Software for 2D and 3D Mathematical Morphology

Richard Alan Peters II
Department of Electrical Engineering
Vanderbilt University School of Engineering
Nashville, TN 37235
(602) 322-7924
rap2@vuse.vanderbilt.edu

24 March 1995

Abstract

Mathematical morphology is a powerful tool for image analysis and enhancement. Morphological operators are shape-dependent, nonlinear image transforms such as erosion, dilation, opening, closing, and rank filters. This paper describes c-language software for mathematical morphology that operates on 2D or 3D images. The software performs many of the possible morphological operations on both binary and 8-bit grayscale images. It performs set-set, function-function, and function-set operations. It will either generate common structuring elements to a user's specification, or it will permit the user to specify a structuring element of arbitrary shape, size, connectivity, and origin.

1 Introduction

Mathematical morphology is the name of a specific collection of set theoretic operators defined on an infinite lattice. These operators, first examined systematically by Matheron [7] and Serra [9, 11] in the 1960's, are an extension of Minkowsky's set theory. The operators are defined on – and themselves form – an infinite lattice. They are especially useful for image analysis and image enhancement. There are, in the general literature, a number of good tutorials on image morphology [1, 3, 4, 5, 6, 10] where the reader may find the fundamentals.

Morphological operators (MO) include, erosion, dilation, opening, closing, rank filters (including median filters), tophat transforms, lower-upper-middle filters, and generalized correlations. These operations can be defined on binary or grayscale images in any number of dimensions. A MO is governed by a small pseudo image called a "structuring element" (SE). When applied to an image, the MO returns a quantitative measure of the image's geometrical structure in terms of the SE. This measure can be used to isolate features in an image or to construct a synthetic image containing regular approximations of the features in the original.

Mathematical morphology extends to the analysis of n-dimensional images. Applied to three dimensional voxel images such as the output from a CT, MRI or PET scanner, 3D MOs are completely analogous in effect to those applied to 2D imagery. If one takes a time-sequence of 2D images to be a 3D spacetime image, then the properties of spacetime can be exploited by morphological processing for the analysis and enhancement of moving imagery.

2 Program Overview

There are two programs, *Morph* for 2D images, and *Morph3d*, for 3D images or 2D time sequences. These c-language programs operate on Sun rasterfile format images, and call subroutines *MorphSub* and *Morph3dSub* to perform the morphological operations.

Morph creates (digital) disk-shaped binary or grayscale structuring elements to the user's specification. *Morph3d* creates binary or grayscale digital spheroids, cylinders, or cones as SEs to the user's specification. The user may also define an SE externally that has arbitrary size, shape, connectivity, or origin location.

Both of the programs implement the following MOs: erosion, dilation, opening, closing, hit-or-miss, rank filter, tophat, bothat, LUM, LUM smoothing filter, LUM sharpening filter, and shape-specific isolated binary deletion. The 3D program implements minmax and maxmin operators specifically for image time-sequences.

There are three fundamental types of operations in mathematical morphology. These are described by Maragos as set-set operations, function-function operations, and function-set operations [4]. Both *Morph* and *Morph3d* perform all three types of operations.

2.1 Design philosophy

This software is an experimental tool for the use of mathematical morphology in image processing. The programs implement fundamental operations with as much generality as possible. Since many complex morphological image operators are simply arithmetic or logical combinations of simpler ones, this design permits quick implementation and evaluation of new algorithms. Because of their generality, the programs are not as fast as they could be in specific configurations. For example, if one restricts the MOs to rectangular neighborhoods, then faster algorithms exist.

To implement a suite of operators, there are two distinct approaches. One is to write a separate program for each operator. The other is to create one program that implements them all. The former approach is the vogue these days since it facilitates the modular design of software. However, these programs use the latter approach. Of the code necessary to implement an MO, perhaps 90% is devoted to setting up the SE, the image or images, and the buffers necessary to perform the operation, as well as the bookkeeping to keep track of everything. That 90% of the code is nearly identical for every operator. Thus, one program is much more space efficient than many.[1]

[1] A user who simply cannot bear to have a single routine perform more than one operation can easily write a simple front

2.2 Operation types

Morph and *Morph3d* expect structuring elements to be either binary or grayscale and to have a specific pixel designated as the origin. The specifications of the default SE shapes will be given in section 3.

