheds, distance function segmentation, jump contrast extraction, mosaic image, waterfall transformation, pyramid of mosaic images.

**Micromorph** provides support for a large set of structuring elements within the square and hexagonal grid like diamonds, disks, lines, rhombus, etc. They can be made flat or non-flat. It also contains specialized functions to deal with structuring element generation and rotation.

The documentation supplied with **Micromorph** makes it a real educational software package of mathematical morphology. This documentation includes on-line help files, a complete reference manual of the programming language of **Micromorph**, applications manual and a course of mathematical morphology. The applications manual consists of several chapters gradually introducing the notions of mathematical morphology, the algorithms needed for operator implementation, and applications to image processing. These notions and applications are introduced through tutorials and exercises.

**Micromorph** is a commercial product including a large collection of functions and procedures implemented in C. Additional information and a FAQ’s is available at http://cmm.ensmp.fr/micromorph web site. **Micromorph** 1.4 is the latest version. It is an independent software package and is only supported for Win95/98/WinNT platform. Unfortunately, there exists no demonstration or free evaluation version available.

Apart from the large number of functions and procedures to deal with image sequences, graphs, and three-dimensional mathematical morphology which makes **Micromorph** unique, perhaps, one of the major advantages of **Micromorph** comes with the course of mathematical morphology that is included in the software package. This course introduces the main concepts of mathematical morphology as well as their properties, from basic operators such as erosion and dilation to complex tools such as watershed based segmentation.

Despite of its state-of-art operators, **Micromorph** exhibits some deficiencies as follows:
• the power of **Micromorph** comes at a price of a high complexity. The system functionality accounts for more than a hundred words or transformations and a large library. As more and more algorithms are added, the size of the system continues to grow. Unfortunately, the design approach chosen for **Micromorph** has the tendency to make all parts of the system depends upon each other. It is not possible to split the system into independent parts - the **Micromorph** has to be used and understood as a whole;

• elementary operators for binary and grey-scale images have individual implementations. Therefore, different algorithms are needed. This characteristic easily shows that the **Mmach** makes a clear distinction in terms of image semantics, having specialized algorithm implementations to deal with that;

• **Micromorph** is data type oriented, which means that additional algorithms must be implemented for every single data type. Therefore, it has a large set of programs dealing with the same morphological operator;

• if a closer look is taken on how algorithms are implemented in **Micromorph**, one sees that they are closely coupled to the data structures as well as with the available set of structuring elements;

• due to its large set of morphological operators derived by the semantics, data type, and data structure distinction, code reutilization is extremely difficult to obtain, yielding code proliferation.

In addition, **Micromorph** includes a great number of secondary functions and procedures, making the system hard to understand and to use.

### 1.3.4 Visilog

**Visilog** graphical approach to digital imaging has been designed both to make things easier for the novice and to save development time for those more experienced in digital imaging. The system provides a large amount of image processing algorithms for **Windows** or **Unix** based platforms. It is also able to perform many two-dimensional morphological operations on grey-scale or binary images.

It includes flexible data structures and associated algorithms for all phases of image processing and analysis, low-level image processing, geometric and topologic feature representation, high-level object reconstruction, coordinate systems and data exchange facilities. Using **Visilog** libraries, the existing functionality can, to a certain extent, be adapted to new environments.

**Visilog** includes a **C**-based macro language. The macro recorder generates **C** code automatically. This provides the user with the advantage of being able to compile code and add them to the system. It also has a **C** language interpreter, allowing the user to program either complex or basic applications. In this regard, it resembles the **SCIL-Image** software package [62].

The system is made up of six different components as follows:
• **Viewer**: To acquire, load, view, annotate, edit and measure or print images;

• **Acquisition**: To control the live, snap and other functions such as frame averaging;

• **Processing Module**: To access imaging algorithms. The system provides a wide selection of image processing algorithms. The processing module contains over 500 functions used to process and analyze images;

• **Recorder**: To build script files containing major transactions as C programming procedures. Operations are recorded automatically allowing for easy and fast generation of macros. Customize applications and create run time versions using Visilog libraries through Visual Basic or Visual C++;

• **Analysis**: To analyze an entire image or individual objects. Visilog is able to transfer all analysis data to Excel or any other compatible spreadsheet;

• **Demo**: To access frequently used macros. The Demo mode allows to store user-defined macros or access Visilog supplied macros.

Visilog is used in machine vision, medical imaging, metallurgy and material science, electronic microscopy and microanalysis imaging, pharmaceuticals, color separation, petrochemical industry, granulometry, biology and cytology, gel analysis, aerospace, defense and surveillance, and non-destructive testing.

