A Parallel Algorithmic Pattern

Marcos C. d'Ornellas[1]
ornellas@inf.ufsm.br

Grupo de Pesquisas em Processamento de Imagens (PIGS)
Centro de Tecnologia (CT)
Universidade Federal de Santa Maria (UFSM)
Av. Roraima - Campus Universitário
97105-900 Santa Maria - RS - Brasil

Abstract

This paper briefly discuss the general class of algorithms that can be implemented using parallel constructions. Common characteristics of these algorithms are also described in order to provide a generic representation for parallel algorithms. In addition it describes the parallel pattern in terms of image scannings and its relationship within mathematical morphology. Such a pattern is essential for the development of morphological operators and operations. Examples of the application of the parallel generic pattern are given for both scalar lattices and non-scalar lattices. Scalar lattices are used to give a parallel pattern representation for real values, parabolic morphology, and b-bit integers. Non-scalar lattices are restricted to color lattices. Each case study presented in this paper match the generic representation of the parallel pattern.

1 Name

Parallel

2 Intent

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 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 occur.

A mathematical framework for the developing of image processing algorithms is mathematical morphology [22]. The available arithmetical and relational operators make the implementation of its basic operations (erosion, dilation, opening and closing) an easy task. However, when algorithmic implementations are derived from the straight mapping of morphological operators such as erosion and dilations, the computational complexity is known to be

[1]Copyright ©2003, Marcos C. d'Ornellas. Permission is granted to copy for the SugarloafPLoP 2003 Conference. All other rights are reserved.

higher since it contains nested loops. Without resorting to parallel computers, the computational complexity due to the number of pixels in an image cannot be reduced. Therefore, the only way to reduce computational complexity significantly is through the application of efficient neighborhood operations.

The parallel pattern solves a well-known algorithmic design problem in image processing, that is, how to keep a one-to-one relationship between mathematical formulation and its effective implementation. The mathematical formulation explains the way it should be constructed and how data is manipulated. This pattern acts as a first step towards the design and implementation of morphological image operators under the complete lattice framework. The generic pattern definition is suggested in order to characterize the basic building blocks given by the iterator, pixel lattice, and adjunction. The mathematical approach leads to the design of consistent methods that address all the requirements needed for the implementation of the parallel pattern.

The parallel pattern is intended for those who wish to facilitate code reuse in software projects in image processing and computer vision. This pattern gives rise to a family of algorithms, which are often used to build useful constructions and to solve a general list of problems that often ocurr. This pattern assumes the reader is reasonably proficient in image processing or mathematical morphology, and have some experience in generic programming as well.

3 Motivation

Designing software from scratch is the choice when the problem at hand requests unconventional solutions, or when the operational circumstances are different from anything seen in the past. In general, when the problem is very difficult one can imagine that everything has to be in line to reach an acceptable solution. For morphological image processing, this is no longer a good engineering practice. Implementing software this way will generate code that will be dedicated to the data, to the problem, and to the platform.

Combinations of various software design techniques such as component-based programming, object-oriented programming and generic programming have proved effective for code optimization for frequently performed operations in mathematical morphology. Due to the versatility of generic programming, one does not need to invest time in specific implementations for every variation of a general algorithm. This means that a small number of generic algorithms are sufficient to implement the functionality of a traditional image processing library.

The straight mapping of erosions and dilations in the pixel and image lattices into algorithm representations are the necessary requirements to introduce the generic pattern in this paper. Such pattern must be fully characterized by its basic building blocks. A specialization of this pattern, namely parallel pattern, is used to describe parallel algorithm implementations. Other specializations of the generic pattern are described in [5].

4 Applicability

This paper proposes a generic model for realizing and using parallel design patterns in mathematical morphology. The term parallel is used to represent the application independent, reusable set of attributes associated with a pattern. It turns out that the model is an ideal candidate for object-oriented style of design and implementation. It can be implemented in `C++` with the Sandard Template Library (`STL`) without requiring any language extension. The generic model, together with the object-oriented and library-based approach, facilitates extensibility.

Algorithms that use a parallel pattern share a certain number of characteristics, which are listed in the following:

- **Fast Pixel Manipulation**: in the lowest level macro of a function implementation all the pixel manipulation is performed with pointers. Typically there is at least one pointer to source image(s) and a pointer to a destination image;
- **Nested Loops**: for a rectangular image the method used to traverse the image data depends on the operation being applied. For an operation using x and y coordinates, two nested loops are used with the pointers being incremented in the inner loop. For a neighbourhood operation, surrounding pixel data is accessed by adding and subtracting offsets from the pointers. An offset of 1 allows data one pixel to the left and right to be accessed and an offset of width allows data one pixel above and below to be accessed. This easily extends for larger neighbourhoods;
- **Image Shape**: for an arbitrarily shaped image there is a boundary structure associated with the image's image size window. The actual data stored corresponds to the rectangular bounding area which encloses the shape of the image. There is a some space redundancy here, as more data is stored than strictly required but this is offset by the fact that the simple pointer manipulations just described can still be used for arbitrarily shaped images, leading to simpler, faster and more robust code. The only difference need be in the loops which control the traversal of the data. These loops consist of an outer loop (to follow the linked list in the boundary structure) and an inner loop to move along each horizontal line segment. Treating the image as rectangular allows the same code to be used for all cases. The cost of processing the extra pixels needs to be weighed against the extra complexity of the traversal loops. This depends on the cost of processing an individual pixel and the particular shape of the image;
- **Origin Included in the Support**: algorithms assume that the origin is included in the structuring element support, which is important because only anti-extensive erosions and extensive dilations are considered.