A structuring element comprises an array of numbers called "members". To facilitate shape and connectivity specification, all members are either active or inactive. Members specified by negative numbers are inactive. Those specified by strictly greater than zero numbers are active. In a grayscale SE, zero members are active. They are inactive in a binary SE applied to a grayscale image. If a binary SE is applied to a binary image, the result is a hit-or-miss transform. The strictly greater than zero members define the "hit" portion of the SE and the zero members define the "miss" portion.

The programs implement the three fundamental operation types as follows: Function-function operations are MOs on grayscale images using grayscale SEs with nonzero active members. Set-set operations are MOs on binary images using either binary SEs or grayscale SEs. Function-set operations are MOs on grayscale images using either binary SEs or grayscale SEs whose active members are all zero.

2.3 2D morphology

Morph performs two-dimensional mathematical morphology on 8-bit grayscale orbinary images. The program presents a command-line interface, decodes the user specified options, sets up the arguments for subroutine *MorphSub*, and reads and writes images in Sun rasterfile format. *MorphSub* performs all the morphology. Users wanting to do morphology on image formats other than Sun rasterfiles, can write a program, like *Morph*, that calls *MorphSub*.

2.3.1 Morph

Morph, after parsing the argument list, reads in the image file. If the file has a color map, *Morph* converts the image to an 8-bit per pixel luminance map. It calls *MorphSub* with the necessary arguments and then writes the resultant image.

2.3.2 MorphSub

After being called, subroutine *MorphSub* verifies that its arguments make sense. Then it either makes an SE to user specification or reads in the appropriate SE from a file specified in the argument list. Based on the arguments, it chooses a specific MO to perform. Then it calculates the size of the internal image necessary to perform the given operation with the given structuring element. The size is a function of the input image size, whether or not zero padding was called for in the argument list, the SE size and origin placement, and the particular operation specified. Function-function operations can return negative pixel values, so for these

end to the program for every operator.

operations, *MorphSub* must allocate 16-bit signed image buffers, otherwise it assigns 8-bit unsigned buffers. Then *MorphSub* calls the appropriate MO.

Morphological opening is an erosion followed by a dilation; closing is dilation followed by erosion. The tophat transform is the original image minus its opening; the bothat is the closing minus the original. These 4 operations require the successive application of two MOs. Thus, if one of them was specified, *MorphSub* must allocate an additional image buffer and perform the secondary MO. If tophat or bothat was specified *MorphSub* performs the necessary subtraction.

When it performs a 16-bit operation, *MorphSub* moves the results to an 8-bit buffer. Depending on the setting of a user specifiable switch, it either scales the result or it clips values outside the range $[0, \ldots, 255]$. It copies the result to the output buffer specified in the argument list, cleans up by freeing all memory it has allocated, and exits.

If the user has specified that the result is to be scaled, and if the range of gray levels in the result is less than one byte but outside of the interval $[0, \ldots, 255]$, the routine subtracts the minimum pixel value in the image from all the pixels. If the range is greater than one byte, the minimum is subtracted from each pixel and the the difference is scaled by the ratio 255/(MaxPixVal-MinPixVal).

There are 5 (each) erosion and dilation algorithms from which *MorphSub* selects, depending on its arguments.

1. *BinBinX* for "set" operations on a binary images using a binary SE;
2. *BinGrayX* for "set" operations on binary images using a grayscale SE;
3. *GrayBinX* for "set" operations on a grayscale images using a binary SE;
4. *GrayGraySetX* for "set" operations on grayscale images using a grayscale SE; and
5. *GrayGrayFctX* for "function" operations on grayscale images using a grayscale SE.

where X is either *Erode* or *Dilate*. (Operations, open, close, tophat, and bothat are constructed from these.) The choice of algorithm depends on four arguments: the operation (erode, dilate, open, close, rank, etc.), the operation type (set or function), the SE type (binary or grayscale), and the image type (binary or grayscale).