Except for the set of operators and functions that handles color processing; threshold and segmentation; arithmetic and Boolean operators; edge detection and linking; geometric operators; frequency domain transforms; linear and non-linear filtering; convolutions; and shape recognition, Visilog contains a compact set of binary and grey-scale morphological operators classified in line with their own characteristics:

**Erode and Dilate**: erosion, dilation, erosion and dilation by a line, erosion and dilation by a disk, and color erosion and dilation;

**Open and Close**: opening, closing, opening and closing by a lines, opening and closing by an arbitrary structuring element, and color opening and closing;

**Top-Hat**: top-hat;

**Morphological Filters**: auto-median, morphological, and proper filter;

**Classic Algorithms**: ultimate erosion, border kill, hole filling, first object, regional extrema, reconstruction, and watershed;

**Conditional Morphology**: propagation and geodesic distance;

**Special Points**: boundary, interior points, end points, triple points, and isolated points;

**Thinning and Thickening**: thinning, thickening, ring, and convex hull;
**Skeleton:** skeleton, centroid, bisector, pruning, skeleton by influence zones (SKIZ), thick skeleton, and thick pruning.

*Visilog* works with predefined structuring elements like squares, disks, segments(lines), circle, bi-point, diamond, as well as user-defined ones. The choice is based on the type of result desired and the purpose of the transformation. All structuring elements are flat but the system admits rectangular and hexagonal grids.

Each operator has a detailed documentation including examples, parameter usage, technical explanations, and graphics that can be accessed using an on-line help. On-line manuals assist the user to understand the system and get the most of its powerful set of operators.

*Visilog* is a commercial product and includes a compact but powerful set of morphological operators implemented in C/C++. It has a web site on the Internet at http://www.visilog.com with additional information, manuals, examples, newsletter, up-to-date information about releases, applications, etc. *Visilog* 5.02 is the latest version. It is a self-contained software package and is available with full code under Win95/98/NT, and Unix workstations. No evaluation or limited version is available.

The system makes a distinction between parallel and sequential morphological operators. In a regular operator, the structuring element examines the pixel values in the original image to establish the new pixel value. In a recursive operator, the new value established by the structuring element is affected by the previously changed values. Providing another way of implementing morphological algorithms is necessary for *Visilog*, producing implementations that are more efficient.

*Visilog* contains its own drawbacks, which are described as follows:

- it is semantic oriented. Grey-scale and binary images are treated distinctly, which means that different implementations must be developed for all kinds of semantics supported by the system;

- it is data type oriented. Therefore, distinct algorithms must be devised and implemented to deal with all kinds of data types supported by the system. Consequently, it yields an unwanted number of operators and a larger library.

Table 1.1 contains a list of 40 operators, which are often used in mathematical morphology. This list ranges from fundamental operators (e.g. erosions and dilations) to advanced operators (e.g. watersheds). A comparison is made in terms of their availability for Mmach, Mmorf, Micromorph, and *Visilog* software packages. Note the superiority of Mmorf and Micromorph with regards to the expressiveness of the operator set.

Based on the review in this chapter, it is clear that the software packages considered share some deficiencies, which are given as follows:

- algorithms must be designed and implemented for every single data type, i.e. they are data type oriented. This leads to specific implementations and to a wide variety of similar algorithms;

- code needs to be written for every single data type, generating code proliferation and lack of reuse. In addition, the large set of operators associated with
code proliferation produce several complications with respect to software main-
tenance.

These aspects show that software for mathematical morphology should be made
generic and should support a large variety of data types.

1.4 The State of Art in Morphological Software De-
velopment

This section refers to the state of art in morphological software, concentrating on
features relevant to software design in mathematical morphology.

As the discussion of criteria relevant to the review and evaluation of morphological
software have shown, there are a variety of features relevant to almost any application
whereas others will largely depend on the user's specific environment and requirements
(granulometries, graph morphology, color watershed segmentation, etc.).

1.4.1 Requirements

Certain qualifications can be made as to vital and basic elements any morphological
software should have. Especially important are those features which determine the
structure and complexity of the algorithms and features related to design and im-
plementation of the algorithms since the value and applicability of the software will
largely depend on these features. This means above all:

1. there should be support for a large variety of image data types. For instance,
Horus supports two-dimensional and three-dimensional images. The pixel values
of an image are scalar values, or a vector of scalar values. A scalar value is
represented by an integer value, one of byte, short, int or a floating point
value, one of float, double. The vector values contain 2 or 3 scalar values.\textsuperscript{II}
The list of image types is shown in table 1.2;

2. the software should be implemented using generic constructions in order to avoid
code proliferation and lack of reuse. This argument is treated in chapter 2;

3. the software should reflect the theory, i.e. the gap between programming and
the theory about mathematical morphology should be as small as possible. In
addition, the orthogonality of the operators in theory should be mapped in
software. Chapter 3 deals with this subject;

4. the software should consider speed, regardless the need for genericity. This topic
will be covered in chapters 4, 5, and 6.

