A Constant-time Algorithm for Erosions/Dilations with Applications to Morphological Texture Feature Computation

A computationally efficient algorithm for computing erosions and dilations by one-dimensional grayscale structuring elements with constant slope is proposed. The computational complexity of this algorithm is independent of the size of the support of the structuring function. This is a generalization of the method proposed by Van Herk for the case of erosion and dilation by flat one-dimensional structuring elements. By appropriate combinations of these structuring elements, it is possible to approximate many useful structuring elements. This enables efficient computation of granulometries where the number of operations depends linearly on the number of openings. Theoretical and experimental results comparing the complexity of this algorithm with other standard techniques is presented. Two memory efficient algorithms are then presented. Several implementation issues in computing a granulometry and moments of the associated morphological pattern spectrum are then addressed. An efficient implementation of granulometries for large images on machines with limited memory, by dividing the image into smaller rectangular patches is then discussed. The optimum size of these patches is a function of the specific hardware and has been obtained experimentally for three different hardware platforms. Finally, parallel implementation of the different algorithms on two multi-processor machines is discussed.

© 2000 Academic Press

K. Sivakumar1, M. J. Patel2, N. Kehtarnavaz2, Y. Balagurunathan2 and E. R. Dougherty2

1School of Electrical Engineering and Computer Science, Washington State University, P.O. Box 642752, Pullman, WA 99164-2752, USA
E-mail: siva@eecs.wsu.edu

2Computer-Assisted Medical Diagnostic Imaging (CAMDI) Laboratory, Department of Electrical Engineering and Texas Center for Applied Technology, Texas A&M University, College Station, TX 77843-3128 USA

Introduction

Texture is an important attribute which is widely used in various image analysis applications. Objects of interest usually exhibit different texture than their surroundings. Therefore, texture features have been used as an initial step for image segmentation and classification. There is no universally agreed upon definition for image texture and different attributes have been used to quantify it, depending upon the application [1]. Most popular texture features are based on co-occurrence matrices [2–4], wavelet coefficients [5], and morphological
granulometries [6,7]. Morphological granulometry is a multi-resolution tool for image analysis that is based on the popular geometric approach to image analysis called mathematical morphology [8,9]. Morphological granulometries were originally developed by Matheron for binary images, which was later extended to grayscale images as a tool for texture analysis by Chen et al. [7,10].

A major concern in using morphological granulometric texture features is the large amount of computations required to obtain a granulometry. This is particularly true when the structuring element used for the granulometry does not have a simple geometry. The problem is further compounded when we have to analyse large images, which is usually the case in medical applications (e.g. mammograms). It is therefore imperative to find efficient algorithms to compute morphological granulometry of an image. Some fast algorithms have been proposed by Vincent [11,12], Van Herk [13], and Soille et al. [14] for one-dimensional (linear) binary (flat) structuring elements. Development of fast algorithms for granulometry computation using a general two-dimensional grayscale (non-flat) structuring element is an active research topic. Another important consideration in implementing a granulometry is the memory requirement of the algorithm. Most algorithms for computing granulometry require excessive amounts of memory (2–3 image arrays). Also, use of secondary storage devices like hard-disk drives for storing intermediate results increases the computation time, since these devices normally tend to be very slow compared to the primary memory. An algorithm with small memory requirement has been proposed by Lee et al. [15] for the case of one-dimensional signals.

In this paper we first propose a computationally efficient algorithm for computing erosions and dilations by one-dimensional grayscale structuring elements with constant slope. The computational complexity of this algorithm is independent of the size of the support of the structuring element. This is a generalization of the method proposed by Van Herk [13] for the case of erosion and dilation by flat one-dimensional structuring elements. This algorithm is then used to efficiently compute the granulometry of an image with two-dimensional grayscale structuring elements, where the number of operations depends linearly (as opposed to quadratic or cubic dependence for traditional algorithms) on the number of openings. Many other useful two-dimensional structuring elements can be approximated by appropriate combinations of these structuring elements, using the properties of erosion and dilation.

Theoretical and experimental results comparing the complexity of this algorithm with other standard techniques is presented.

We then address several implementation issues in computing a general granulometry. An efficient approach to compute the "local volume" (required to compute texture features from the granulometry) is discussed. In order to make efficient use of the hardware resources (e.g. on-chip cache, primary memory, etc.) available in the computer, it is often necessary to divide a large image into smaller patches (sub-images). Naturally, the optimum size of these patches is a function of the specific hardware. We have experimentally obtained the optimum patch size for three different hardware platforms (Sun SPARC 20 workstation, SGI 10000XL Power Challenge, and DELL 400 workstation). Parallel implementation of the different algorithms on two multi-processor machines (SGI 10000XL Power Challenge, and DELL 400 workstation) is then discussed. For memory efficient implementation, a "circular buffer" approach and a two-dimensional extension of Lee’s algorithm is proposed. Finally, we present some concluding remarks.

We conclude the introduction by providing the basic morphological definitions. For an image \( X \) and grayscale structuring element \( B \), the erosion \( X \ominus B \) is defined by

\[
X \ominus B = \inf_{z \in D_B} \{ X \ominus z - B(z) \} = \inf_{z \in B_x} \{ X(z) - \hat{B}_z \}
\]

and the dilation \( X \oplus B \) is defined as

\[
X \oplus B = \sup_{z \in D_B} \{ X_z + B(z) \} = \sup_{z \in D_x} \{ X(z) + B_z \},
\]

where \( X_z \) represents \( X \) shifted by \( z \), \( D_B \) is the domain of \( B \), and \( \hat{B}_z \) denotes the reflection of structuring element \( B \).

These basic operations are combined to generate opening and closing operations [16]:

\[
X \circ B = (X \ominus B) \oplus B \text{ and } X \bullet B = (X \oplus B) \ominus B,
\]

respectively. Opening and closing are duals of each other, in the sense that \( X \circ B = (X^c \bullet B)^c \). Furthermore, openings have the property of anti-extensivity, i.e. an opened image is fully contained inside the original image \((X \circ B) \leq X\), whereas closing is an extensive operation.

A morphological granulometry (anti-granulometry) is essentially a sequence of morphological openings (closings) by a series of structuring elements given by...
where dimensional structuring elements with constant slope. In this section, we present a computationally efficient algorithm for erosions and dilations by grayscale one-dimensional structuring elements, where \( k \) is a parameter that successively reduces the volume beneath the opened (closed) image. The rate at which this volume is removed (increased) depends on the structuring element \( B \) and is often used to characterize geometric texture information in the image. The variation of this volume vs. the size \( k \) of the structuring element is called a size distribution \( \Omega(k) \) [9,16].

