Machine Vision and Applications (1988) 1:23–40

Machine Vision and Applications

Pipeline Architectures for Morphologic Image Analysis

Lynn Abbott
Department of Electrical and Computer Engineering and The Coordinated Science Laboratory, The University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA

Robert M. Haralick and Xinhua Zhuang*
Department of Electrical Engineering, University of Washington, Seattle, WA 98195 USA

Abstract: The concepts of mathematical morphology provide some very powerful tools with which low-level image analysis can be performed. Low-level analysis, by its very nature, involves repeated computations over large, regular data structures. Parallelism appears to be a necessary attribute of a hardware system which can efficiently perform such image-analysis tasks, and there is a variety of forms that this parallelism can take. This paper gives a tutorial description of the basic morphological transformations and demonstrates how the basic morphological transformations can be implemented in the pipeline processing form of parallelism. Correspondingly, plausible designs for pipeline architectures are developed.

Key Words: image analysis, morphology, pipeline architectures, parallel processing, programmable delay memories

1. Introduction

Low-level image analysis involves large, regular data structures and often requires very fast rates of computation. Many different computer architectures have been proposed or built over the past decade for such image-processing tasks. In exploiting parallelism, pipeline architectures can sometimes offer advantages over arrays of individual processing elements.

Mathematical morphology is the study of shape. The tools of morphology can be used to realize inherent advantages of speed and flexibility for many image-processing applications (Gerristen and Verbeek 1984; Sternberg 1985). These "image algebra" tools are well matched to a pipeline architecture.

This paper presents a quick overview of the morphological approach to image analysis, and demonstrates how it relates to pipeline processing. Then, we present an overview of the concepts of mathematical morphology. Some existing architectures for implementing the morphologic operations are briefly described. Finally, some pipeline architectures for image morphological operations are sketched.

2. Background and Definitions

Images are often represented as two-dimensional arrays of brightness values, known as pixels. Such an array can be quite large; an image from the LANDSAT satellite can contain 4000 × 4000 8-bit pixels from each of several spectral bands (Reeves 1984). More common applications may involve video image of 256 × 256 pixels.

A *logical* or *binary* image is one in which each pixel consists of a single bit. By multilevel thresholding, any gray-level image can be converted to a registered stack of binary images (Preston 1983). These black-and-white images are transformed by logical rather than arithmetic operations. After processing, the resulting set of binary images can be converted back to gray-level format by arithmetic summations.

In a digital image, each pixel can be thought of as a cell. A set of nearby cells, known as a neighborhood, can be used to define transformation functions which map the given cell into a new one, which is part of a new, transformed image. Application of a set of such transforms to each pixel in an original image can result in a completely new image.

Cellular operations based on a 3 × 3 neighbor

* Permanent affiliation is with the Zheijung Computing Technique Institute, Hangzhou, People's Republic of China.

hood can be very powerful (Gernsten and Verbeek 1984). In certain cases, a sequence of 3×3 window operations can be equivalent to a single transformation using an arbitrary number of neighboring pixels (Lougheed and McCubbrey 1980). This paper is concerned with sequences of operations that may involve pixels which may be close together, such as those contained in a 3×3 neighborhood, or may involve pixels which may be many rows and columns apart. Operations using pixels which are spatially spread apart have important computational advantages in their ability to compose an operation sequence having N operations which, in effect, covers a neighborhood as large as 2^N pixels.

A *cellular logic computer* operates on binary images. Preston (1983) presents a survey of such machines. A *cellular array computer* usually refers to a two-dimensional array of processing elements; this is discussed further in section 4.1.

3. Overview of Mathematical Morphology

Morphology is the study of shape or form. Mathematical morphology, or *image algebra*, is the study of shape, using the tools of set theory. It is a branch of image analysis that uses set-theoretic descriptions of images and their transformations. This science was introduced by Matheron (1975). Serra (1982) presents these concepts in detail. A tutorial is found in Haralick et al. (1987).

Dilation and *erosion* are the primary transformations of mathematical morphology. In loose terms, these operations cause the swelling or shrinking of areas when the structuring element has a disklike shape. In the context of image analysis, the areas are usually portions of two-dimensional images, although the concepts of image algebra hold for arbitrary dimensionality.

We now present some definitions, based on an image X and a structuring element B, where each is contained in a two-dimensional Euclidean space, E^2. The relations generalize to k dimensions. We first consider binary morphological transformations, and then describe gray-scale transformations.