5 Structure

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 [23]. 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] [12] [16] [20] [21].

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 [14] [15] to watersheds [2].

Consider the most general structure of erosion and dilation for images $f \in \mathrm{Fun}\,(\mathbb{E}, \mathcal{L})$, where $\mathbb{E}$ is a discrete space and $\mathcal{L}$ is the complete pixel lattice. The basic morphological operators are erosion and dilation written as:

$$(\varepsilon f)(x) = \bigwedge_{y \in \mathbb{E}} \varepsilon(x, y)(f(y)) \tag{1}$$

and

$$(\delta f)(x) = \bigvee_{y \in \mathbb{E}} \delta(x, y)(f(y)) \tag{2}$$

where $\varepsilon(y, x)$ and $\delta(x, y)$ are erosions and dilations in the pixel lattice $\mathcal{L}$. Equations 1 and 2 form an adjunction on $\mathcal{L}$.

A pixel lattice $\mathcal{L}$ is characterized by the value set V (the collection of all possible pixel values) and the partial ordering on V (denoted $\leq$). It is explicitly described by the tuple: $\mathcal{L} = (V, \leq, \bigvee, \bigwedge, \bigvee_{\mathcal{L}}, \bigwedge_{\mathcal{L}})$, where $\bigvee$ and $\bigwedge$ are the supremum (least upper bound) and infimum (greatest lower bound) operators respectively. Observe that this lattice includes additional elements other than V and $\leq$ since they are needed in the implementation of morphological image operators. In the discrete case, only the supremum and infimum of a finite number of values are needed. Then, $\bigvee$ and $\bigwedge$ can be rightfully called maximum and minimum respectively. $\bigvee_{\mathcal{L}}$ and $\bigwedge_{\mathcal{L}}$ are also needed as the lattice supremum and infimum.

A straightforward implementation of equations 1 and 2 is called parallel because the order in which the locations $x \in \mathbb{E}$ are being accessed has no influence on the result. The same algorithm can be implemented in a sequential hardware. It can also run on parallel hardware where every processor is responsible for processing a part of the image [17].

Consider, for instance, the dilation in the image lattice:

$$(\delta f)(x) = \bigvee_{y \in \mathbb{E}} \delta(x, y)(f(y)) \tag{3}$$

where $f \in \mathrm{Fun}\,(\mathbb{E}, \mathcal{L})$, and $\mathcal{L} = (V, \leq, \bigvee, \bigwedge, \bigvee_{\mathcal{L}}, \bigwedge_{\mathcal{L}})$. The domain $\mathbb{E}$ of the images in this paper are rectangular subsets of $\mathbb{Z}^d$ where d is the dimension of the image f. Note that the pixel lattice $\mathcal{L}$ is a complete lattice, i.e. a partial ordered set (*poset*) with extra requirements including a supremum and an infimum.

The pixel lattice dilation $\delta(x, y)$ is parameterized by two locations in the image domain $\mathbb{E}$. In words, equation 3 dictates that at every location x, $(\delta f)(x)$ is calculated by looping over all locations $y \in \mathbb{E}$ and dilating $f(y)$ with the pixel lattice dilation $\delta(x, y)$ resulting in $\delta(x, y)(f(y))$. Then the supremum of all these values is taken as the result. An algorithmic representation for $h = \delta f$ is given in figure 1.

6 Participants

The parallel pattern includes the following elements:

- **iterator**: an iterator is a fundamental structure that abstracts the process of moving through a finite set of elements. It allows for the selection of each element of the set without knowing the underlying structure of the set. Using a programming language's

Figure 1 Parallel dilation algorithm representation demonstrating the straight mapping from its mathematical equation.

```
for all ($x \in \mathbb{E}$) do
  $h(x) = \bigwedge_{\mathcal{L}}$
  for all ($y \in \mathbb{E}$) do
    $h(x) = \bigvee\{h(x), \delta(x, y)(f(y))\}$
  end for
end for
```

terminology, an iterator can be considered as an abstract pointer to an element in the set. Algorithms use iterators to operate on data structures (containers). Iterators set bounds for algorithms, regarding the extent of the container. This is a powerful feature, partly because it allows for learning a single interface that works with all containers, and partly because it allows containers to be used interchangeably.

Iterators are more complex at the implementation level. Therefore, there is a need for a generalization of the iterator concept for two or more dimensions. That is, traversal functions must be provided to tell the iterator in which coordinate direction to move. For instance, the algorithm in figure 1 would be better characterized with a two-dimensional iterator to move through all the elements $x \in \mathbb{E}$ and another to move through all elements $y \in \mathbb{E}$. Since images are bounded and have a finite number of elements, algorithm characterizations using one-dimensional iterators are also feasible;

- **pixel lattice**: The mathematical theory states that only two elements are needed to describe complete lattices, i.e. the value set V and the partial ordering relation $\leq$, yielding $\mathcal{L} = (V, \leq)$. Note, however, that a decision was made to have all the operators and constants explicitly available. Therefore, the supremum $\bigvee$ and infimum $\bigwedge$ operator as well as the lattice supremum $\bigvee_{\mathcal{L}}$ and infimum $\bigwedge_{\mathcal{L}}$ are included in the lattice definition $\mathcal{L} = (V, \leq, \bigvee, \bigwedge, \bigvee_{\mathcal{L}}, \bigwedge_{\mathcal{L}})$.