The following conditions supersede the arguments: All automatically generated SEs are grayscale. If all the active members in a grayscale SE are zero, then the operation is of type "set." If the input image is binary, then the operation defaults to "set" even though the SE might be grayscale with nonzero members. If the requested MO is one of the LUM filters, the image is treated as grayscale; the SE can be of either type with support specified as above.

There are separate binary and grayscale rank filters. *MorphSub* computes the active support (area in pixels) of the SE whenever a rank filter is requested. If the rank value equals the support size, then rank is equivalent to erode. The subroutine selects the appropriate erosion algorithm instead of the rank operator because it is faster. For similar reasons, if the rank requested is 1, the subroutine chooses a dilation algorithm.

2.4 3D morphology

As with the 2D program, *Morph3d* calls a subroutine, *Morph3dSub* to do the morphological operation. This permits the actual morphology routine to be independent of any image format. Given the limitations of current technology, it is often not possible to load an entire 3D image (or time-sequence of 2D images) into the memory of a workstation. Thus, *Morph3dSub* "pushes" a sequence of 2D images through the memory of the machine using "call-back" procedures supplied by *Morph3d*. The subroutine keeps in memory only the minimum number of images necessary for the creation of one output image.

To use *Morph3dSub*, a host program such as *Morph3d*, must pass it the addresses of an image input routine and an image output routine. The input routine reads the next file in the image sequence and puts the result in a buffer specified by *Morph3dSub*. The output routine writes the next image file on command. *Morph3dSub* uses these routines to "call back" to the host program for the next image in the sequence or to request that an image be output. (Section 2.4.3 describes the call-back procedures in more detail.)

Because the host program reads and writes the image files, it may need to keep track of the correspondence between input files and output files to appropriately name the latter. Due to initial conditions, *Morph3dSub* will usually request a number of input files before it requests an output. Thus, the host program may need to know the delay between input and output. This delay is dependent on the MO to be used, the z-dimension, of the SE and the z-coordinate of the SE origin. Two companion routines, *GetSE* and *CalcDelay*, provide this information. *GetSE* loads the SE from a user specified file or builds the SE from user specifications. It returns, among a number of parameters, the SE information needed for the delay calculation which *CalcDelay* performs.

2.4.1 Morph3d

Morph3d does the following: After parsing the command-line argument list, it calls *GetSE* and *CalcDelay*. Then it allocates and initializes a name list to keep track of the file names of the images currently in memory. It reads header information from the first input image for use in the I/O call-back routines. Then it calls *Morph3dSub* with the appropriate arguments. Upon completion, *Morph3d* frees some buffers and quits.

2.4.2 Morph3dSub

Morph3dSub does the following: After validating the arguments and initializing some variables, it selects the appropriate MO routine given the arguments. Similar to *MorphSub*, the subroutine determines the wordlength and dimensions of the intermediate buffers and allocates them. If the MO is (or requires) an open or close then there is a secondary MO, so the routine allocates a secondary set of buffers.

MorphSub3d pads the input image sequence with zero images at the beginning and the end depending on the MO and the z-coordinate of the SE origin. It does this so that the number of output images equals the number of input images. Zero images are prepended in sufficient number so that the first output image is computed when the first (non-pad) input image is pushed to the z-coordinate origin of the input (or

secondary) buffer. The subroutine computes four values to control this, two input delays and two output delays, one each for the primary and secondary buffers.

The main sequence processing loop proceeds as follows: If the end of the sequence has not been reached, the routine calls back for the next image. The call-back procedure may find that the sequence is at an end and return a flag so indicating. If the end-of-sequence flag is set, and some images have been processed, the routine increments the first end counter and checks to see if it has passed the first end delay. If it has, and there is no secondary MO, or if there is a secondary MO but the routine has also passed the second end delay, then the routine exits the the main loop. Otherwise, the routine scrolls the image into the primary image sequence buffer and increments the input count. (Notes: The buffer is a FIFO, so the image at the other end is discarded. If the end-of-sequence has been reached, a zero image is scrolled in.) If the input count is less than the first input delay, the routine returns to the top of the loop.