When normalized \( \Phi(k) = 1 - \Omega(k)/\Omega(0) \), \( \Phi(k) \) represents a probability distribution function called a size distribution, where \( \Omega(0) \) is the volume under the graph of the original image. In this case, \( p(k) = \Phi(k) - \Phi(k - 1) \) is a probability density function and is referred to as the pattern spectrum [17]. The moments of the pattern spectrum have been used as morphological texture features [6,7,10]. Normally, texture is regarded as a local property and any volume decrease is computed within a local window \( W \) placed around each pixel. Hence, the texture features will, in general, be a function of the size of the local window \( W \). Morphological texture features have been extensively used for the analysis, segmentation and classification of texture [6,8,18]. These texture features, however, are computationally very intensive. This poses a significant problem in applications which involve large images or require close to real-time operation.

A Computationally Efficient Algorithm

In this section, we present a computationally efficient algorithm for erosions and dilations by grayscale one-dimensional structuring elements with constant slope.

Consider two structuring functions \( B_{1h} \) and \( B_{2h} \), where

\[
B_{1h}(u) = \begin{cases} 
    a, & u = (0,0) \\
    0, & u = (1,0) 
\end{cases}
\]

and

\[
B_{2h}(u) = \begin{cases} 
    a, & u = (0,0) \\
    0, & u = (-1,0) 
\end{cases}
\]

Given an image \( X \), we want to obtain the dilated image \( X \oplus kB_{ih} \), where

\[
kB_{ih} = \begin{cases} 
    B_{ih} & \text{for } k = 1 \\
    (k-1)B_{ih} \oplus B_{ih} & \text{for } k = 2,3,\ldots 
\end{cases}
\]

Since the support of \( kB_{ih} \) is a subset of the horizontal line, we can perform the dilation \( X \oplus B_{ih} \) “row-wise.”

Henceforth, without loss of generality, we shall denote pixels using just one co-ordinate, namely the horizontal co-ordinate, and denote a single row of the image \( X \) by \( f \).

We first consider dilation of \( f \) with the structuring function \((k-1)B_{ih}\), where \( k \geq 2 \). Note that the support of \((k-1)B_{ih}\) has \( k \) points. First divide the input image \( f \) into sub-arrays of size \( k \). If necessary, the image is extended so that the number of pixels in \( f \) is a multiple of \( k \) by padding it with the value \(-\infty\) at every \( k \)th pixel. Henceforth, we shall assume, without loss of generality, that the length \( N \) of the image array \( f \) satisfies \( N = Mk \).

The following proposition gives an algorithm for computing the dilation \( f \oplus B_{ih} \) of \( f \) by \( B_{ih} \).

**Proposition 1**: Given an image \( f \), define two images \( g \) and \( h \) as follows:

\[
g(u) = \begin{cases} 
    f(u) & \text{for } u = mk, m = 0, 1, \ldots, M-1 \\
    f(u) \vee (g(u-1) - a) & \text{for } u = mk+i, \text{ where } i = 1, 2, \ldots, k-1, \text{ and } m = 0, 1, \ldots, M-1 \\
    \bigvee_{i=0}^{u \mod k} (f(u-i) - ia) & \text{for } u \equiv 0 \mod k
\end{cases}
\]

and

\[
h(u) = \begin{cases} 
    f(u) & \text{for } u = (m+1)k-1, \text{ where } m = 0, 1, \ldots, M-1 \\
    f(u) \vee (h(u+1) + a) & \text{for } u = mk+i, \text{ where } i = k-2, \text{ and } m = 0, 1, \ldots, M-1 \\
    \bigvee_{i=0}^{k-1-(u \mod k)} (f(u+i) + ia) & \text{for } u \equiv k-1 \mod k \\
    \bigvee_{i=(u \mod k)}^{k-1} (f(u-i+k-1) + (k-1-i)a) & \text{for } u \equiv (u \mod k)
\end{cases}
\]

Note that the array \( g \) is computed recursively, from left to right in each sub-array of length \( k \), whereas the array \( h \) is computed recursively, from right to left in each sub-array of length \( k \). The dilation of \( f \) by \( B_{ih} \) is given by \( d = f \oplus (k-1)B_{ih} \).

---

1Proofs for all results are given in the appendix.

2In the following, \( \bigvee \) denotes supremum and \( \bigwedge \) denotes infimum.
where
\[ d(u) = h(u - k + 1) \lor (g(u) + (k - 1)a). \]  
(9)

Notice that computation of \( g \) involves one max and one subtraction operation per pixel, whereas computation of \( h \) and \( d \) involve one max and one addition operation per pixel. Hence, we require three max and three addition/subtraction operations (per pixel) to compute the dilation \( f \oplus (k - 1)B_{1h} \), which is independent of \( k \).

We now have an analogous result for the case of dilation with the structuring function \( B_{2h} \).

**Proposition 2:** Given an image \( f \), define images \( g \) and \( h \) as follows:

\[
\begin{align*}
g(u) &= \begin{cases} f(u) & u = mk, \ m = 0, 1, \ldots, M - 1 \\
               & u = mk + i; \ i = 1, 2, \ldots, k - 1, \\
               & \text{and} \ m = 0, 1, \ldots, M - 1 \\
               & \lor \sum_{i=k-1-(u \bmod k)}^{k-1} (f(u + i - (k - 1)) + (k - 1 - i)a), \\
               & k - 1 - (u \bmod k) \\
\end{cases} \\
&= \begin{cases} f(u) \lor (h(u + 1) - a) & u = mk + i; \ i = k - 2, \\
               & k - 3, \ldots, 0, \text{and} \\
               & m = 0, 1, \ldots, M - 1 \end{cases} \\
&= \sum_{i=0}^{k-1-(u \bmod k)} (f(u + i) - ia).
\end{align*}
\]

(10)

and

\[
\begin{align*}
h(u) &= \begin{cases} f(u) & u = (m + 1)k - 1, \ m = 0, 1, \ldots, M - 1 \\
               & u = mk + i; \ i = k - 2, \\
               & k - 3, \ldots, 0, \text{and} \\
               & m = 0, 1, \ldots, M - 1 \end{cases} \\
&= \sum_{i=0}^{k-1-(u \bmod k)} (f(u + i) + ia).
\end{align*}
\]

(11)

Note that the array \( g \) is computed recursively, from left to right in each sub-array of length \( k \), whereas the array \( h \) is computed recursively, from right to left in each sub-array of length \( k \). The dilation of \( f \) by structuring function \( (k - 1)B_{2h} \) is given by \( d = f \oplus (k - 1)B_{2h} \), where

\[ d(u) = g(u + k - 1) \lor (h(u) + (k - 1)a). \]  
(12)

Notice that computation of \( g \) and \( d \) involve one max and one addition operation per pixel, whereas computation of \( h \) involves one max and one subtraction operation per pixel. Hence, we require three max and three addition/subtraction operations (per pixel) to compute the dilation \( f \oplus (k - 1)B_{2h} \), which is independent of \( k \).

The following two Propositions are the analogs of Propositions 1 and 2, respectively, for the case of erosion.