3.1 Binary Images

Dilation is defined as

$$X \oplus B = \{y | y = x + b, x \in X, b \in B\}$$

Similarly, erosion has the definition

$$X \ominus B = \{y | b \in B, \text{ implies } (y + b) \in X\}.$$

If X_b denotes the translation of X by the point b,

$$X_b = \{y | y = x + b, x \in X\},$$

then it can be shown that

$$X \oplus B = \bigcup_{b \in B} X_b$$

$$X \ominus B = \bigcap_{b \in B} X_{-b}.$$

This means that a dilation of X by B can be described as the union of translations of X by all points b contained in the structuring element B. For erosion, a similar description holds, except that the result is an *intersection* of translations by all points $-b$, where b is contained in B.

Dilation and erosion both exhibit the property of translation invariance:

$$X_b \oplus B = (X \oplus B)_b$$

$$X \oplus B_b = (X \oplus B)_b$$

$$X_b \ominus B = (X \ominus B)_b$$

$$X \ominus B_b = (X \ominus B)_{-b}$$

There exist chain rule relations for dilation and erosion:

$$(B_1 \oplus B_2) \oplus B_3 = B_1 \oplus (B_2 \oplus B_3)$$

$$(B_1 \ominus B_2) \ominus B_3 = B_1 \ominus (B_2 \oplus B_3)$$

This means that if

$$C = B_1 \oplus B_2 \oplus \ldots \oplus B_s,$$

then

$$X \ominus C = (\ldots [(X \ominus B_1) \ominus B_2] \ominus \ldots \ominus B_S)$$

$$X \oplus C = (\ldots [(X \oplus B_1) \oplus B_2] \oplus \ldots \oplus B_S)$$

Thus, if a structuring element can be broken down to a chain of dilations of smaller structuring elements, the desired operation may be performed as a strong of suboperations. This property lends itself well to pipelining.

If each B_i contains n points, then the dilation

$$C = B_1 \oplus B_2 \oplus \ldots B_S$$

is called an *n-point decomposition* of the structuring element C (Zhuang et al. 1986). If the origin is contained in each of these B_i, the decomposition is said to be *canonical*. Any structuring element admitting a canonical decomposition must contain the origin.

3.2 Gray-scale Morphology

We now generalize to gray-scale images, for which each pixel is not restricted to one of two values, but can contain any numerical value. The details of this generalization, which comes about through the umbra homomorphism theorem, can be found in Haralick et al. (1987).

If X now represents the gray-scale image, we refer to the value of the pixel at row r and column c by $X(r,c)$. Similarly, the value of the pixel at row i and column j of the structuring element B is $B(i,j)$. Then the gray-scale dilation and erosion can be computed by

$$(X \oplus B)(r,c) = \max_{\substack{(i,j)\in B \\ (r,c)-(i,j)\in X}} \{X(r - i, c - j) + B(i,j)\}$$

$$(X \ominus B)(r,c) = \min_{(i,j)\in B} \{X(r + i, c + j) - B(i,j)\}$$

All of the relationships presented for the binary case are preserved here in a form in which intersection is replaced by max and union is replaced by min.

3.3 Synopsis

Why are these operations useful? First, morphology is especially useful because its operations relate directly to shape (Haralick et al. 1987). By shrinking and swelling objects in some algorithmic fashion, we make transformations that can result in the detection of edges, corners, and other salient features of an object. Next, using the chaining property of morphological operations and decomposed structuring elements, pipelined architectures can perform these transformations very efficiently.

4. Existing Architectures

In an attempt to overcome the Von Neumann bottleneck inherent in conventional computer architectures, various forms of parallelism have been explored. This parallelism is especially important in image processing, in which very large, regular data structures are used to represent images.

Because of these large amounts of data, both *array processors* and *pipeline processors* have been proposed for image processing. Here we refer to a general array processing architecture which can have either SIMD (Single Instruction-stream, Multiple Data-stream) or MIMD (Multiple Instruction-stream, Multiple Data-stream) characteristics. This is often a two-dimensional array of identical, processing elements (PEs) which achieve *spatial parallelism*. A pipeline computer achieves parallelism by pushing the data through a "manufacturing assembly line" where each processor continually performs a particular operation on the data stream as the data stream passes through it, very much as a worker on the manufacturing assembly line does a particular assembly operation.

4.1 Array Processors

Examples of array processors that have been built are the Illiac IV, containing an 8×8 array of processing elements, and the MPP, containing 16,384 PEs in a 128×128 matrix (Reeves 1984), the CLIP processors, DAPP, GAPP, and the AISI machine with 1024 processors. In these designs, each PE is identical to all others in the system. A central control unit broadcasts instructions and data to all of the PEs. Reviews of these architectures and others can be found in Yalamanchili et al. (1985), Kruse (1983), Sternberg (1981), and Rice and Jamieson (1985).