If the routine has met the first delay, it executes the first MO. If there is no secondary MO, the routine copies the result to the output buffer, calls back to the host program for image output, and returns to the top of the loop. If there is a secondary MO, it scrolls the output of the primary output buffer into the secondary input buffer. If the routine has not progressed beyond the second delay, it goes to the top of the loop. Otherwise, it does the secondary MO, copies the result to the output buffer, calls back for output, and returns to the top of the loop.

After the subroutine reaches the end of the sequence, goes beyond the end delays, and exits the loop, it deallocates all its buffers and returns.

2.4.3 I/O call-back procedures

The call-back procedure approach to image input and output lets a user write procedures to do file I/O on any image format he or she wants and use them with *Morph3dSub*. The user's host program passes the addresses of these two call-back procedures to *Morph3dSub* during its function call. Let *ImageIn*() and *ImageOut*() be the user-written input and output procedures. Then the host program calls *Morph3dSub* as follows:

```
Morph3dSub( MorphOp, ImageIn, InParams, ImageOut, OutParams, ...
```

InParams and OutParams are pointers to structures that contain all the data necessary for *ImageIn*() and *ImageOut*() to work, yet about which *Morph3dSub* does not need to know.

Morph3dSub expects the call-back procedures to have the following structure:

```
int ImageIn( In, NumInReqs, eof, InStruct )

    byte *In;

    int NumInReqs;

    int *eof;
```

struct IOS *InStruct;

{

1. Figure out the name of the next input file;
2. Read the next image file and set *eof = FALSE; (If there is an error in the read, return the error.)
3. If the file cannot be read be cause there are no more images set *eof = TRUE and return without error;
4. Decode the image and extract the luminance information from image.
5. Copy the grayscale image in row major order, one byte per pixel, *to* consecutive locations starting at In;
6. Return without error.

}

and

int ImageOut(Out, NumOutReqs, OutStruct)

byte *Out;

int NumOutReqs;

struct IOS *OutStruct;

{

1. Figure out the name of the next output file;
2. Copy the grayscale image in row major order, one byte per pixel, *from* consecutive locations starting at Out;
3. Construct the output image in appropriate format;
4. Write the image to the file;
5. If the write was successful, return with no error. (Else return with an error.)

}

The first argument, In, of *ImageIn* is a pointer to an image array. *Morph3dSub* allocates this array and passes its address to *ImageIn* through this variable. NumInReqs is the number of input requests that have been made by *Morph3dSub*. *Morph3dSub* calls *ImageIn* the first time with NumInReqs = 1. The third

argument, eof, is a pointer to a flag in *Morph3dSub*. *ImageIn* writes a zero there whenever it inputs an image. If there are no more images in the input sequence, *ImageIn* writes a nonzero number there.

InStruct is a pointer to a structure that includes information necessary for *ImageIn* to work but is not directly needed by *Morph3dSub*. InStruct is *ImageIn*'s link back to the host program. For example, InStruct may contain image header and colormap information, filename tags, or information to be conveyed to *ImageOut*.

The first argument, Out, of the *ImageOut* is a pointer to an image array. *Morph3dSub* allocates this array and passes its address to the *ImageOut* through this variable. NumOutReqs is the number of output requests that have been made by *Morph3dSub*. *Morph3dSub* calls *ImageOut* the first time with NumOutReqs = 1. OutStruct is similar to InStruct in the input call-back routine. It can be convenient to have OutStruct = InStruct.

Both call-back procedures return a functional value. A zero returned indicates normal completion of the routine. A nonzero value indicates an I/O error that should abort the entire operation.

3 Structuring Element Specification

As described in section 2.3.2, all structuring elements are either grayscale or binary, have active members, and may have inactive members. This permits a user to define SEs of arbitrary shape, size, connectivity and origin location. Such user-defined SEs are supplied to the programs as ASCII files that include the size of the bounding rectangle, the origin location, and the SE members themselves in row-major order.