  It is the responsibility of the software developer (who instantiates a particular pixel lattice) to make these operators and constants explicitly. For example, the algorithm in figure 1 needs the appropriate values for the pixel lattice infimum $\bigwedge_{\mathcal{L}}$ and the supremum operator $\bigvee$;

- **adjunction**: the image dilation $(\delta f)(x)$ included in the algorithm in figure 1 is characterized by the pixel lattice dilation $\delta(x, y)$. With any pixel lattice dilation, a unique pixel lattice erosion is associated. Such a pair of adjoint operators is called an adjunction $\mathcal{A} = (\varepsilon, \delta)$.

These three elements, when joined together, form the building block representation of the generic parallel pattern. The generic implementation of the algorithm in figure 1 is really all that is needed to implement morphological operators on $\mathrm{Fun}(\mathbb{Z}^d, V)$. If an image operator does not fit into this scheme, it is not placed into mathematical morphology theory.

7 Collaborations

- **Iterator** is coupled with pixel scanning methods acting on containers (e.g. vectors). All data are accessed in parallel by one-dimensional iterators for every pixel scan. Pixels scans might differ from each application even though they must produce the same results;

- **Pixel Lattice** defines the value set used for the data involved. In other words, it determines a set of common rules to be used in the parallel pattern for a family of data types;

- **Adjunction** is closely related with the way pixel scannings are conducted. Therefore, it works together with the **iterator**.

8 Consequences

- The applicability of a general theory for mathematical morphology, well-established in the complete lattice framework, is needed for the construction of the generic pattern for morphological algorithm representations. The generic pattern is made of the three basic building blocks namely the iterator, the pixel lattice and the adjunction and was applied in the effective implementation of algorithms and applications. It is known that many algorithms in the literature are all reduced to the parallel pattern.

- The implementation of generic algorithms for morphological image operators still need a nested loop `for` construction to process the image data by means of pixel scans, a typical characteristic of the parallel pattern. Consider the nested loop within the dilation algorithm. If the time to execute $h(x)$ is independent of any indexing variable, the complexity is $\mathcal{O}(\#N_\delta)$, $\#N_\delta$ being the number of pixels in N_δ respectively. Due to the performance penalty its high complexity produces, this approach requires additional strategies to reduce such complexity.

- Flat operators are defined in such a way that pixel lattice erosions and dilations are not taken into account. They play an important role in mathematical morphology for its theoretical aspects and practical use. Therefore, flat morphology introduces changes in the representation of the parallel pattern with respect to the adjunction since only the minimum and maximum are needed to characterize erosions and dilations in the image lattice over a specified neighborhood. Flat morphology works with structuring elements of all shapes and sizes.

- A generic programming approach might be applied to morphological image operators based on pixels scans like thinnings and thickenings, parabolic morphology and skeletons. Generic programming tools as `C++` with `STL` applies a more evolutionary and experimental approach to morphological algorithm development. The proposed parallel pattern enhances morphological algorithm reuse.

- Morphological operators are often used with large structuring elements, which makes them highly inefficient for practical purposes. Alternative solutions for the lack of speed of flat morphological operators and applications derived from the parallel pattern employ structuring element decomposition. Other fast algorithms for flat morphology based on bitmapped representations and linear structuring element decompositions uses van Herk's algorithm.

9 Implementation

Morphological operator design must comply with the complete lattice framework theory, i.e. algorithmic implementations must be tied to the generic pattern representation that includes the iterator, the pixel lattice, and the adjunction. Iterators in the generic representation for parallel algorithms are not influenced by the containers since they do not influence or change the final result, which must be the same for all set of one-dimensional or two-dimensional iterators. Later, the iterator is combined with the pixel lattice and the adjunction, producing the parallel pattern.

The following implementation issues are relevant for the parallel pattern:

Containers: Vectors are often used in the implementation of algorithms that are governed by n-dimensional pixel scans. Items stored in a vector structure are pixel addresses or pixel values, but the vector must be able to handle other features according to the implementation. The size of the vector can be fixed or controlled dynamically.

`C++` **and** `STL`**:** The STL is a relatively small library which achieves a remarkable degree of reuse through its basis in the principles of generic programming and its use of `C++` templates. Because of this, it has a particularly clear shape. The distinction between containers, iterators and algorithms is its most striking structural feature: dynamically, the way a container delivers iterators which are then used by algorithms is a consistent and fundamental pattern of use.

10 Algorithm Samples and Usage

Applications of the parallel pattern are widely used in image processing. Examples are given in this section with respect to scalar lattices and non-scalar lattices. Such applications are grounded on the theoretical pattern concepts and pattern description and can be easily seen as a natural mapping from morphological formulas into algorithm representation. Examples of the parallel pattern are expressed in terms of iterator, pixel lattice, and adjunction. These examples are restricted to changes in the generic pattern with respect to pixel lattice and adjunction. Additional attention is given to color lattices in the sense that it is still not a completely solved concept in mathematical morphology.