**Proposition 3:** Given an image \( f \), define images \( g \) and \( h \) as follows:

\[
\begin{align*}
g(u) &= \begin{cases} f(u) & u = mk, \ m = 0, 1, \ldots, M - 1 \\
               & u = mk + i; \ i = 1, 2, \ldots, k - 1, \\
               & \text{and} \ m = 0, 1, \ldots, M - 1 \end{cases} \\
&= \sum_{i=k-1-(u \bmod k)}^{k-1} (f(u + i - (k - 1)) - (k - 1 - i)a)
\end{align*}
\]

(13)

and

\[
\begin{align*}
h(u) &= \begin{cases} f(u) & u = (m + 1)k - 1, \ m = 0, 1, \ldots, M - 1 \\
               & u = mk + i; \ i = k - 2, \\
               & k - 3, \ldots, 0, \text{and} \\
               & m = 0, 1, \ldots, M - 1 \end{cases} \\
&= \sum_{i=0}^{k-1-(u \bmod k)} (f(u + i) + ia).
\end{align*}
\]

(14)

Note that the array \( g \) is computed recursively, from right to left in each sub-array of length \( k \), whereas the array \( h \) is computed recursively, from right to left in each sub-array of length \( k \). The erosion of \( f \) by structuring function \( (k - 1)B_{1h} \) is given by \( e = f \ominus (k - 1)B_{1h} \), where

\[ e(u) = g(u + k - 1) \land (h(u) - (k - 1)a). \]  
(15)

**Proposition 4:** Given an image \( f \), define two images \( g \) and \( h \) as follows:

\[
\begin{align*}
g(u) &= \begin{cases} f(u) & u = mk, \ m = 0, 1, \ldots, M - 1 \\
               & u = mk + i; \ i = 1, 2, \ldots, k - 1, \\
               & \text{and} \ m = 0, 1, \ldots, M - 1 \end{cases} \\
&= \sum_{i=0}^{u \bmod k} (f(u - i) + ia)
\end{align*}
\]

(16)
and

\[ h(u) = \begin{cases} 
  f(u) & u = (m+1)k - 1, \\
  f(u) \land (h(u + 1) - a) & u = mk + i; i = k - 2, \\
  & k - 3, \ldots, 0, \\
  & m = 0, 1, \ldots, M - 1 
\end{cases} \]

\[ = \bigwedge_{i=(u \mod k)}^{k-1} (f(u - i + k - 1) - (k - 1 - i)a). \quad (17) \]

Note that the array \( g \) is computed recursively, from left to right in each sub-array of length \( k \), whereas the array \( h \) is computed recursively, from right to left in each sub-array of length \( k \). The erosion of \( f \) by structuring function \((k - 1)B_{2k}\) is given by \( e = f \odot (k - 1)B_{2k} \), where

\[ e(u) = h(u - k + 1) \land (g(u) - (k - 1)a). \quad (18) \]

Therefore, computation of the dilation \( f \odot kB = (f \odot kB_{1k}) \lor (f \odot kB_{2k}) \) would require seven max and six addition/subtraction operations per pixel, independent of the value of \( k \). Finally, observe that when \( a = 0 \), Propositions 1–4 reduce to the algorithm proposed by Van Herk [13] for computing flat erosions and dilations for one-dimensional structuring elements.

**Comparison with other Algorithms**

In this section, we shall derive expressions for the computational complexity for granulometry computation using different algorithms.

**Direct approach**

This is the most straightforward way of computing a granulometry by performing a sequence of openings and closings. This, in turn, is done by erosions and dilations, using the first expression in Eqns (1) and (2), by shifting the image over each pixel of the structuring element, and then adding/subtracting the value of the structuring element to it. This scheme is more appropriate for software pipelining. The image is stored in memory one row at a time, i.e., the second row starts at the end of the first one, and so on. Computers that use software pipelining run faster by accessing the data in the sequence in which it is stored, rather than jumping from row to row. Openings done using this approach require at least two images to be stored in the memory at all times—one for the original image and one for the eroded/dilated image. Moreover, to avoid re-reading the image from a secondary storage device, at least three images have to be stored in the memory at all times. For very large images such as mammograms, this poses a bottleneck.

Consider an \( N \times N \) image \( X \) and a grayscale structuring element \( B \) whose support has \( M \times M \) pixels. Direct computation of the opening \( X \odot B \) and closing \( X \bullet B \) of \( X \) by \( B \) would require \( N^2M^2 \) additions and \( N^2M^2 \) subtractions for dilation and \( N^2M^2 \) subtractions for erosion. Computing the granulometry of \( X \) by \( B \) requires the computation of openings \( \{X \odot nB\}_{n=1,2,\ldots,p} \), where \( p \) is some large integer. Since the support of \( nB \) has \( (n(M - 1) + 1) \times (n(M - 1) + 1) \) pixels, the direct computation of \( X \odot nB \) would require

\[ 2N^2(n(M - 1) + 1)^2 \]

comparisons and an equal number of additions/subtractions, whereas computing the granulometry would require

\[ 2N^2\left(p + (M - 1)^2 \sum_{n=1}^{p} n^2 + 2(M - 1) \sum_{n=1}^{p} n\right) \]

\[ = 2N^2\left(p + (M - 1)^2\frac{p(p + 1)(2p + 1)}{6} \right) \]

\[ + (M - 1)p(p + 1) \]

(20)

comparisons and an equal number of additions/subtractions. Thus the computational complexity for granulometry computation with this direct approach is (ignoring lower order terms) \( O(N^2M^2p^3) \). As an example, a 10 point granulometry on a 3000 × 5000 16 bits/pixel mammogram, starting with a 3 × 3 structuring element, requires about \( 10^{11} \) operations, and about 315 MBytes of memory. And this is merely for one structuring element. In practice, many different structuring elements or texture types are examined.

**Cascaded approach**

This known approach removes certain redundancies in the direct approach. It takes advantage of the fact that the granulometry is formed by opening/closing the image by structuring elements of the form \( kB \), \( k = 1, 2, \ldots, p \). Using the serial decomposition properties \( X \odot (B \odot C) = (X \odot B) \odot C \) and \( X \odot (B \oplus C) = (X \odot B) \oplus C \)
of erosion and dilation, it is easy to see that
\[
X \circ nB = (X \ominus nB) \oplus nB = (\ldots ((X \ominus B) \ominus B) \oplus \ldots \ominus B) \oplus \ldots \oplus B.
\]

Therefore, \(X \circ nB\) can be computed using \(2N^2M^2n\) comparisons and an equal number of additions/subtractions. While computing the granulometry \(X \circ nB\), we can use \(X \ominus (n-1)B\), which was computed for \(X \circ (n-1)B\) (assuming it has been stored at the previous step). This approach is illustrated in Figure 1 and is referred to as the cascaded approach. In this case, computing the granulometry would require
\[
N^2M^2 \sum_{n=1}^{p} (1 + n) = N^2M^2(2p^2 + 3p)/2
\]
comparisons and an equal number of additions/subtractions. Thus the computational complexity for granulometry computation with this approach is (ignoring lower order terms) \(O(N^2M^2p^2)\). In this approach, it is required to store at least three images in the memory at any given time—one for the eroded image, one for the intermediate dilated image, and one more for the next dilated image. One more positive aspect of this approach is that it does not require rereading the original image from the secondary storage device for doing the successive openings/closings. However, this approach requires retaining the eroded/dilated image of the last opening/closing stage in the memory or the secondary storage device.