Morph (through subroutine *MorphSub*) provides three predefined SEs and one user specifiable family of SEs. These comprise the most often used SEs for most applications. The predefined SEs are all grayscale with graylevel zero. They are a 3-by-3 square, a 3-by-3 plus (square with 4 corners missing), and a 5-by-5 plus (square with 4 corners missing). The user specifiable SE is a graylevel SE with spatial support in the shape of a digital disk (or ellipse). The user may specify the x and y dimensions of the disk and the graylevel of its center member. If the graylevel is zero, then all the active members are set to zero so that the routine executes a function-set or a set-set MO. If the center graylevel is nonzero, the SE members are assigned so that a plot of the graylevels along an axial cross-section of the SE is a half-ellipse. If a and b are the x and y pixel dimensions of the disk, if g is the specified graylevel, and if i and j are the x and y pixel indices, then the graylevel, $s(i,j)$, of an active member is given by

$$s(i,j) = g * \sqrt{1 - \left(\frac{a-2i}{a}\right)^2 - \left(\frac{b-2j}{b}\right)^2} \tag{1}$$

whenever the the square root exists. When it doesn't, $s(i,j) = -1$; the pixel is flagged inactive. Thus, the SE is dome shaped in graylevel and the routine executes a function-function MO.

Morph3d provides three predefined and three user specifiable SEs through subroutine *GetSE*. The predefined SEs – all grayscale with graylevel zero – are a 3-by-3-by-3 cube, a 3-by-3-by-3 plus (cube with 8 corners

missing), a 5-by-5-by-5 plus (cube with 8 corners missing). The user specifiable SEs include a spheroid, a cylinder, and a double cone. The names refer to the 3D spatial support of the SEs. The double cone has an hourglass shape.

The sphere is most useful for volumetric processing, whereas the cylinder and cone are useful for time sequences. The axes of both the cylinder and the cone are parallel to the z-axis (which is the time axis in an image sequence). The cylinder traces the same spatial neighborhood in each image in a time sequence. A cylinder of radius one tracks the time series of a single pixel. The cone applies successively larger neighborhoods to progressively more distant images. This permits velocity dependent operations. For example, a grayscale closing with a a-by-b-by-c cone will track pixels in a time sequence that have a maximum velocity in (x, y) given by $(a/c, b/c)$ pixels per frame.

For each of the three SEs, the user specifies x, y, and z dimensions and a graylevel. If the graylevel is zero, all the active members in the SE are zero and the routine executes a set-set or function-set MO. If the graylevel is nonzero, the result is different for each of the three SE types. The spheroid is assigned graylevels using

$$s(i,j,k) = g * \sqrt{1 - \left(\frac{a-2i}{a}\right)^2 - \left(\frac{b-2j}{b}\right)^2 - \left(\frac{c-2k}{c}\right)^2}, \tag{2}$$

whenever the the square root exists. When it doesn't, $s(i,j,k) = -1$; the pixel is flagged inactive. Letters a, b, c and i, j, k, are the x, y, and z dimensions and indices, respectively; g is the user specified graylevel and s is the graylevel of the SE pixel. The cylinder and the cone are assigned graylevels using equation (1) for every z-plane in the SE. However, the cone differs from the cylinder in that its xy cross sectional diameter changes in each z-plane. The x and y diameters, a_k and b_k, of the cone in the kth z-plane are a function of c given by

$$a_k = \frac{a-1}{r(c)} \mid k - r(c) \mid + 1, \tag{3}$$

$$b_k = \frac{b-1}{r(c)} \mid k - r(c) \mid + 1. \tag{4}$$

where $r = \text{int}(c/2)$.

4 Individual Operator Specifications

This section contains descriptions of the algorithms which implement the operators. The following attributes are common to all of the algorithms: Each performs a simultaneous rasterscan of the input and output images. For each output pixel, the routine processes input pixels within a corresponding rectangular (or in the case of 3D, parallelpiped) neighborhood. That neighborhood is the bounding box of the SE. Within each neighborhood, the routine performs a rasterscan and processes only those pixels that correspond to active SE elements. To avoid modulating operations at image boundaries, the programs zero pad the input buffers.