Extremely high computation rates can be achieved by the array machines, but only after image data has been loaded into the array. Sternberg (1985) has contended that array machines spend significant amounts of time simply in loading and unloading image data. In other words, input and output of image data can present a serious drawback to these designs.

Typically, each PE in an array architecture will contain just 1 pixel, and can communicate with its nearest neighbors. This facilitates cellular neighborhood operations. But even the largest of the array processors can contain only a subset of a typical image. This requires that an image must be partitioned into several segments that are processed separately. This, unfortunately, causes undesirable effects at the borders of the segments, which must eventually be compensated for in the final transformations. It is argued that these border drawbacks, and the input/output (I/O) overhead, significantly reduce the effectiveness of the array architecture.

4.2 Pipelined Processors

These problems can be remedied to some extent in a pipeline architecture. Image data is presented in raster-scan format, as from a TV camera, into a pipeline of processing stages. In general, each stage in a pipeline can perform a limited number of functions; a control unit configures each stage before processing can begin. After configuration, a stage performs the same operations on each input, and passes the result to the next stage. The advantages of this approach are that the image data need not be put into a special format, and that only simple interconnections are required (Reeves 1984). Once

the pipeline is filled, very high throughput is achieved.

Some examples of pipeline architectures are the TAS (Klein and Serra 1973), the Cytocomputer (Lougheed and McCubbrey 1980), the MVI Genesis 4000 (Sternberg 1985), and the MITE (Kimmel et al. 1985). Each is now discussed briefly. The Texture Analyser System (TAS) performed binary morphological operations in a pipeline of three stages. Originally a single hexagonal structuring element was supported, but later versions could perform more general logical operations.

The Cytocomputer was developed at the Environmental Research Institute of Michigan, and has been used for biomedical image processing (Sternberg 1983). The Cytocomputer consists of a serial pipeline, in which each stage performs a single transformation of an entire image. Eighty-eight identical stages perform logical morphological operations, and 25 other stages follow for numerical processing. Each logical stage is fully table driven. Each logical stage can perform 3×3 neighborhood operations. In the numerical stages, operations are performed on gray-scale images.

The MVI Genesis 4000 (Sternberg 1985) provides several stages that are not all identical to perform arithmetic, geometric, and statistical operations. Image data passes through this pipeline at 10 Mbytes/s. Its image memory planes consist of several 512×512 buffers.

The arithmetic group can perform additions, multiplications, and so on, and can logically compare an image to another image or to a constant. The statistical group can count the occurrences of pixels at specified intensities or in specified windows, and can locate all pixels in a given stage. The geometric group performs morphological transformations, one or more in a single stage. Each stage can chain structuring elements that consist of two points each, where one of these points is the origin and the other is an arbitrary point in the image plane.

The MITE is an advanced architecture that has the possibility of multiple pipelines whose interconnectivity can be reconfigured.

5. Pipelined Architectures for Binary Images

We first consider the case of logical images. Section 6 will cover numerical images. Consider again the characterization for dilation and erosion:

$$X \oplus B = \bigcup_{b \in B} X_b$$

$$X \ominus B = \bigcap_{b \in B} X_{-b}$$

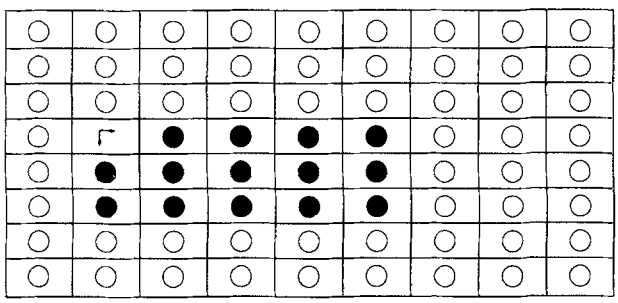
To implement these operations using a pipeline architecture, first consider the case of a two-point structuring element. We then generalize these results to n-point elements, and consider a computer pipeline composed of such stages.

5.1 Practical Considerations

5.1.1 The image window. Any practical image must be limited to some finite number of elements. This suggests some constraints that may be employed in the design of a processor architecture. If we assume an $N \times M$ rectangular image, having N lines and M pixels per line, we see that it is possible first to perform morphological operations assuming an infinite image plane, and then produce the final result by a windowing operation. Alternatively, this windowing operation can be performed concurrently with a morphological operation. With either method, this is equivalent to multiplying individual pixels by 1 within the $N \times M$ window, and by 0 outside, after performing the morphological operations. For a binary image, this is equivalent to ANDing by a rectangular window consisting of 1's upon a background of 0's.