\textbf{Proposed fast algorithm}

The main feature of our proposed algorithm (similar to the one proposed by Van Herk [13] for flat structuring elements) is that the number of comparisons and additions/subtractions to compute the opening \(X \circ nB\) is independent of \(n\). Indeed, from Propositions 1–4 it follows that \(X \circ nB_{1h}\) can be computed using \(6N^2\) comparisons and an equal number of additions/subtractions. Consequently, computing the granulometry would involve \(6N^2p\) comparisons and an equal number of additions/subtractions.

The price one has to pay for this computational saving is that the choice of structuring element \(B\) is restricted. Propositions 1–4 are restricted to one-dimensional grayscale structuring elements \(B\) with a constant slope. However, by a judicious choice of such one-dimensional structuring elements, and using some properties of erosion and dilation, it is possible to achieve substantial computational savings for computing granulometries by a large class of structuring elements, including two-dimensional ones, as explained below. For example, consider the structuring elements \(B_{1h}, B_{2h}\) given by (4), (5), and their vertical counterparts \(B_{1v}, B_{2v}\) given by
\[
B_{1v}(u) = \begin{cases} 
  h, & u = (0, 0) \\
  0, & u = (0, 1), 
\end{cases}
\]
and
\[
B_{2v}(u) = \begin{cases} 
  h, & u = (0, 0) \\
  0, & u = (0, -1). 
\end{cases}
\]

It is then easy to see that \(B_h = B_{1h} \lor B_{2h}\) is a triangular grayscale horizontal structuring element. Therefore,
Experimental Results

In this section, we present some experimental results comparing the CPU time for granulometry computation, using our algorithm and other standard implementations discussed in the previous section. First we studied the CPU times for the different algorithms as a function of the number of openings \( p \) in the granulometry. A 1792 \( \times \) 2560 pixel digitized mammogram image (100 \( \mu \)m resolution with 8 bits/pixel) was used. The structuring element \( B \) depicted in Figure 2 was used. Figure 3 depicts the CPU times on a DELL 400 workstation. From the graph, it is clear that the execution time increases linearly with \( p \) using our algorithm, quadratically with \( p \) for a cascaded implementation, and cubically with \( p \) for a direct implementation. This is in agreement with the computational complexities derived in the previous section.

As is clear from the discussion in the previous section, the proposed algorithm can achieve substantial computational savings provided the structuring element can be decomposed into a small number of one-dimensional (grayscale) structuring elements with constant slope (by means of dilation and supremum operations). As the number of one-dimensional structuring elements \( q \) in the decomposition increases, the computational savings decrease, since the computational complexity is directly proportional to \( q \). As discussed in the previous section, the computation time (for a single opening) of our algorithm increases linearly with the number of linear segments in the decomposition of the structuring element, whereas the computation time of the direct/cascaded approach increases linearly with the number of pixels in the structuring element. Basically, while our algorithm can, in principle, be used with any structuring element, it is most advantageous to use it for those structuring elements \( B \) which can be decomposed by means of a small number of one-dimensional structuring elements, and for computing openings with large structuring elements \( nB \), as is required for granulometric texture analysis.

Memory Efficient Approaches

In this section, we present two approaches to computing a granulometry whose memory requirements are pretty modest.

Circular buffer approach

This approach uses circular buffers to generate a memory efficient implementation of erosion and dilation operations. It uses the second expression for erosion and dilation in Eqns (1) and (2), respectively. Here, the erosion/dilation is computed by sliding the structuring element...
element over each pixel of the image, adding the gray-value of each pixel to that of the structuring element, and computing a local minimum/maximum operation for each pixel.

A circular buffer is a finite element array which, when filled, allows the next new entry to overwrite the oldest one in the buffer. In a two-dimensional circular buffer, each entry is a pointer to a row of the image. Thus, when pixel values of the opened image for a particular row are computed, it is replaced by a new row of the original image. As shown in Figure 4, as the structuring element slides from left to right and top to bottom, the top left pixel under the structuring element does not come under the structuring element in subsequent scans. This pixel is then immediately sent to the next processing stage.

Figure 3. Comparison of execution times on a DELL 400 workstation for computing a granulometry with three different approaches. The execution times include writing each of the opened images onto disk.

Figure 4. Circular buffer approach for opening.
For opening, two circular buffers are required—an erosion buffer, and a dilation buffer—each having the same number of rows as that of the structuring element, and the same number of columns as that of the original image. As the structuring element is slid over the image, the gray-value of each structuring element pixel is subtracted from that of the image. It is then compared with the corresponding pixels in the erosion buffer in order to obtain the minimum value at each pixel. The top left pixel under the structuring element no longer comes under the influence of the structuring element, and is immediately processed for dilation. In the erosion buffer this location is then initialized to its initial value (usually \( \infty \), or the largest possible integer in the machine). This pixel now assumes the role of a new image pixel to be eroded. In the dilation buffer each pixel of the structuring element is added to the eroded pixel, and compared with the corresponding pixel values in the dilation buffer in order to find the maximum value. Again, the top left pixel under the structuring element never again comes under the influence of the structuring element. This pixel is now an opened pixel, and is immediately saved to a file. The method described above can also be used to compute the local volume inside a window of the opened image. Thus, granulometries and local volumes can be computed by a single scan, and without storing the entire image (just a few circular buffers) in memory.

The circular buffer approach can be used in two ways:

- Perform one opening for a scan of the original image, and then rescan the original image to do the next opening.
- Perform multiple openings in just one scan of the original image.

The first method requires multiple reads of the original image for computing multiple openings. Of all data transfers, those to a secondary storage device are the slowest, and hence should be avoided whenever possible. The second method minimizes the accesses to a secondary storage device. However, the implementation of this version requires a considerable amount of indirections (pointers pointing to pointers), which greatly slows down a general purpose computer. On the other hand, this type of implementation would be suitable for specially designed hardware, where the complete size distribution for each pixel can be computed by utilizing three circular buffers designed to carry out erosion, dilation, and volume computation.

Since, the circular buffer approach and the direct approach directly use Minkowski algebra (Eqns (1) and (2)) to compute the openings/closings, they require the same number of operations. For computing openings on a general purpose computer which has data pipelining, the circular buffer approach is actually more efficient than the direct approach, since jumping from one row to another is less efficient than going over the pixels in the way they are stored in memory. However, the advantage of this approach lies in its very small memory requirement. An opening using circular buffer requires two circular buffers having the number of rows equal to that of the structuring element, and the number of columns equal to that of the image.