4.1 Erosion

Within both the 2D and 3D image morphology programs there are five erosion operators. (See section 2.3.2.) The exact algorithm depends on the type of image, operation, and SE.

In its most general form, the set erosion of a black and white, binary image is a hit-or-miss transform. Let I be the input image, X the set of white pixels in I, and let $X' = X \setminus I$ be the set of black pixels. Let O be the output image. Let $Z = A \cup B$ (such that $A \cap B = \emptyset$) be the SE with A as the hit portion and B as the miss portion. Let Z_p be the translation of Z so that its origin lies at pixel $p = (i, j)$. For each p, the result of the transform is white only if the *hit* portion of the SE covers only white pixels, and the *miss* portion covers only black pixels. That is, let p be a pixel location, then

$$O(p) = \begin{cases} \text{white}, & \text{if} \quad A_p \in X \text{ AND } B_p \in X' \\ \text{black}, & \textit{otherwise}. \end{cases} \tag{5}$$

As described in section 3, the active members in a binary SE are zeros and ones. Zeros delineate the miss area and ones, the hit area. *BinBinErode* computes an exclusive-or between the active members of the SE and the corresponding neighborhood pixels. The routine breaks out of the neighborhood loop (indexed on (i, j)) with a result of black the first time that $Z_{(x,y)}(i, j)$ XOR $I(x + i, y + j)$ is true. If the exclusive-or is never true, then the result for the neighborhood is white. Note that this is valid even if the miss portion or the hit portion of the SE is empty.

The erosion of a binary image with a grayscale SE is necessarily "hit only" – a standard set-set erosion. The first time an active member of the SE is found to cover a black pixel, *BinGrayErode* breaks the neighborhood loop and outputs a black pixel. If all active elments cover white pixels, the result is a white pixel.

Both *BinBinErode* and *BinGrayErode* have a *not* option. In *BinBinErode*, the effect is to delete white features with the shape of the "hit" portion of the SE. In *BinGrayErode* the effect is to delete the interiors from sets of white pixels in a binary image. This is performed as follows: if the exclusive-or is true, rather than returning a black pixel, the original pixel is passed. Otherwise, if every "hit" hits and every "miss" misses, a black pixel is passed (rather than white in the usual computation). If I is the original (binary) image and E is the eroded image, then the *not* flag causes I AND (NOT E) pixelwise to be output.

The set erosion of a grayscale image is the simply the minimum graylevel in the neighborhood defined by the active SE members. If a zero is encountered during the neighborhood scan, the scan terminates, since zero is necessarily the minimum. *GrayBinErode* and *GrayGraySetErode* differ only in the definition of which SE pixels are active (see sec 2.2).

The function-function erosion of a grayscale image is the minimum of the pixelwise difference between the input image and the active portion of the SE.

4.2 Dilation

There are a number of equivalent definitions of the dilation operator. An algorithm is implicit in each. All five of the dilation algorithms in *Morph* and *Morph3d* use the *reflected* structuring element, $\hat{Z}$, which is the SE rotated by 180 degrees (in both xy and yz planes in the 3D case). The routines execute this simply by indexing backward through the SE while performing a forward rasterscan within each rectangular neighborhood.

Routine *BinBinDilate* computes the following:

$$O(p) = \begin{cases} \text{white}, & \text{if} \quad \hat{A}_p \cap X \text{ OR } \hat{B}_p \cap X' \\ \text{black}, & \textit{otherwise}. \end{cases} \tag{6}$$

Thus the output pixel is white unless $\hat{A}_p$ is covered by *black* pixels *and* $\hat{B}_p$ is covered by *white* pixels. Within a neighborhood, the first time that $\hat{A}_{(x,y)}(i,j)$ AND $I(x+i,y+j)$ is true or $\hat{B}_{(x,y)}(i,j)$ AND NOT $I(x+i,y+j)$ is true, the routine breaks out of the loop (indexed on (i,j)) and returns a white pixel. If neither is true for any pixel, the result is black. Like the corresponding erosion, *BinBinDilate* is a hit-or-miss transform. Note that in most cases SEs with $B = \emptyset$ are used for set dilation.