Example: Here is an image window of dimensions 3×5, where a one is represented as "●" and a zero as "○." The origin is represented as "⌜." Conceptually, the zeros extend to infinity in all directions.

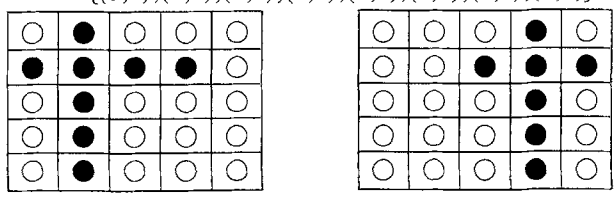
Consider now an image which has nonzero pixels only within a rectangular window. We wish to shift the image, and display the result within the original window.



An Image Window: W

Example: Here is an instance of translation, using a rectangular display with $N = M = 5$. Notice that as pixels are moved out of the image window, they are simply forgotten.

$A = \{(0,1),(1,0),(1,1),(1,2),(1,3),(2,1),(3,1),(4,1)\}$



A $\quad$ $A_{(0,2)}$

5.1.2 Raster-format image data. For a pipeline architecture, it is convenient to present image data in raster format. We now formalize this concept.

We define a function $R{:}E^2 \rightarrow E^1$ which maps an $N \times M$ image to raster format. For an image A, $R[A]$ represents a one-dimensional array of pixel value in row-major form. $R[A](t)$ refers to a particular pixel: by incrementing t from 0 to $NM - 1$ we are given, in order, the image pixels of A beginning in the upper left-hand corner of A and proceeding left to right, top to bottom. $R[A]$ is defined to be zero for all other values of t. $R[A](rM + c)$ is the value of the image pixel in the r^{th} row and c^{th} column of A, for $0 \leq r \leq N - 1$ and $0 \leq c \leq M - 1$.

We need to refer to a "delayed" raster image. This is represented as $R[A](t - \tau)$, where τ is the amount of delay. If the pixels of $R[A]$ are input one at a time into any kind of device that behaves like a shift register, we can refer to the input of the shift register as $R[A](t)$. Then $R[A](t - \tau)$ refers to the output of the shift register. The final version of a shifted image is just an appropriately delayed output of the shift register, which may then be masked to zero out those pixels which would have wrapped around onto another line.

For a raster image $R[A](u)$, we also define its *duration* to be the instances in time for which $0 \leq u \leq NM - 1$. Notice that the argument u may represent a delayed image, where $u = t - \tau$.

Example: Now consider the raster-format data for the previous image, A, as shown below. The first row is $R[A](t)$, the input image data for A. The next row is $R[A](t - 2)$, which is the original delayed by two time units. The third is the mask $R[M](t)$ with which we AND the second row to prevent wraparound. The fourth line is the image window $R[W](t)$. Finally, the last line is the translated image $R[A_{(0,2)}](t)$, which corresponds to $A_{(0,2)}$. This is formed by ANDing lines 2, 3, and 4.

Because this translation operation can be performed easily by delaying and masking raster-format image data, this yields a strategy for implementing pipeline morphological operations.

5.2 Two-Point Canonical Structuring Element

We consider first dilation, and then erosion. An architecture is then presented that can perform these operations, using raster-scan image data. Binary $N \times M$ images are assumed throughout.

5.2.1 Theory—dilation. *Proposition 1:* Dilation of an image by a structuring element consisting of just the origin and a point $(r,0)$ below it, where r is positive, can be realized by delaying the raster input by rM time delays, and ORing that with the input. The resulting output must be windowed by the duration of the nondelayed input image.

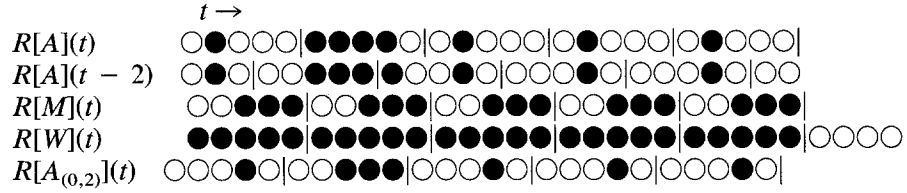
Proof: If we delay $R[A](t)$ by RM time units, we have $R[A](t - rM)$, which corresponds to a version of the original image that has been shifted down by r pixels. This is because a delay of the entire image by an integral number of raster lines r results in the entire image being shifted down by r pixels. If we OR $R[A](t)$ with $R[A](t - rM)$, a dilation is formed. We must AND this with $R[W]$ to produce the final, windowed result.