Real Values: Real valued images $f \in \mathrm{Fun}(\mathbb{E}, \mathbb{R})$ are considered in this section. The definition of an erosion in the image lattice is given by:

$$(\varepsilon f)(x) = \bigwedge_{y \in \mathbb{E}} \varepsilon(x, y)(f(y)) \tag{4}$$

where $\varepsilon(x, y)$ is the erosion in the pixel lattice $\mathcal{L}$.

A translation invariant definition is derived from equation 4, yielding:

$$(\varepsilon f)(x) = \bigwedge_{y \in \mathbb{E}} \varepsilon_t(y - x)(f(y)) \tag{5}$$

where $\varepsilon_t(y-x)(f(y)) = f(y) - g(y-x)$, where g denotes the structuring element. Such definition leads to the classical structural erosion given by:

$$(\varepsilon f)(x) = \bigwedge_{y \in \mathbb{E}} \{f(y) - g(y-x)\} \tag{6}$$

If the definition in equation 6 is used for finite image domains, the border effect only shows up in case one implements g as a function $g : \mathbb{E} \to V$ as well. The confusion comes from the fact that the notion of translation of images (through $g(y-x)$) is implicitly defined, which is only valid on infinite domains.

The classical structural erosion described by equation 6 can be seen as a special function lattice since it has a purely geometrical interpretation [6] [7] [10] [24]. This interpretation matches the topographical view for a two-dimensional Euclidean space, where points are given by triples of coordinates; the first two locate the position in the two-dimensional support set and the third coordinate gives the height.

The parallel pattern for the real values is presented as follows:

- **iterator**: parallel algorithms using iterators (e.g. `for` loops) are the most classical not only in conventional image processing but also in mathematical morphology. An algorithm is said to be parallel if the pixel values in the neighborhood $N(x)$ are taken in the original image. Implementations derived from this pattern are independent of the order of image scannings;
- **pixel lattice**: all the elements must be explicitly defined by $\mathcal{L} = (\mathbb{R}, \leq, \bigvee, \bigwedge, \bigvee_{\mathcal{L}}, \bigwedge_{\mathcal{L}})$.
- **adjunction**: the adjunction $\mathcal{A} = (\varepsilon, \delta, N_\varepsilon, N_\delta)$ is given by the classical definitions of structural erosions and dilations:

$$\begin{aligned} (\varepsilon f)(x) &= \bigwedge_{y \in N_\varepsilon(x)} \{f(y) - g(y-x)\} && (7) \\ (\delta f)(x) &= \bigvee_{y \in N_\delta(x)} \{f(y) + g(x-y)\} && (8) \end{aligned}$$

  where $N_\varepsilon(x) = \{y \mid g(-y) \neq -\infty\}$ and $N_\delta(x) = \{y \mid g(y) \neq +\infty\}$.

Parabolic Morphology: Consider A to be a $n \times n$ symmetric positive definite matrix. The parabolic or quadratic structuring function (`QSF`) associated with this matrix is denoted as q_A and is given by $q_A(x) = -\frac{1}{2}\langle x, A^{-1}x\rangle$. Note that the simplest `QSF` is the rotationally symmetric one: q_I with I being the identity matrix ($q_I(x) = -\frac{1}{2} \parallel x \parallel^2$).

If two `QSF's` q_A and q_B are dilated, then $q_A \oplus q_B = q_{A+B}$, which implies that the class of `QSF's` is closed under dilation. A `QSF` q_A can be dimensionally decomposed along the eigenvectors of the matrix A. For a diagonal matrix, the eigenvectors are along the x and y axis and thus the one-dimensional dilations are along the rows and columns. Note that the `QSF` q_I is the unique rotational invariant function that can be dimensionally decomposed with respect to dilation [2] [25][18] [26].

[2]This is a property that it shares with the Gaussian function being the unique isotropic kernel that can be dimensionally decomposed with respect to convolution.

The erosion scale function $F^{\ominus}(x,\rho)$ is obtained by eroding the original image f with structuring function q^{ρ} : $F^{\ominus}(x,\rho) = (f \ominus q^{\rho})(x)$. Dually, the dilation scale function $F^{\oplus}(x,\rho)$ is obtained by dilating the original image f with structuring function q^{ρ} : $F^{\oplus}(x,\rho) = (f \oplus q^{\rho})(x)$. For increasing values of ρ the scale in the resulting image, (either $F^{\ominus}$ or $F^{\oplus}$) decreases as the original image f is processed with a structuring function with increasing size. Therefore, $F^{\ominus}$ and $F^{\oplus}$ are two sequences of images, each derived from the original image and ordered with respect to their internal scale. A parabolic scale-space was proposed in [26] based on those scaling properties.