Two-dimensional extension of Lee’s algorithm

Lee et al. [15] have introduced a memory efficient algorithm for computing the granulometry of one-dimensional signals, with arbitrary structuring element. We have extended this algorithm to two-dimensional signals (images).

Opening of an image \( X \) by a structuring element \( B \) of size \( M \times M \) at pixel location \((n_x, n_y)\) can be represented as follows:

\[
(X \circ B)(n_x, n_y) = \sup_{x', y' \in D_B} \left\{ \inf_{x, y \in D_B} \left\{ X(n_x - x' + x, n_y - y' + y) + h(x', y', x, y) \right\} \right\}
\]

(25)

where

\[
h(x', y', x, y) = B(x', y') - B(x, y).
\]

(26)

The idea in extending Lee’s algorithm to image signals lies in representing the image and the basis matrix. Consider a row vector:

\[
X_{ij}(n_x, n_y) = \begin{cases} 
X(n_x - i, n_y - j), & X(n_x - i + 1, n_y - j), \\
X(n_x - i + M - 1, n_y - j), & X(n_x - i, n_y - j + 1), \\
X(n_x - i + 1, n_y - j + 1), & X(n_x - i + M - 1, n_y - j + 1), \\
X(n_x - i, n_y - j + M - 1), & X(n_x - i + 1, n_y - j + M - 1), \\
X(n_x - i + M - 1, n_y - j + M - 1), & \ldots 
\end{cases}
\]

(27)
and define an image matrix as follows:

\[
X(n_x, n_y) = \begin{bmatrix}
X_{00}(n_x, n_y) \\
\vdots \\
X_{(M-1)0}(n_x, n_y) \\
\vdots \\
X_{0(M-1)}(n_x, n_y) \\
\vdots \\
X_{(M-1)(M-1)}(n_x, n_y)
\end{bmatrix}.
\] (28)

Similarly, consider a basis matrix row vector:

\[
H_{ij} = [h(i, j, 0, 0), h(i, j, 1, 0), \ldots, h(i, j, M - 1, 0),
\]

\[
h(i, j, 0, 1), h(i, j, 1, 1), \ldots, h(i, j, M - 1, 1), \ldots,
\]

\[
h(i, j, 0, M - 1), h(i, j, 1, M - 1), \ldots,
\]

\[
h(i, j, M - 1, M - 1)],
\] (29)

and define a basis matrix:

\[
H = \begin{bmatrix}
H_{00} \\
\vdots \\
H_{(M-1)0} \\
\vdots \\
H_{0(M-1)} \\
\vdots \\
H_{(M-1)(M-1)}
\end{bmatrix}.
\] (30)

It follows from (25)–(30) that \((X \circ B)(n_x, n_y)\) can be obtained by finding the supremum of the infimums of each row of the matrix \(X(n_x, n_y) + H\). If \(R_{ij}(n_x, n_y)\) is the row-wise infimum of the \((Mj + i)^{th}\) row of the matrix \(X(n_x, n_y) + H\), then it can be shown that³

\[
R_{ij}(n_x, n_y) = R_{(ij-1)}(n_x, n_y) - 1 + h(i, j, i, j - 1),
\] (31)

for every \(i = 0, 1, \ldots, M - 1\) and \(j = 1, 2, \ldots, M - 1\). The opening \(X \circ B\) can now be rewritten as

\[
(X \circ B)(n_x, n_y) = \sup \left\{ R_{00}(n_x, n_y), \ldots, R_{(M-1)0}(n_x, n_y),
\right.
\]

\[
(R_{00}(n_x, n_y - 1) + h(0, 1, 0, 0)), \ldots,
\]

\[
(R_{(M-1)0}(n_x, n_y - 1) + h(M - 1, 1, M - 1, 0)), \ldots,
\]

\[
(R_{0(M-2)}(n_x, n_y - 1) + h(0, M - 1, 0, M - 2)), \ldots,
\]

\[
(R_{(M-1)(M-2)}(n_x, n_y - 1) + h(M - 1, M - 1, M - 1, M - 2))\right\}.\] (32)

Analogous expressions for closings can be derived easily.

Figure 5 illustrates the implementation of this algorithm. The origin of the structuring element \(B\) is placed at the current pixel (assumed to be \((0,0)\) here for illustration), and the first row of the matrix \(X(0,0)\) is formed by scanning the image pixels along the dotted arrow generating the row vector (see also (27) and (28))

\[
X_{00}(0,0) = \begin{bmatrix}
X(0,0), X(1,0), X(2,0), X(0,1), X(1,1),
\end{bmatrix},
\]

\[
X(2,1), X(0,2), X(1,2), X(2,2).
\]

The second row of \(X(0,0)\) is formed by placing the origin of the structuring element at \((-1,0)\) and scanning the image in the same fashion. The rest of the rows of the matrix \(X(0,0)\) are formed by placing the origin of the structuring element at pixels along the solid arrow, and scanning the image pixels in the direction of the dotted arrow. Thus, the \(X(0,0)\) matrix can be formed. Now suppose the row-wise infimums for all rows for the pixel

³A proof can be found in the appendix

Figure 5. Modified Lee’s algorithm for images.
(0,-1) has been found, then while finding the row-wise infimums for the rows obtained by placing the origin of the structuring element at pixel (0,0), only the row-rise infimum of the rightmost pixels (i.e. \(X(0,0), X(1,0), X(2,0)\)) has to be found. The remaining row-wise infimums can be found be adding a term of the basis matrix to the row-wise infimums found while computing the opening at \((0,-1)\) in accordance with \((31)\). Opening at the pixel \((0,0)\) can be found by taking the supremum of each of these row-wise infimums. Thus, for finding the opening of each new pixel, row-wise infima of \(M\) rows have to be found (infima of \((M-1)\times M\) rows being derived from the row-wise infima computed for the previous pixel). Supremum of all these row-wise infima is then computed.

Now, consider the opening of an image by a structuring element of size \(M \times M\) using our 2D extension of Lee’s algorithm. For computing the row-wise infimum for every new row, \(M^2\) additions and \((M^2 -1)\) comparisons are required. For every pixel of image, there are \(M\) new rows, and \((M^2 - M)\) old rows. Each old row requires only one addition. For finding the final column-wise supremum, \((M^2 -1)\) comparisons are required. Thus, the total number of operations required for opening each pixel is

\[
M(M^2 + M^2 - 1) + M^2 - M + M^2 - 1 = 2M^3 + 2M^2 - 2M - 1. \tag{33}
\]

Hence, the total of operations of opening an \(N \times N\) image is \(N^2(2M^3 + 2M^2 - 2M - 1)\), which is very high compared to that of the direct approach \((4N^2M^2)\). However, as is evident from the above description, it requires much less memory for computing openings/closings, and is well suited for limited memory implementations.

### Implementation Issues

In this section, we discuss some issues that are pertinent to the implementation of algorithms like volume computation, patch implementation and parallel implementation.