This is the simplest case, because no wraparound problems arise. A similar case follows.

Proposition 2: Dilation by a structuring element consisting of just the origin and a point above it $(r,0)$ where r is negative, can be realized by delaying the raster input by $(|r|)M$ time units, and ORing that with the nondelayed input. The output must be windowed by the duration of the *delayed* image.

Proof: As we saw before, a shift of the entire image by an integral number $|r|$ of raster lines results in the image just being shifted down by $|r|$ pixels. If we OR $R[A](t)$ with $R[W](t - |r|M)$, a dilation is formed. We must AND this with $R[W](t - |r|M)$, which corresponds to the duration of $R[A](t - |r|M)$, to produce the desired result.

This was an example of a *noncausal* operation, since the image *window* needed to be delayed. Recall that for raster input, an image is presented one pixel at a time. In general, we refer to a noncausal operation as one in which some pixel of the original raster image must be logically combined with an image pixel that has not yet been presented in the raster data. For example, when the pixel $R[A](t_1)$ is received, it needs to be combined with $R[A](t_1 + n)$, where n is positive. This pixel has not yet been received. Our solution to this dilemma is simply to delay the input image, and to define this delayed



Example:

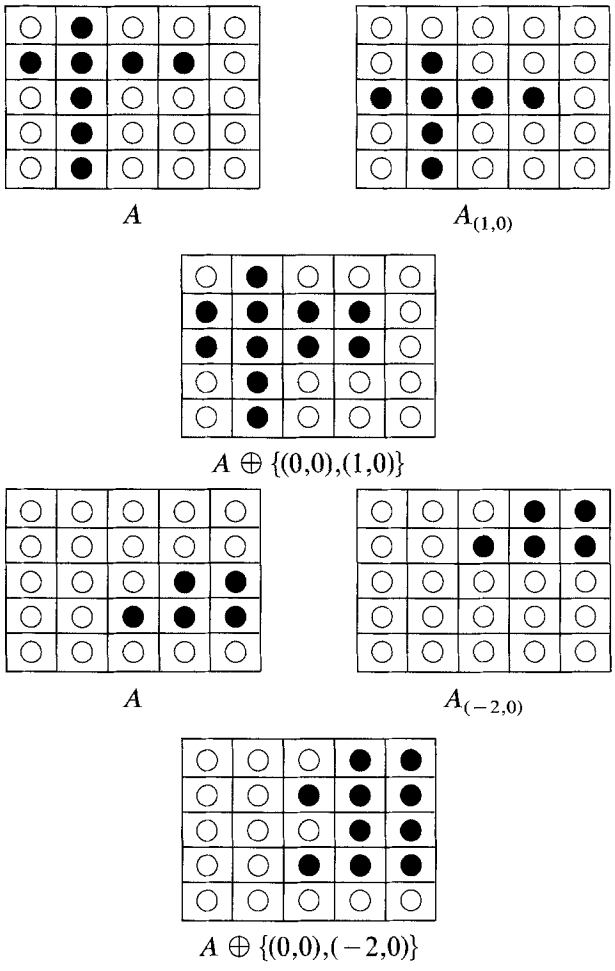


image as the *primary* image. A delayed image window $R[A](t - n)$ accompanies this image.

The window must be delayed since the original input effectively corresponds (in the noncausal case) to the translated image, $A_{r,c}$, for which $rM + C \leq 0$. If we were to window by the duration of the original input, we would essentially be using $W_{(r,c)}$ as our image window. By delaying $R[W]$ by $|rM + c|$ time units, we effectively recover $W_{(0,0)}$.

Similarly, a *causal* operation is one in which all logical operations with some input pixel of the original image involve only pixels which have already been presented in the raster data. The original image remains the *primary* image and $W_{(0,0)}$, without any delay, is used as the image window. These earlier pixels could be stored in a shift register for this operation, and would correspond to a shifted image $A_{(r,c)}$, for which $rM + c \geq 0$.

Notice that Propositions 1 and 2 involve vertical shifts only, and that no wraparound problems are present. Now we consider horizontal shifts, for which wraparound pixels must be masked.