The parallel pattern for parabolic morphology is the one presented as follows:

- **iterator**: the iterator is the same for the real values (using `for` loops) and does not need to be changed in this case;
- **pixel lattice**: all the elements must be explicitly defined by $\mathcal{L} = (\mathbb{R}, \leq, \bigvee, \bigwedge, \bigvee_{\mathcal{L}}, \bigwedge_{\mathcal{L}})$;
- **adjunction**: the classical adjunction, defined by $\mathcal{A} = (\varepsilon, \delta,$ is extended to $\mathcal{A} = (\varepsilon, \delta, \mathbb{E})$, since $N_{\varepsilon} = N_{\delta} = \mathbb{E}$. Parabolic erosions and dilations are governed by the equations:

$$F^{\ominus}(x,\rho) = (f \ominus q^{\rho})(x) \quad (9)$$
$$F^{\oplus}(x,\rho) = (f \oplus q^{\rho})(x) \quad (10)$$

  where $q(x) = -\frac{1}{4} \parallel x \parallel^2$, a unique structuring function that can be separated by dimension.

b-bit Integers: A b-bit integer is the term used to express that an image can be mapped using a byte representation (e.g. 8-bits for a grey-scale image). Consider the set $\mathcal{S}$ being one of $\mathbb{N}, \mathbb{Z}, \mathbb{R}, \ldots$ and assume that l_{inf} and l_{sup} are the lattice infimum and lattice supremum respectively. If the set $\mathbb{N}$ is assigned to $\mathcal{S}$, one obtains a finite set with values between $[0,\ldots,K], K \in \mathbb{N}$ which is not closed under addition and subtraction. Therefore, by approaching this problem using truncated values between $l_{inf} = 0$ and $l_{sup} = K$, the adjunction property is lost.

In order to solve this problem, a new formulation for erosions and dilations for grey-scale images was introduced in [11] [12] [21]:

$$(\varepsilon f)(x) = \bigwedge_{y \in N_{\varepsilon}(x)} \{f(y) \dot{-} g(y-x)\} \quad (11)$$
$$(\delta f)(x) = \bigvee_{y \in N_{\delta}(x)} \{f(y) \dot{+} g(x-y)\} \quad (12)$$

The dot operators $\dot{-}$ and $\dot{+}$ are given by:

$$a \dot{-} b = \begin{cases} K & \text{if } a = K; \\ K & \text{if } (a < K \ \textit{and} \ a - b > K); \\ 0 & \text{if } (a < K \ \textit{and} \ a - b \leq 0); \\ a - b & \text{if } (a < K \ \textit{and} \ 0 \leq a - b \leq K). \end{cases} \quad (13)$$

$$a \dot{+} b = \begin{cases} 0 & \text{if } a = 0; \\ 0 & \text{if } (a > 0 \ \mathit{and} \ a + b \leq 0); \\ K & \text{if } (a > 0 \ \mathit{and} \ a + b > K); \\ a + b & \text{if } (a > 0 \ \mathit{and} \ 0 \leq a + b \leq K). \end{cases} \tag{14}$$

where $a \in [0, \dots, K]$ and $b \in \mathbb{R}$.

The parallel pattern for the b-bit integers is the one presented as follows:

- **iterator**: the iterator is just like the same for the real values (using `for` loops) and does not need to be changed in this case;
- **pixel lattice**: all the elements must be explicitly defined by $\mathcal{L} = (V, \leq, \bigvee, \bigwedge, \bigvee_{\mathcal{L}}, \bigwedge_{\mathcal{L}})$. For instance, for binary and grey-scale images:

$$\mathcal{L} = ([0,1], \leq, \bigvee, \bigwedge, 1, 0)$$
$$\mathcal{L} = ([0,255], \leq, \bigvee, \bigwedge, 255, 0)$$

  Note that other integer intervals can be used to compose a new pixel lattice;
- **adjunction**: the adjunction $\mathcal{A} = (\varepsilon, \delta, N_\varepsilon, N_\delta)$ follows the classical definition except for the fact that erosions and dilations are governed by equations 11 and 12 respectively.

Color Lattices: In color morphology, operators that are applied to the whole image can also be applied to the components separately, because the filters commute with infimum and supremum respectively. This kind of marginal processing is equivalent to the vectorial approach defined by the canonic lattice structure when only supremum and infimum operators and their compositions are involved and induces a totally ordered lattice as presented in [8] [9]. Consider two images f and f', each one made up by a number of i components. Therefore:

$$f \leq f' \iff f(i) \leq f'(i), \forall i \in 1, \dots, m$$

With these relations, the supremum of a family $\{f_j\}$ is the vector $\bigvee f$ where each component $\bigvee f(i)$ is the supremum of the $\{f_j(i)\}$. Respectively, the infimum of a family $\{f_j\}$ is the vector $\bigwedge f$ where each component $\bigwedge f(i)$ is the infimum of the $\{f_j(i)\}$. However, this morphological procedure fails because every color can be seen as a vector, which cannot be totally ordered and so the supremum or infimum of the two is a mixture of both the colors. Using this procedure, one obtains the same results as the simple marginal processing of the data and new colors not contained in the input image will appear.

Another approach, grounded on the use of a vector transformation from $\mathbb{R}^m$ into $\mathbb{R}^Q$ followed by a marginal ordering on $\mathbb{R}^Q$ was defined in [8]. If $Q > 1$, the marginal ordering induces a partial ordering on the vectors. $Q = 1$ is required to obtain a total ordering with an h-adjunction. The important point is to transform the image data by means of a surjective transformation h, which is better suited for the morphological approach. However, h is neither bijective nor injective. A major drawback in practice is that the extrema of a family $\{f_j\}$ are not necessarily unique. Therefore, many different vectors can lead to the same result $h(f) = \bigwedge_i \{f_i\}$ or $h(f) = \bigvee_i \{f_i\}$ for erosions and dilations respectively. Consequently, there is a need for a new equivalence relation $=_h$.