A short glance at software packages, toolboxes, and libraries available suggests
that most software packages must fulfill the following requirements when considering
implementation aspects:

\textsuperscript{II}Note that support for geometrical structures [14] is left out since the focus is on general images.
the software must be platform independent;

- the development environment should comply with the current standards in software development, which means: C++ programming language and STL based on the current level of technology. C++ relies on the principle of abstract data types, inheritance, and patterns. In this way, the new code can be designed in such a way that the number of functions is reduced drastically while the number of different data types is increased. A set of elementary image processing operators may also be provided in order to make software development easier;

- the software must provide the environment with which new algorithms can be developed. Existing implementations of algorithms can be compared and evaluated. As such, it must be easy for users to add their own algorithms (or new data types if required), and to experiment with algorithms that have already been implemented, i.e. a flexible and reusable expansion of the software must be provided, allowing integration with the supplied private or third party libraries;

- the software must be easy and intuitive to use, which means that the user interface should function in a way similar to typical Windows applications and it must offer ways to export results of image processing algorithms, e.g. pasting resulting images in presentation applications and data to data processing applications;

- there must be a clear development documentation. Examples showing how to add functionality must be provided;

- the software should support images with distinct dimensionalities and different image types. The main requirement for additional types will be that both read and write functionality are available as well as conversions between image types;

- help information should be provided as much as possible.

This thesis concentrates on the first four fundamental aspects previously cited and leave implementational aspects to chapter 2.

Looking at the available morphological packages reviewed in this thesis with these requirements in mind, it seems that the majority of these packages do not meet the requirements. Morphological software development still means above all constructing a set of fundamental operators optimally suited to the majority of image processing applications where it is to be used.

If one takes into account the present state of the art of morphological software development, it seems that the list of requirements might well be useful in more than one way: like checklists for other types of software, it may also stimulate ones to revise their concepts and to improve morphological software development. Therefore, the following section returns to the problem statement with a birds-eye view on a non-exhaustive list of requirements any software developer should keep in mind when designing libraries for morphological image processing or general purpose image processing.
1.4.2 Problem Statement Revisited

A lot of research in the field of algorithms and data structures provides qualitative and quantitative analysis for structures like hierarchical queues, heaps, binary trees, or priority queues. Nevertheless, there is little mention of how to implement these structures in ways that emphasize maintainability, adaptability, and reuse. That is to say, it is difficult to identify a comprehensive set of data structures, which would fit the need of image processing.

That is the reason why the initial problem statement needs to be rephrased into: *How can generic programming be applied to mathematical morphology in order to obtain generic, reusable, and theoretically compliant software while maintaining very good processing speed?*

A morphological software development framework that employs generic patterns must be justifiable. Morphological patterns must describe commonly recurring structure of components that accomplish a general problem within a particular context. Clearly, the word *pattern* suggests recurrence: if something does not recur, it cannot possibly be a pattern. However, recurrence is not the sole characteristic of importance. It is also needed to show that a morphological pattern is fit for use, and that it is a useful one. Recurrence is a purely quantitative characteristic. Fitness and usefulness are qualitative characteristics. The software developer must show fitness by explaining how a particular morphological pattern is successful and usefulness by explaining why it is successful and rewarding.

1.4.3 Thesis Overview

This thesis focus on the design principles of software for mathematical morphology using algorithmic patterns. Implementational details are almost absent since the thesis is much more concentrated on the general common principles of mathematical morphological computations. Chapter 2 consist of a general introduction why the use of generic programming is advantageous as the basis for a mathematical morphology library in comparison to other libraries. Chapter 3 defines the complete lattice framework in as much it is relevant to the computational patterns to appear later on. Chapter 4 describes the parallel pattern, chapter 5 deals with the sequential patterns, and chapter 6 is on the grassfire or bucket or queue-based pattern. All chapters 4, 5, and 6 have a theoretical section, an observational section, a pattern section, and examples of implementations.