The dilation of a binary image with a grayscale SE is necessarily "hit only" – a standard set-set dilation. The first time an active member of the reflected SE is found to cover a white pixel, *BinGrayDilate* breaks the neighborhood loop and outputs a white pixel. If all active elements cover black pixels, the result is a black pixel.

Both *BinBinDilate* and *BinGrayDilate* have a *not* option. In *BinBinDilate*, the effect is to delete black features with the shape of the "hit" portion of the reflected SE. In *BinGrayDilate* the effect is to delete the interiors from sets of black pixels in a binary image. This is performed as follows: if the result of the dilation of a neighborhood is white the original value of the image at the origin pixel is returned. If the dilation result is black, white is returned. If I is the original (binary) image and D is the dilated image, then the "not" flag causes I OR (NOT D) pixelwise to be output.

The set dilation of a grayscale image is the simply the maximum graylevel in the neighborhood defined by the active reflected SE members. If a white pixel is encountered during the neighborhood scan, the scan terminates, since white is necessarily the maximum. *GrayBinDilate* and *GrayGraySetDilate* differ only in the definition of which SE pixels are active (see sec. 2.2).

The function-function dilation of a grayscale image is the maximum of the sum of the input image and the active portion of the reflected SE.

4.3 Opening, closing, tophat and bothat

Opening and closing are performed through through simple iteration of erosion and dilation. Opening is erosion followed by dilation. Closing is dilation followed by erosion. These iterations are performed automatically by *MorphSub* and *Morph3dSub*. Tophat and bothat use the result of opening and closing.

The tophat transform is the original image minus the opening. The bothat is the closing minus the original image.

4.4 Rank filter

The output of a binary rank filter of order k at a given pixel is white if the the associated input neighborhood (as defined by active SE elements) contains at least k white pixels. The rank filter routines in *Morph* and *Morph3d* count white pixels during the neighborhood scan. When the kth is found the scan terminates and a white pixel is output. If k are not found during the scan a black pixel is output.

A grayscale rank filter of order k sorts the (active) neighborhood pixels into descending order by graylevel and outputs the kth in the list. (This pixel value is also called the kth order statistic.) Note that this approach could also be used to compute a binary rank filter. The grayscale rank filter routines in *Morph* and *Morph3d* scan a neighborhood and build a graylevel histogram. Then a running sum is tallied while indexing the histogram from the large-value end. The kth graylevel is the index at which the sum equals or exceeds k. If if the end of the histogram is reached before the sum equals k, black (gray-level 0) is output.

For both the binary and grayscale filters, valid ranks, k, range from 1 to N, the number of active members in the structuring element. When $k = 1$, the rank filter, whether binary or grayscale, is equivalent to an erosion. When $k = N$, the operation is equivalent to a dilation with the reflected SE. (That is, the operation itself is performed *without* reflecting the SE). When N is odd and $k = (N + 1)/2$, the rank filter is a *median* filter.

4.5 LUM filters

The Lower-Upper-Middle (LUM) filters are described by Hardie and Boncelet in [2]. There are three varieties, a smoothing filter, a sharpening filter, and a hybrid combination of them. They compare the neighborhood's center pixel to upper and lower order statistics from the neighborhood. Depending on the ordering, either the center pixel or one of the order statistics is output.

In the LUM smoothing filter, the center pixel is compared to the order statistics a distance k from the median. That is, if the SE has support of N pixels, and $m = (N + 1)/2$, then the mth order statistic is the median of the neighborhood, $m - k$ is the rank of the upper order statistic, and $m + k$ is the rank of the lower order statistic.

If the center pixel is less than the lower statistic, the lower statistic is output. If the center pixel is greater than the upper statistic, the upper statistic is output. If the center pixel is within the bounds, it is output directly.