To extend the vector approach to color images, it is necessary to define an order relation who orders the colors as vectors, rather than ordering the individual components as suggested in [13]. This can be done using reduced ordering. This kind of ordering imposes a total ordering relationship that can be accomplished by the lexicographical ordering [4] [3] as reported in [28]. The flat structuring element for the vector morphological operations defined here is the set g, and the scalar-valued function used for the reduced ordering is $r : \mathbb{R}^3 \rightarrow \mathbb{R}$. Erosions and dilations are given by:

$$\begin{aligned} (\varepsilon f)_r(x) &= \bigwedge_{y \in N_\varepsilon(x)} \{f_r(y) - g_r(y - x)\} && (15) \\ (\delta f)_r(x) &= \bigvee_{y \in N_\delta(x)} \{f_r(y) + g_r(x - y)\} && (16) \end{aligned}$$

where f_r and g_r are the coded representations for f and g under the reduced ordering r.

Erosions and dilations in conjunction with a total ordering induced by a reduced ordering generate no new colors. Under the total ordering relation, the infimum or supremum is one of the actual colors. Therefore, the only colors in the output image are those obtained from translations of the input ones.

Since the output of the vector filter depends on the scalar-valued function used for reduced ordering, the selection of this function provides flexibility in incorporating spectral information into the multivalued image representation. For example, linear combinations of the tristimulus values can be used. This can be written as:

$$t \leq t' \leftrightarrow \begin{cases} r(t) = a_1 t_1 + a_2 t_2 + a_3 t_3 \\ r(t') = a_1 t'_1 + a_2 t'_2 + a_3 t'_3 \\ r(t) \leq r(t') \end{cases} \qquad (17)$$

if the image is filtered in the `RGB` color model. For the case where $a_1 = 0.299, a_2 = 0.587$, and $a_3 = 0.114$, $r(t)$ and $r(t')$ become the luminance component. Multiscale opening in this case would suppress bright objects at each scale. The values of a_1, a_2, and a_3 can also be selected to enhance or suppress specific colors. For example, if $a_1 = 1, a_2 = 0$, and $a_3 = 0$, then the effect of a multiscale opening would be to suppress objects with high `red` content. The same holds to the `green` and `blue` color channels.

The parallel pattern for the non-scalar lattice working on color images is given as follows:

- **iterator**: just like for all the examples in this paper, the iterator (e.g. `for` loops) structure does not change;
- **pixel lattice**: all the elements must be explicitly defined by $\mathcal{L} = (V, \leq, \bigvee, \bigwedge, \bigvee_\mathcal{L}, \bigwedge_\mathcal{L})$. For instance, for a `RGB` color image:

$$\mathcal{L} = (([0, 255], [0, 255], [0, 255]), \leq, \bigvee, \bigwedge, [255, 255, 255], [0, 0, 0])$$

[3] An ordered pair (i, j) is lexicographically earlier than (i', j') if either $i \leq i'$ or $i = i'$ and $j \leq j'$. It is lexicographic because it corresponds to the dictionary ordering of two-letter words.

Observe that the partial ordering $\leq$ can vary in accordance with the scalar valued functions introduced previously. Indeed, any other partial ordering can be used, leading to different output results;

- **adjunction**: the adjunction is represented by $\mathcal{A} = (N)$ since color operators are extensions of flat operators using a flat and symmetric structuring element. Erosions and dilations are given by:

$$(\varepsilon f)_r(x) = \bigwedge_{y \in N_\varepsilon(x)} \{f_r(y)\} \tag{18}$$

$$(\delta f)_r(x) = \bigvee_{y \in N_\delta(x)} \{f_r(y)\} \tag{19}$$

  These equations are vectorial representations for flat structural erosion and dilation of an image f with r components.

Some results derived from the application of the parallel pattern in color erosions and dilations are provided in figures 2 and 3 respectively. Take notice the results for two ordering relationships: bitmix and maximum.

Figure 2 RGB Erosion showing the results for a flat 5×5 structuring element: input image(a), bitmix(b), and maximum(c).