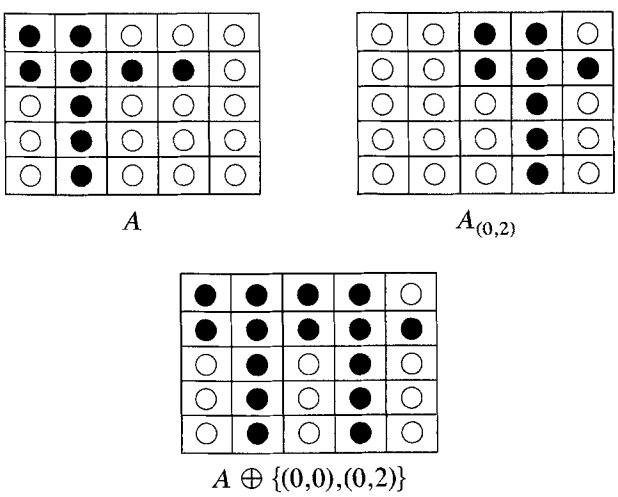
Proposition 3: Dilation by a structuring element consisting of just the origin and a point to the right of it $(0,c)$, where c is positive, can be realized by delaying the raster input by c time units, masking that, ORing that with the input, and windowing by the duration of the input image. The masking operation required here is to prevent wraparound; the first c pixels in each delayed raster are set to 0.

Proof: A simple delay of the input by c, for $0 \leq c \leq M$, results in a wraparound, so that some pixels which were on line k become elements of line $k + 1$. If for each line these c pixels are set to 0, a translated, masked image results. If this is ORed with the input, and windowed (ANDed) by $R[W](t)$, the required morphological dilation is performed.

Proposition 4: Dilation by a structuring element consisting of just the origin and a point $(0,c)$ directly to the left of it (c is negative) can be realized by delaying the raster input by $|c|$. The output is formed if we now OR this delayed input with a masked version of the (nonshifted) input, and AND this with an image window delayed by $|c|$ time units. The mask must set to 0 the last $|c|$ pixels in each (nondelayed) raster line.

Proof: This is another example of a noncausal operation. The delayed image is the primary image. Relative to the primary image $R[A](t - |c|)$, the nondelayed input $R[A](t)$ represents a negative delay of the image. If we constrain c to be in the range $0 < |c| < M$, there results a wraparound, so that some pixels which were on line k become elements of line $k - 1$. If these $|c|$ pixels are masked to 0, a left-shifted, windowed image results. If this is ORed with the delayed input, and ANDed with a delayed image window $R[W](t - |c|)$, the required morphological dilation is performed.

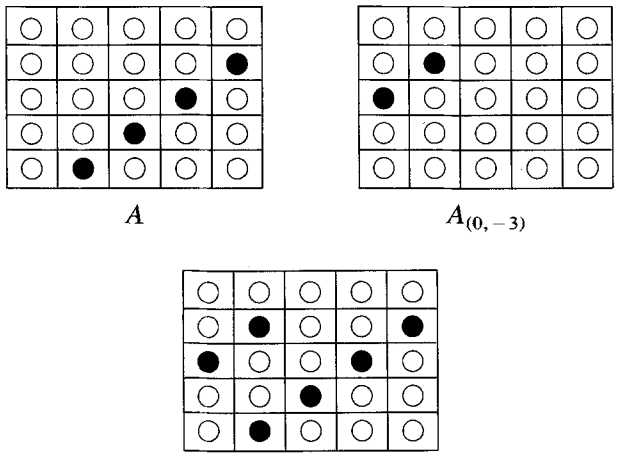
Now that we have examined several special cases, it is time to examine the general case.



Proposition 5: Dilation by a structuring element consisting of just the origin and an arbitrary point (r,c) in the image plane can be realized by delaying the raster input by $|Mr + c|$ time units. (Note that r and c can be independently positive, negative, or zero.) If the sum $Mr + c$ is positive or zero (the *causal* case), the output can be formed by ORing a delayed, masked version of the input with the nondelayed input, and then windowing by the duration of the original input. If the sum $(Mr + c)$ is negative (the *noncausal* case), the shifted image will become the primary image. If we OR this delayed image with a masked version of the (nonshifted) input, and AND this with the duration of the delayed input, the final result is produced.

Proof: Note first that $A_{(r,c)} = (A_{(r,0)})_{(0,c)}$, from the definition of translation of an image. Thus an arbitrary translation can be decomposed to two independent translations, one horizontal and the other vertical. To achieve the vertical translation $A(r,0)$, the input image can be delayed by $|rM|$, an integral multiple of M. This requires no masking except by the image window. Similarly, the horizontal translation can be accomplished by a delay of $|c|$, with appropriate masking, when we constrain c to be in the range $0 \leq |c| \leq M - 1$.