#### Efficient local volume computation

As discussed before, morphological features are obtained by computing the moments of the pattern spectrum. This involves computing the volume beneath each of the opened images in the granulometry, over a suitable local window \(W\). While computing the local volume by shifting a window over to each image pixel, there is a lot of redundancy, since there is considerable overlap between these windows. This redundancy can be eliminated by finding the volume inside a window around a given pixel, in terms of that of its adjacent pixel, in a recursive fashion. This is done by subtracting gray-values of pixels that are outside the new window but inside the old window from the previously computed local volume, and then adding the gray-values of pixels that are inside the new window but outside the old window. The window is slid from top to bottom, and from left to right, as shown in Figure 6, because the images are stored in memory row-wise.

#### Patch implementation

As the image size increases, most computers would require the use of virtual memory on secondary storage devices for storing data. Of all data transfers, data transfers from secondary storage devices (e.g. disk drives) are the slowest. Thus, for maximizing the speed of computation, the image may be divided into smaller segments or patches. Computation on each of the patches can be done separately and the results can be put together in the end. The size of the patch would naturally depend on the hardware configuration of the machine. Since the opening at a given pixel depends on the pixel values in its neighborhood, extra pixels need to be processed at the patch edges. In other words, the
patches must overlap. The number of overlapping pixels at an edge is equal to
\[ (N_e \times s_t) + (N_d \times s_r) + \lfloor L/2 \rfloor, \]
(34)
at the left-hand side of the patch,
\[ (N_e \times s_r) + (N_d \times s_t) + \lfloor L/2 \rfloor, \]
(35)
at the right-hand side of the patch
\[ (N_e \times s_t) + (N_d \times s_b) + \lfloor L/2 \rfloor, \]
(36)
and at the top of the patch, and
\[ (N_e \times s_b) + (N_d \times s_t) + \lfloor L/2 \rfloor, \]
(37)
at the bottom of the patch. Here, \( N_e, N_d \) is the number of erosions and dilations, respectively, \( L \times L \) is the window size, \( s_t, s_r, s_t, s_b \) are the number of structuring element pixels to the left, right, top and bottom, respectively, of the origin, and \( \lfloor x \rfloor \) denotes the truncation of a real number to an integer. The number of extra computations required for each patch depends on its perimeter. Since the perimeter of a rectangular patch of a fixed area is minimum for a square patch, the number of extra computations would be the smallest for a square patch. This was verified experimentally by dividing the image into rectangular patches of the same area but with different dimensions.

The computation time of our new algorithm was compared with that of the cascaded approach for computing \( p = 10 \) openings and closings, followed by local volume and moment computations, using a patch implementation. The codes were tested on a 1500 \( \times \) 1500 16-bit mammogram image using the elementary 3 \( \times \) 3 structuring element depicted in Figure 2 and a 15 \( \times \) 15 window \( W \). We used three different machines—SGI Power Challenge 10000XL, SUN SPARC 20 workstation and Pentium II powered Dell 400 workstation—to benchmark our code. Since the machines have different primary and secondary cache, main memory, and architecture, they have a different optimum patch size at which the computation time is minimized.

Table 1. Comparison of local granulometry timings for different square patch sizes on SGI Power Challenge using the cascaded approach

<table>
<thead>
<tr>
<th>Patch size (in pixels)</th>
<th>Patch area</th>
<th>No. of patches</th>
<th>Timings (nearest second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 ( \times ) 50</td>
<td>2500</td>
<td>900</td>
<td>643</td>
</tr>
<tr>
<td>75 ( \times ) 75</td>
<td>5625</td>
<td>400</td>
<td>582</td>
</tr>
<tr>
<td>100 ( \times ) 100</td>
<td>10000</td>
<td>225</td>
<td>555</td>
</tr>
<tr>
<td>150 ( \times ) 150</td>
<td>22500</td>
<td>100</td>
<td>671</td>
</tr>
<tr>
<td>250 ( \times ) 250</td>
<td>62500</td>
<td>36</td>
<td>1157</td>
</tr>
<tr>
<td>300 ( \times ) 300</td>
<td>90000</td>
<td>25</td>
<td>1350</td>
</tr>
<tr>
<td>375 ( \times ) 375</td>
<td>140625</td>
<td>16</td>
<td>1417</td>
</tr>
<tr>
<td>500 ( \times ) 500</td>
<td>250000</td>
<td>9</td>
<td>1262</td>
</tr>
<tr>
<td>750 ( \times ) 750</td>
<td>562500</td>
<td>4</td>
<td>1349</td>
</tr>
</tbody>
</table>

Table 2. Comparison of local granulometry timings for different square patch sizes on SGI Power Challenge using our new algorithm

<table>
<thead>
<tr>
<th>Patch size (in pixels)</th>
<th>Patch area</th>
<th>No. of patches</th>
<th>Timings (nearest second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 ( \times ) 50</td>
<td>2500</td>
<td>900</td>
<td>353</td>
</tr>
<tr>
<td>60 ( \times ) 60</td>
<td>3600</td>
<td>625</td>
<td>344</td>
</tr>
<tr>
<td>75 ( \times ) 75</td>
<td>5625</td>
<td>400</td>
<td>288</td>
</tr>
<tr>
<td>100 ( \times ) 100</td>
<td>10000</td>
<td>225</td>
<td>248</td>
</tr>
<tr>
<td>125 ( \times ) 125</td>
<td>15625</td>
<td>144</td>
<td>266</td>
</tr>
<tr>
<td>150 ( \times ) 150</td>
<td>22500</td>
<td>100</td>
<td>235</td>
</tr>
<tr>
<td>250 ( \times ) 250</td>
<td>62500</td>
<td>36</td>
<td>274</td>
</tr>
<tr>
<td>300 ( \times ) 300</td>
<td>90000</td>
<td>25</td>
<td>262</td>
</tr>
<tr>
<td>375 ( \times ) 375</td>
<td>140625</td>
<td>16</td>
<td>271</td>
</tr>
<tr>
<td>500 ( \times ) 500</td>
<td>250000</td>
<td>9</td>
<td>345</td>
</tr>
<tr>
<td>750 ( \times ) 750</td>
<td>562500</td>
<td>4</td>
<td>384</td>
</tr>
<tr>
<td>1500 ( \times ) 1500</td>
<td>2250000</td>
<td>1</td>
<td>395</td>
</tr>
</tbody>
</table>

Table 3. Comparison of local machines for cascaded approach, each machine using its own optimal patch size

<table>
<thead>
<tr>
<th>Computer</th>
<th>Preferred patch size (in pixels)</th>
<th>Timings (nearest second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGI Power Challenge 10000XL</td>
<td>100 ( \times ) 100</td>
<td>555</td>
</tr>
<tr>
<td>DELL 400 workstation</td>
<td>750 ( \times ) 750</td>
<td>416</td>
</tr>
<tr>
<td>SUN SPARC 20 workstation</td>
<td>750 ( \times ) 750</td>
<td>1318</td>
</tr>
</tbody>
</table>