The sharpening filter passes a center pixel that is out of the bounds of the order statistics. If the pixel is inside the bounds it passes the closer of the two order statistics.

The general filter compares the center pixel to order statistics of rank, $m - l$, $m - k$, $m + k$, and $m + l$, where $l > k$. If the center pixel is within $m - l$ and $m - k$ or within $m + k$ and $m + l$ it is output otherwise,

the closest of the 4 order statistics is output.

Hardie and Boncelet defined the filters as two-dimensional constructs but they are generalizable to n dimensions. The implementation in *Morph3d* is a straightforward extension.

Like the grayscale rank filter, the LUM filters compile a histogram for a neighborhood. They index through the the histogram from the bright end and accumulate to find the gray levels of the specified order statistics. As soon as the position of the center pixel with respect to the order statistics is known, the histogram accumulation stops and the appropriate value is output.

4.6 Minmax and maxmin

Minmax and maxmin were designed for use with time sequences. They are computed by *Morph3d* and have no counterpart in *Morph*. Consider a 3D structuring element to be a stack of 2D SEs. The 2D SEs delineate neighborhoods in adjacent sequential images. Maxmin finds the minimum pixel value in each 2D neighborhood and takes the maximum of those. Minmax takes the smallest maximum from among the 2D neighborhoods.

The idea behind minmax is this: Applied to a time sequence, a 3D structuring element demarcates a neighborhood area in successive images (which may or may not be the same in each image.) If a bright moving particle stays within the successive neighborhoods, the result of the minmax will be a bright pixel. If, on the other hand, the particle moves out of one of the neighborhoods, the result will be a darker pixel. Thus, when used with minmax, an appropriately shaped SE can act as a velocity filter. Maxmin is the same with with respect to dark particles.

5 Conclusion

Morph and *Morph3d* implement many of the possible morphological operators for the processing of 2D and 3D digital binary and grayscale images. They were written with the support of AFOSR grant F49260-88C-0053. The programs have been in development for three years and have been used extensively by a number of researchers. The software is available for anonymous ftp at image.vuse.vanderbilt.edu (129.50.100.16). The file is /pub/morph.tar.Z.

References

[1] Giardina, C. R. and E. R. Dougherty, *Morphological Methods in Image and Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[2] Hardie, R. C., and C. G. Boncelet, "LUM filters: A class of rank-order-based filters for smoothing and sharpening," *IEEE Trans. Signal Process.*, vol. SP-41, No. 3, March 1993.

[3] Haralick, R. M., S. R. Sternberg, and X. Zhuang, "Image analysis using mathematical morphology," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-9, No. 4, pp. 532-550, 1987.

[4] Maragos, P. and R. W. Schafer, "Morphological filters – part I: their set theoretic analysis and relations to linear shift invariant filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-35, No. 8, pp. 1153-1169, 1987.

[5] Maragos, P. and R. W. Schafer, "Morphological filters - part II: their relations to median, order-statistic, and stack filters," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-35, No. 8, pp. 1153-1169, 1987.

[6] Maragos, P. and R. W. Schafer, "Morphological Systems for multidimensional signal processing," *Proc. IEEE*, vol. 78, No. 4, pp. 690-710, 1990.

[7] Matheron, G., *Random Sets and Integral Geometry*, Wiley, New York, 1975.

[8] Ronse, C., *Why Mathematical Morphology Needs Complete Lattices*, Manuscript M-270, Philips Research Laboratory Brussels, Avenue Van Becelaere 2, Box 8, B-1170 Brussels, Belgium, November, 1988.

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

[10] Serra, J., "Introduction to mathematical morphology," *Comp. Vision, Graph., Image Process.*, vol. 35, pp. 283-305, 1986.

[11] Serra, J., ed., *Image Analysis and Mathematical Morphology, Vol. 2: Theoretical Advances*, Academic Press, New York, 1988.

[12] Sternberg, S. R., "Grayscale morphology," *Comp. Vision, Graph., Image Process.*, vol. 35, , pp. 333-355, 1986.