1. When $c \geq 0$ and $r \geq 0$, the sum of the delays is $|rM + C|$ as in the proposition. The image window $R[W](t)$ is not delayed.
2. When $c \leq 0$ and $r \leq 0$, or $c < 0$ and $R \leq 0$, the sum of the delays is still $|rM + c|$ as before. But the image window must be delayed as $R[W](t - |rM + c|)$ because, as we saw before, the image window needed to be delayed when r or c were independently less than zero.
3. When $c \leq 0$ and $r > 0$, we have two opposing terms. The rM-term requires that the image be delayed by $|rM|$ before combining with the input image, and that the image window $R[W|(t)$ not be delayed. The c-term would independently cause a delay of $|c|$ before logically combining the two, but it would cause the image window to be delayed also, becoming $R[W](t - |c|)$. Since the rM-term dominates, these opposing effects can be combined so that a total delay of $|rM + c|$ is used with an image window of $R[W](t)$.
4. When $c \geq$ and $r < 0$, the same types of opposing effects are seen, but with the signs reversed. The rM-term again dominates, and the total image delay required is $|rM + c|$. Since rM is negative, the entire term $rM + c$ is negative, and the sequence is noncausal. This requires that the shifted image window $R[W])t - |rM + c|)$ be used.



A $A_{(0,-3)}$

$A \oplus \{(0,0),(0,-3)\}$

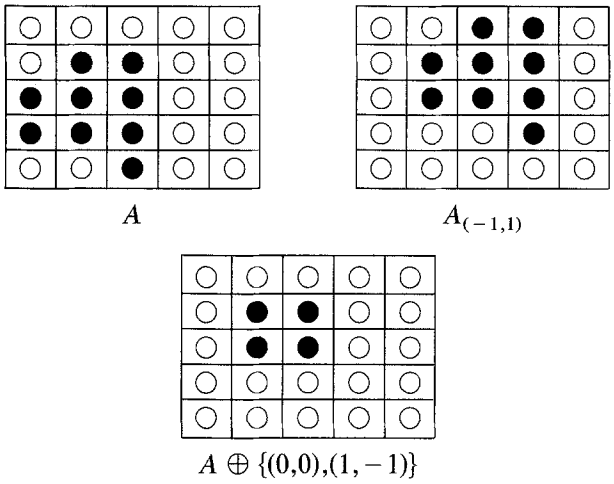
5.2.2 Theory—erosion. Because erosion is the morphological dual to dilation, many of the concepts derived above are applicable here. The differences are that the AND operation is used between shifted copies of the image instead of OR, and the relative shifts are reversed in direction from those for dilation. We proceed directly to the general case.

Proposition 6: Erosion by a structuring element consisting of just the origin and an arbitrary point (r,c) in the image plane can be realized by delaying the raster input by $|Mr + c|$ time units. Again, note that r and c can be positive, negative, or zero. If the sum $(Mr + c)$ is positive (here the *noncausal* case), the output can be formed by ANDing a masked version of the input with the delayed input, and then windowing by the duration of the delayed input. If the sum $(Mr + c)$ is negative or zero (the *causal* case), we now AND the nondelayed input with a masked version of the shifted input and then window by W.

Proof: For erosion, the shifts are in the $-r$ and $-c$ directions. As in the case of dilation, the arbitrary translation can be decomposed into two independent translations. The arguments of Proposition 5 hold here, by which the two image shifts are reconciled to require only $|rM + c|$ delays. The image window is determined by the sign of $rM + c$. When the sum is negative zero, $R[W](t)$ is used, and when the sum is positive, $R[W](t - |rM + c|)$ is used. After masking for wraparound and ANDing with the appropriate image window, the final output is seen. The wraparound mask must set to zero the $|c|$ pixels that would otherwise carry over to another image line.

5.2.3 Masking. Up to this point, we have often

Example:



mentioned the masking operation that sets to zero those pixels that have wrapped around to another image row because of image delays. It is time to consider this more closely.

We define (r,c) to be the amount by which an image is to be translated. For dilation, this is a point in the structuring element; for erosion, this is the *negation* of a point in the structuring element. We have seen that $|rM + c|$ is the amount of delay needed from the original image data. We have also seen that the sum $(rM + c)$ can represent two independent shifts, one vertical and one horizontal. Because of this fact, it is c that determines the wraparound mask.

If $c \geq 0$, mask off (set to zero) the first c pixels of each image row, relative to the primary image, of the nonprimary image. If $c < 0$, mask of the last $|c|$ pixels of each image row, relative to the primary image of the nonprimary image.

An example was given earlier for $c = 2$. Later, in section 5.3.3, an example will be given for both $c > 0$ and $c < 0$.