Table 4. Comparison of machines for our new algorithm, each machine using its own optimal patch size

<table>
<thead>
<tr>
<th>Computer</th>
<th>Preferred patch size (in pixels)</th>
<th>Timings (nearest second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGI Power Challenge 10000XL</td>
<td>150 ( \times ) 150</td>
<td>235</td>
</tr>
<tr>
<td>DELL 400 workstation</td>
<td>300 ( \times ) 300</td>
<td>208</td>
</tr>
<tr>
<td>SUN SPARC 20 workstation</td>
<td>300 ( \times ) 300</td>
<td>746</td>
</tr>
</tbody>
</table>
optimal patch size, which also depends on the algorithm. In general, if the patch size is less than the optimum patch size, more patches are required. This leads to extra overhead for managing the patches, as well as for computing the extra information at the edges of each patch. On the other hand, if the patch size is larger than the optimum patch size, there are more frequent cache/main memory misses, and this requires a fresh data transfer from the main memory/secondary storage device, which is much slower than that between the processor and cache/main memory.

Moreover, on a single processor basis, the DELL workstation provided better timings for both the algorithms as compared to the other two machines. This is because, this machine shares the bus with only one other Pentium II processor. Each processor has a 512KB set-associative onchip cache, and the Pentium IIs run at 300MHz. Also, it has 256MB of RAM. In the SGI Power Challenge super-computer, the processors share the common bus, and this slows down the accesses to the main memory/secondary storage device significantly. Each processor in the SGI Power Challenge has 1MB of cache, and runs at 200MHz.

Parallel implementation

Parallelization of a code can be achieved in three ways—function partitioning, data partitioning, and a combination of function and data partitioning. In function partitioning, different functions are assigned to different processors. This partitioning scheme is appropriate when there are functions that can be done in parallel, and these functions do not depend on one another for their execution. In data partitioning, data is divided into pieces, and these pieces are divided amongst the different processors. This scheme is suitable when different pieces of data can be computed independently of each other. Function partitioning is not an ideal scheme for parallel implementation of the cascaded approach, since each opening depends on the previous one. On the other hand, our patch implementation (with either algorithm), is ideal for data partitioning, since a set of patches can be assigned to each processor. Hence, data partitioning was used in parallelizing the final implementations. The codes for computing \( p = 10 \) openings and closings, and three moments of the resulting pattern spectrum, using both the cascaded approach and our new algorithm were parallelized on the SGI and DELL machines. The timings obtained are shown in Tables 5, 6, 7 and 8.

Each processor was assigned a set of patches. Square patches of the optimum size were utilized for each machine. Since the patches were all of the same size, the computation was equally distributed amongst the processors. A 1500 x 1500 16-bit mammogram image, a 3 x 3 structuring element, and a 15 x 15 window were used. As can be seen the computation time decreased

<p>| Table 5. Timings for parallel processing implementation of the code for finding 10 openings/closings and moment computation, using cascaded approach, on a SGI Power Challenge |</p>
<table>
<thead>
<tr>
<th>No. of processors</th>
<th>Timings (nearest second)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>582</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>359</td>
<td>1.6</td>
</tr>
<tr>
<td>3</td>
<td>255</td>
<td>2.3</td>
</tr>
<tr>
<td>4</td>
<td>210</td>
<td>2.8</td>
</tr>
<tr>
<td>5</td>
<td>172</td>
<td>3.4</td>
</tr>
<tr>
<td>6</td>
<td>148</td>
<td>3.9</td>
</tr>
<tr>
<td>7</td>
<td>141</td>
<td>4.1</td>
</tr>
</tbody>
</table>

<p>| Table 6. Timings for parallel processing implementation of the code for finding 10 openings/closings and moment computation, using cascaded approach, on a DELL workstation |</p>
<table>
<thead>
<tr>
<th>No. of processors</th>
<th>Timings (nearest second)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>413</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>340</td>
<td>1.2</td>
</tr>
</tbody>
</table>

<p>| Table 7. Timings for parallel processing implementation of the code for finding 10 openings/closings and moment computation, using our new algorithm, on a SGI Power Challenge |</p>
<table>
<thead>
<tr>
<th>No. of processors</th>
<th>Timings (nearest second)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>292</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>223</td>
<td>1.3</td>
</tr>
<tr>
<td>3</td>
<td>166</td>
<td>1.8</td>
</tr>
<tr>
<td>4</td>
<td>140</td>
<td>2.0</td>
</tr>
<tr>
<td>5</td>
<td>128</td>
<td>2.2</td>
</tr>
<tr>
<td>6</td>
<td>115</td>
<td>2.5</td>
</tr>
<tr>
<td>7</td>
<td>102</td>
<td>2.9</td>
</tr>
</tbody>
</table>

<p>| Table 8. Timings for parallel processing implementation of the code for finding 10 openings/closings and moment computation, using our new algorithm, on a DELL workstation |</p>
<table>
<thead>
<tr>
<th>No. of processors</th>
<th>Timings (nearest second)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>208</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>152</td>
<td>1.4</td>
</tr>
</tbody>
</table>
with an increase in the number of processors but not in a linear fashion. This is because parallelization has its own overheads in creating each thread and in combining them. In general, the use of more processors does not necessarily translate into faster processing times. Best performance was obtained by using seven processors on the SGI Power Challenge. When the number of processors was increased to eight, the parallel processing overhead actually caused an increase in the computation time.

Conclusions and Summary

In this paper, we derived a computationally efficient algorithm for computing erosions and dilations by one-dimensional grayscale structuring elements with constant slope. This was an extension of an algorithm proposed by Van Herk for binary (flat) structuring elements. The main feature of our proposed algorithm is than the number of comparisons and additions/subtractions to compute an opening is independent of the size of the structuring element. This enables efficient computation of granulometries where the number of operations depends linearly on the number of openings. On the other hand, the complexity of most existing algorithms for granulometry is either quadratic or cubic in the size of the structuring element. The price one has to pay for this computational savings is that the choice of structuring element is restricted. However, by a judicious choice of simple structuring elements, and using some properties of erosion and dilation, it is possible to achieve substantial computational savings for computing granulometries by a large class of structuring elements. Experimental results comparing the complexity of this algorithm with other standard techniques agree with the theoretically derived expressions.

We then addressed another major concern in implementing a granulometry—memory requirement. Algorithms for computing granulometry tend to require excessive amounts of memory, especially when dealing with large images. Use of secondary storage devices such as hard-disk drives for storing intermediate results increase the computation time since these devices tend normally to be very slow as compared with the primary memory. Two memory efficient implementations—a circular buffer approach and a 2D extension of Lee’s algorithm—were discussed. The computational complexities and memory requirements for each different algorithm are summarized in Table 9.