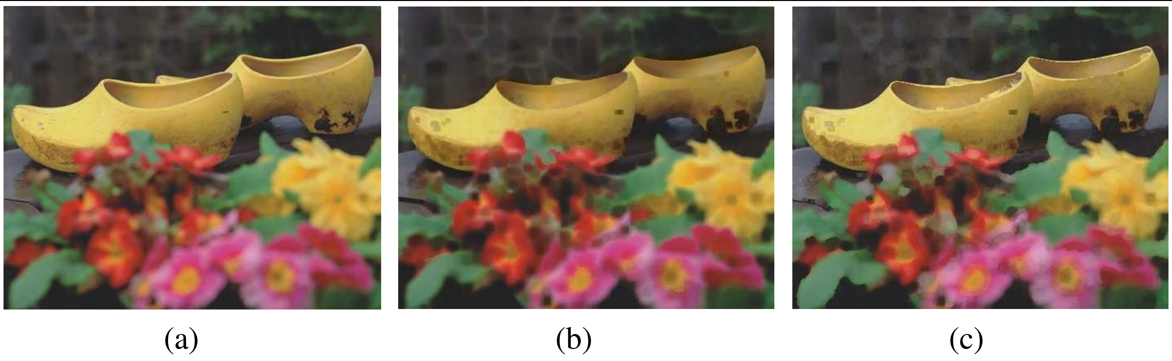
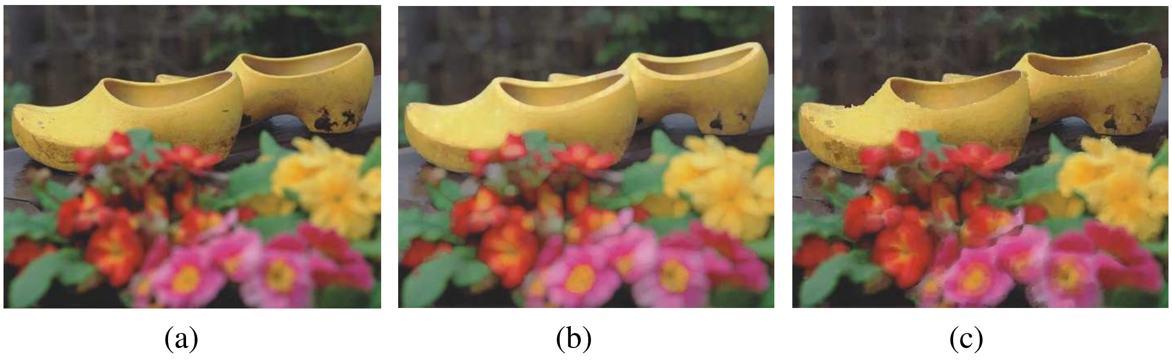


Figure 3 RGB dilation showing the results for a flat 5×5 structuring element: input image(a), bitmix(b), and maximum(c).



11 Known Uses

Several parallel operators and operations have been proposed in the literature leading to a large collection of applications, which are characterized as follows:

fast algorithms for flat morphology: A fast algorithmic implementation for basic binary morphological operations on general-purpose sequential computers was proposed in [27] and works with structuring elements of arbitrary size and shape. Rather than representing binary images as bit-planes inserted in grey-scale images, the bitmap representation was used. This representation is very efficient both in terms of memory requirements and in terms of algorithmic efficiency because the CPU operates on 32 pixels in parallel. A combination of a bitmapped representation with the logarithmic decomposition of structuring elements leads to very fast algorithms for the basic morphological operators;

parabolic morphology: The morphological scale-space is generated by dilations (erosions) with a parabolic structuring function of increasing width. Most often, extending the scalar parameter to the entire real axis combines the erosion and dilation scale spaces. Negative scales are interpreted as the erosion scales (background) and positive scales as the dilation scales (foreground). Foreground and background are treated distinctly, which make a great difference from linear theory since convolution cannot have this as it is a self-dual operator [25];

color morphology: To extend the vector approach to color images, it is necessary to define an order relation who orders the colors as vectors, rather than ordering the individual components as suggested in [13]. This can be done using reduced ordering. This kind of ordering imposes a total ordering relationship that can be accomplished by the lexicographical ordering [4] Erosions and dilations in conjunction with a total ordering induced by a reduced ordering generate no new colors;

thinnings: Thinning is the method used to remove selected foreground pixels from a binary image. It is generally used after an edge detection operation to tidy up by reducing all the lines to one pixel thickness. The operation is quite similar to the Hit-or-miss operation. The structuring element is moved over the input image pixels and if the foreground and background pixels of the structuring element and the image are the same then the input pixel below the origin of the structuring element is set to the background pixel value. Otherwise it is left unchanged [3];

thickenings: Thickening is the operation of adding to an image pixels with the same configuration. It really grows certain pixels of the foreground. The structural element is superimposed on the input image and if the foreground and background pixels of the structuring element match that of the image, the input pixel is set to the foreground pixel value. Otherwise it is left unchanged. Thickening is the dual of thinning [3];

skeletons: Shape representation is an important image analysis task which can be used for contour coding and feature extraction. The morphological skeleton is a geometrical shape description by means of maximal inscribed structuring elements. The form of the structuring element is usually chosen a priori, and such process can be implemented using a parallel pattern.It provides improved progressive contour transmission and the extraction of shape features [19];

polygon filling: Polygon filling is often executed in the image or frame buffer. It is assumed that a polygon with proper closed borders are given and that inside the polygon no pixel has the color value with which it is to be filled. For the seed fill algorithm, we need a starting point which is inside the polygon. Starting at this seed position, the algorithm proceeds in a way determined by the pixel scans and sets each pixel to the required fill color until the border is reached;

robot planning: Research in robot planning has considered the interleaving of planning and execution for example, as well as the issue of planning sensor actions as a way of gaining information for planning. A major theme in robot planning is also the integration of predictive and reactive planning as a way of making activity more resilient in the face of a changing environment. Predictive planning often applies the parallel pattern in the configuration space using a visibility graph and through free space decomposition techniques.