This thesis is based on the following papers:


M. C. d’Ornellas, R. v.d. Boomgaard, A. W. M. Smeulders and D. Koelma. A generic programming paradigm for image processing. *In preparation*

R. v.d. Boomgaard, and M. C. d’Ornellas. Complete lattice theory: The algebraic framework for morphological image processing. *In preparation*

M. C. d’Ornellas, R. v.d. Boomgaard. A formal specification for sequential algorithms in mathematical morphology. *In preparation*

M. C. d’Ornellas, R. v.d. Boomgaard. Morphological algorithmic patterns. *In preparation*

**Bibliography**


Table 1.1 Table containing a list of morphological operators with respect to their availability for Mmach, Mmorph, Micromorph, and Visilog software packages.

<table>
<thead>
<tr>
<th>Morphological Operator</th>
<th>Mmach</th>
<th>Mmorph</th>
<th>Micromorph</th>
<th>Visilog</th>
</tr>
</thead>
<tbody>
<tr>
<td>erosion</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>dilation</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>conditional erosion</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>conditional dilation</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>distance transform</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>geodesic distance transform</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>opening</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>closing</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>center filter</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>alternate sequential filtering (ASF)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>area opening</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>area closing</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>opening by reconstruction</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>closing by reconstruction</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inf-reconstruction</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>sup-reconstruction</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>regional minima</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>regional maxima</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>valley removal</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>peak removal</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASF by reconstruction</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>minima imposition</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>hole filling</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>edge off</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>morphological gradient</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>opening top-hat</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>closing top-hat</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>open by reconstruction top-hat</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>close by reconstruction top-hat</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ultimate erosion</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>morphological skeleton</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>sup-generating (hit-miss)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>inf-generating</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>adaptive threshold</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>thinning</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>thickening</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>conditional thinning</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>conditional thickening</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>skeleton by influence zones (SKIZ)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>watersheds</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
## Table 1.2 List of prospected image data types in Horus.

<table>
<thead>
<tr>
<th>Image Data Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar integer byte 2D</td>
<td>large field images</td>
</tr>
<tr>
<td>Scalar integer short 2D</td>
<td>common grey-valued image</td>
</tr>
<tr>
<td>Scalar integer int 2D</td>
<td>intermediate results requiring more precision</td>
</tr>
<tr>
<td>Scalar real float 2D</td>
<td>large contrast operations</td>
</tr>
<tr>
<td>Scalar real double 2D</td>
<td>preferred to float as it has less arithmetic problems</td>
</tr>
<tr>
<td>Scalar integer byte 3D</td>
<td>common grey-valued image in 3D</td>
</tr>
<tr>
<td>Scalar integer short 3D</td>
<td>representation for many computations in 3D</td>
</tr>
<tr>
<td>Scalar integer int 3D</td>
<td>common representation for vector fields</td>
</tr>
<tr>
<td>Scalar real float 3D</td>
<td>common for motion field computations</td>
</tr>
<tr>
<td>Vector 2 integer byte 2D</td>
<td>large color fields</td>
</tr>
<tr>
<td>Vector 2 integer short 2D</td>
<td>some color models, e.g. RGB</td>
</tr>
<tr>
<td>Vector 2 integer int 2D</td>
<td>other color models, e.g. HSV, CMY, YIQ</td>
</tr>
<tr>
<td>Vector 2 real float 2D</td>
<td>3D color images</td>
</tr>
<tr>
<td>Vector 2 real double 2D</td>
<td>vector computation in 3D</td>
</tr>
<tr>
<td>Vector 2 integer byte 3D</td>
<td>this is useful for vector computation in 3D</td>
</tr>
<tr>
<td>Vector 2 integer short 3D</td>
<td></td>
</tr>
<tr>
<td>Vector 2 integer int 3D</td>
<td></td>
</tr>
<tr>
<td>Vector 2 real float 3D</td>
<td></td>
</tr>
<tr>
<td>Vector 2 real double 3D</td>
<td></td>
</tr>
<tr>
<td>Vector 3 integer byte 2D</td>
<td></td>
</tr>
<tr>
<td>Vector 3 integer short 2D</td>
<td></td>
</tr>
<tr>
<td>Vector 3 integer int 2D</td>
<td></td>
</tr>
<tr>
<td>Vector 3 real float 2D</td>
<td></td>
</tr>
<tr>
<td>Vector 3 real double 2D</td>
<td></td>
</tr>
<tr>
<td>Vector 3 integer byte 3D</td>
<td></td>
</tr>
<tr>
<td>Vector 3 integer short 3D</td>
<td></td>
</tr>
<tr>
<td>Vector 3 integer int 3D</td>
<td></td>
</tr>
<tr>
<td>Vector 3 real float 3D</td>
<td></td>
</tr>
<tr>
<td>Vector 3 real double 3D</td>
<td></td>
</tr>
</tbody>
</table>