Finally, several implementation issues in computing a granulometry and moments of the associated morphological pattern spectrum were discussed. Efficient implementation of granulometries for large images on machines with limited memory requires dividing the image into smaller rectangular patches. The optimum size of these patches, which is a function of the specific hardware, was obtained experimentally for three different hardware platforms. This data partitioning of an image into patches was ideally suited for a parallel implementation on multiprocessor machines. Experimental results of our parallel implementation demonstrate that the achieved parallel processing speedup is limited by the overhead associated with creating various threads.

Our research team has employed these algorithms for computation of morphological texture features in mammogram images. They form the computation engine of an “electronic second opinion” system to allow computer prompts of textural abnormalities in digitized or digital mammograms [19, 20].

Acknowledgements

This work was supported by the Texas Center for Applied Technology. The authors would like to thank S. Batman for several discussions and constructive comments.
References


Appendix A. Proof of propositions

Proof of proposition 1: We consider the following two cases:

**Case 1:** \( u \mod k = k - 1 \).

In this case \( (u - k + 1) \mod k = (u + 1) \mod k = 0 \).

Hence (see Eqn 8),

\[
h(u - k + 1) = \sum_{i=0}^{k-1} (f(u - i) + (k - 1 - i)a)
\]  

(A1)

and (see Eqn 7)

\[
g(u) + (k - 1)a = \sum_{i=0}^{k-1} (f(u - i) - ia) + (k - 1)a
\]

\[
= \sum_{i=0}^{k-1} (f(u - i) + (k - 1 - i)a),
\]  

(A2)

which shows that (see Eqns (9), (A1), and (A2))

\[
d(u) = h(u - k + 1) \lor (g(u) + (k - 1)a)
\]

\[
= \sum_{i=0}^{k-1} (f(u - i) + (k - 1 - i)a)
\]

\[
= (f \oplus (k - 1)B_{1k})(u).
\]  

(A3)

**Case 2:** \( u \mod k \leq k - 2 \).

In this case \( (u - k + 1) \mod k = (u + 1) \mod k = (u \mod k) + 1 \). Therefore (see Eqn 8),

\[
h(u - k + 1) = \sum_{i=(u \mod k)+1}^{k-1} (f(u - i) + (k - 1 - i)a)
\]  

(A4)
and (see Eqn (7))

\[
g(u) + (k-1)a = \bigvee_{i=0}^{\frac{u}{mod\;k}} (f(u-i) - ia) + (k-1)a
\]

\[
= \bigvee_{i=0}^{\frac{u}{mod\;k}} (f(u-i) + (k-1-i)a), \quad (A5)
\]

which shows that (see Eqns (9), (A4), and (A5))

\[
d(u) = h(u-k+1) \lor (g(u) + (k-1)a)
\]

\[
= \bigvee_{i=0}^{k-1} (f(u-i) + (k-1-i)a)
\]

\[
= (f \oplus (k-1)B_1h)(u). \quad (A6)
\]

This completes the proof.

**Proof of proposition 2:** We consider the following two cases:

**Case 1:** \(u \mod k = 0\).

In this case \((u+k-1) \mod k = (u-1) \mod k - 1\). Hence (see Eqn (10)),

\[
g(u+k-1) = \bigvee_{i=0}^{k-1} (f(u+i) + (k-1-i)a) \quad (A7)
\]

and (see Eqn (11))

\[
h(u) + (k-1)a = \bigvee_{i=0}^{k-1} (f(u+i) - ia) + (k-1)a
\]

\[
= \bigvee_{i=0}^{k-1} (f(u+i) + (k-1-i)a), \quad (A8)
\]

which shows that (see Eqns (12), (A7), and (A8))

\[
d(u) = g(u+k-1) \lor (h(u) + (k-1)a)
\]

\[
= \bigvee_{i=0}^{k-1} (f(u+i) + (k-1-i)a)
\]

\[
= (f \oplus (k-1)B_2h)(u). \quad (A9)
\]

**Case 2:** \(u \mod k \geq 1\).

In this case \((u+k-1) \mod k = (u-1) \mod k - 1\). Therefore (see Eqn (10)),

\[
g(u+k-1) = \bigvee_{i=0}^{k-1} (f(u+i) + (k-1-i)a)
\]

\[
= (f \oplus (k-1)B_2h)(u). \quad (A10)
\]

and (see Eqn (11))

\[
h(u) + (k-1)a = \bigvee_{i=0}^{k-1} (f(u+i) - ia) + (k-1)a
\]

\[
= \bigvee_{i=0}^{k-1} (f(u+i) + (k-1-i)a), \quad (A11)
\]

which shows that (see Eqns (12), (A10), (A11))

\[
d(u) = g(u+k-1) \lor (h(u) + (k-1)a)
\]

\[
= \bigvee_{i=0}^{k-1} (f(u+i) + (k-1-i)a)
\]

\[
= (f \oplus (k-1)B_2h)(u). \quad (A12)
\]

This completes the proof.

**Appendix B. Derivation of 2D extension of Lee’s algorithm**

The opening of an image \(X\) by a structuring element \(B\) of size \(M \times M\) can be written as

\[
(X \circ B)(n_x, n_y)
\]

\[
= ((X \ominus B) \oplus B)(n_x, n_y)
\]

\[
= \sup_{x', y' \in D_B} \left\{ \inf_{x, y \in D_B} \left\{ X(n_x - x' + x, n_y - y' + y) - B(x, y) \right\} + B(x', y') \right\}
\]

\[
= \sup_{x', y' \in D_B} \left\{ \inf_{x, y \in D_B} \left\{ X(n_x - x' + x, n_y - y' + y) - B(x, y) \right\} + B(x', y') \right\}
\]

\[
= \left\{ \inf_{x, y \in D_B} \left\{ X(n_x - x' + x, n_y - y' + y) - B(x, y) \right\} + B(x', y') \right\}, \quad (B1)
\]
where \( h(x', y', x, y) \) is given by Eqn (26). This shows Eqn (25).

If \( R_{ij}(n_x, n_y) \) is the row-wise infimum of the \( (M_j + i)^{th} \) row of the matrix \( X(n_x, n_y) + H \), then

\[
R_{ij}(n_x, n_y) = \inf_{k,l} [X(n_x - i + k, n_y - j + l) + h(i, j, k, l)]
\]

\[
= \inf_{k,l} [X(n_x - i + k, n_y - j + l) + B(i, j) - B(k, l)]
\]

\[
= \inf_{k,l} [X(n_x - i + k, n_y - j + l) - B(k, l) + B(i, j) - B(i, j - 1)]
\]

\[
= \inf_{k,l} [X(n_x - i + k, n_y - j + l) - B(k, l) + B(i, j) - B(i, j - 1) + h(i, j - 1, k, l)]
\]

\[
= R_{ij-1}(n_x, n_y - 1) + h(i, j, i, j - 1),
\]

for every \( i = 0, 1, \ldots, M - 1 \) and \( j = 1, 2, \ldots, M - 1 \). This shows Eqn (31).