12 Related Patterns

Sequential and Queue-based Patterns[5] are also used in mathematical morphology and in image processing. Although these patterns are more eficient than the parallel pattern, the parallel pattern itself is essential for several applications ranging from parallelized erosions and dilations to thinnings, thickenings and skeletons. A parallel pattern needs to be used when there is a chance to perform several operations at the same time and all results does not impact or produce changes in the final output. Morphological operations, which involves the rotation of the structuring element are known to be time consuming but ar implemented using the parallel pattern.

References

[1] G. J. F. Banon. Formal introduction to image processing. Research report INPE-7682-PUD/43, Instituto Nacional de Pesquisas Espaciais - INPE, São José dos Campos, 2000.

[2] S. Beucher and F. Meyer. The morphological approach to segmentation: the watershed transformation. In E. R. Dougherty, editor, *Mathematical Morphology in Image Processing*, chapter 12, pages 433–481. Marcel Dekker, New York, 1993.

[3] J. Brown and A. Hoger. A morphological point thinning algorithm. *Pattern Recognition Letters*, 17:197–207, 1996.

[4] M. L. Comer and E. J. Delp. An empirical study of morphological operators in color image enhancement. *Proceedings of the SPIE Conference on Image Processing Algorithms and Techniques III, February 10-13, 1992, San Jose, California*, 1657:314–325, 1992.

[5] M. C. d'Ornellas. *Algorithmic Patterns for Morphological Image Processing*. PhD thesis, Univesiteit van Amsterdam, 2001.

[6] E. R. Dougherty. Euclidean grayscale granulometries: Representation and umbra inducement. *Journal of Mathematical Imaging and Vision*, 1(1):7–21, 1992.

[7] E. R. Dougherty and C. R. Giardina. Morphology on umbra matrices. *International Journal of Pattern Recognition and Artificial Intelligence*, 2:367–385, 1988.

[8] J. Goutsias, H. J. A. M. Heijmans, and K. Sivakumar. Morphological operators for image sequences. *Computer Vision and Image Understanding*, 62:326–346, 1995.

[9] C. Gu. *Multivalued Morphology and Segmentation-based Coding*. PhD thesis, Ecole Polytechnique Federale de Laussane - EPFL, Lausanne, Switzerland, 1995.

[10] H. J. A. M. Heijmans. A note on the umbra transform in gray-scale morphology. *Pattern Recognition Letters*, 14:877–881, 1993.

[11] H. J. A. M. Heijmans. Mathematical morphology: Basic principles. In *Proceedings of Summer School on Morphological Image and Signal Processing*, Zakopane, Poland, 1995.

[12] H. J. A. M. Heijmans and C. Ronse. The algebraic basis of mathematical morphology – part I: Dilations and erosions. *Computer Vision, Graphics and Image Processing*, 50:245–295, 1990.

[13] R. Jones and H. Talbot. Morphological filtering for color images with no new colors. *IVCNZ'96 (Image and Vision Computing New Zealand)*, pages 149–154, 1996.

[14] P. Maragos and R. W. Schafer. Morphological filters – part I: Their set-theoretic analysis and relations to linear shift-invariant filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35:1153–1169, 1987.

[15] P. Maragos and R. W. Schafer. Morphological filters – part II: Their relations to median, order-statistics, and stack filters. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35:1170–1184, 1987.

[16] G. Matheron. Filters and lattices. In J. Serra, editor, *Image Analysis and Mathematical Morphology, Volume II: Theoretical Advances*, chapter 6, pages 35–43. Academic Press, London, 1988.

[17] W. K. Pratt. *Digital Image Processing*. John Wiley and Sons, New York, 1991.

[18] S. Makram-Ebeid R. v. d. Boomgaard, L. Dorst and J. Schavemaker. Quadratic structuring functions in mathematical morphology. In R. W. Schafer P. Maragos and M. A. Butt (Eds.), editors, *Proceedings of the ISMM'96 Workshop Mathematical Morphology and Its Applications to Image and Signal Processing*, pages 147–154. Kluwer Academic Publishers, 1996.

[19] J. M. Reinhardt and W. E. Higgins. Comparison between the morphological skeleton and morphological shape decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(9):951–956, 1996.

[20] C. Ronse. Why mathematical morphology needs complete lattices? *Signal Processing*, 21:129–154, 1990.

[21] C. Ronse and H. J. A. M. Heijmans. The algebraic basis of mathematical morphology – part II: Openings and closings. *Computer Vision, Graphics and Image Processing: Image Understanding*, 54:74–97, 1991.

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

[23] J. Serra, editor. *Image Analysis and Mathematical Morphology. II: Theoretical Advances*. Academic Press, London, 1988.

[24] S. R. Sternberg. Grayscale morphology. *Computer Vision, Graphics and Image Processing*, 35:333–355, 1986.

[25] R. v. d. Boomgaard. *Mathematical Morphology: Extensions Towards Computer Vision*. PhD thesis, University of Amsterdam, 1992.

[26] R. v. d. Boomgaard and A. W. M. Smeulders. The morphological structure of images: the differential equations of morphological scale space. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:1101–1113, 1994.

[27] R. v. d. Boomgaard and R. v. Balen. Methods for fast morphological image transforms using bitmapped binary images. *Computer Vision, Graphics and Image Processing: Graphical Models and Image Processing*, 54(3):252–258, May 1992.

[28] D. Wood. *Data Structures, Algorithms, and Performance*. Addison Wesley Publishing Company, New York, 1993.