5.2.4 Implementation. A conceptual realization of the operations described above is shown in Figure 1. In all cases, the circuit performs logical operations on the input raster stream, and on a delayed version of it. The *input pixels* are presented one at a time in raster-scan format. A set of *delay elements* permits the choice of the appropriate input delay. This delay varies with the structuring ele-

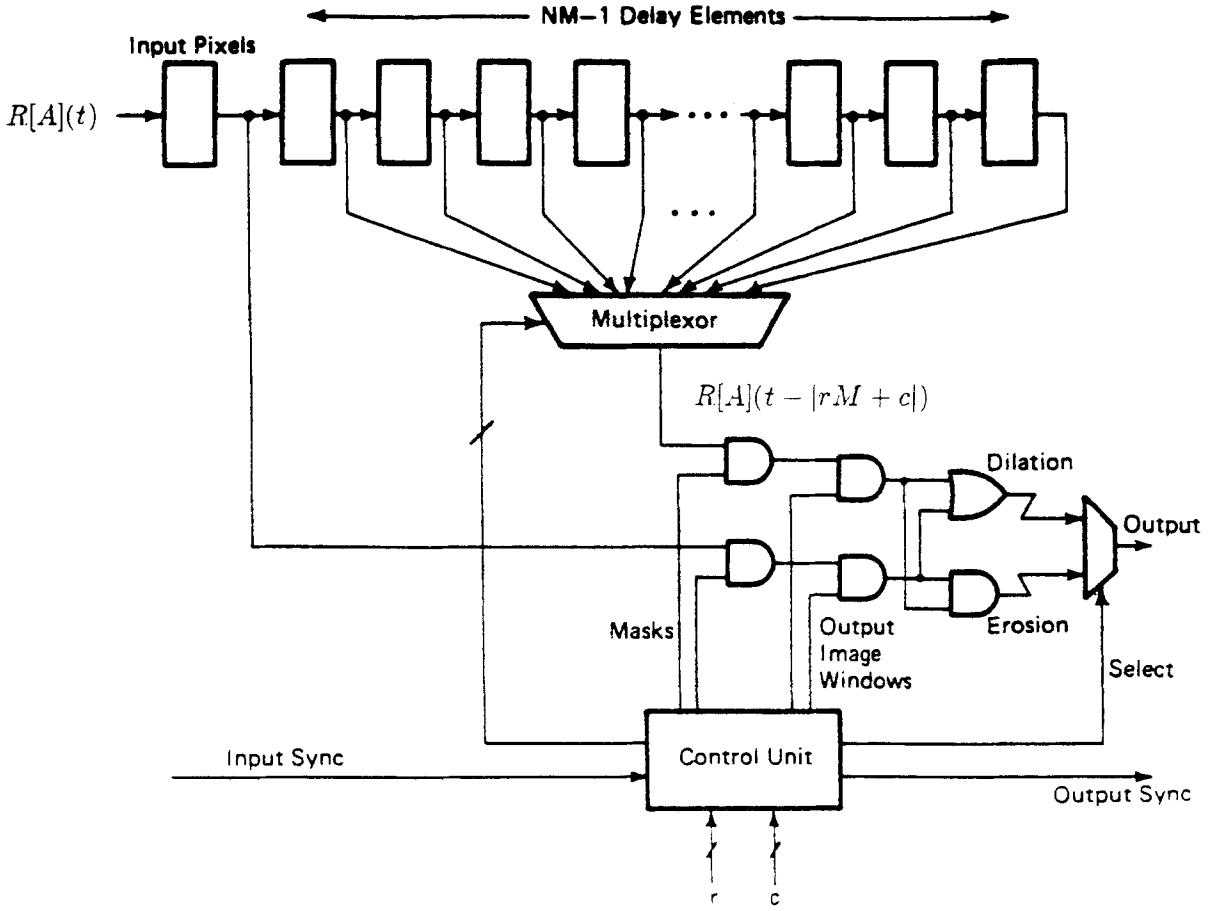


Figure 1. This illustrates a conceptual pipeline architecture for performing morphological operations.

ment, and is always $|rM + c|$. A *multiplexer* selects this delay. The first delay element is needed because of practical considerations. Clock signals are not shown in the figure.

A *control unit* provides the appropriate mask and window signals, based on (r,c) and on an *input sync* signal which indicates the start of a new image. An *output sync* signal is provided to specify the start of the output image. This is necessary for the window function because the output image may be delayed relative to the input.

Only one of the *masks* will operate for a given (r,c); the other will be set to 1. One of the masks is for the causal case, and one is for the noncausal case. Using knowledge of M, r, and c, the control units can produce the appropriate mask signal to prevent wraparound. The *output image windows* ensure that only the proper portion of the (relatively) delayed image is used in forming the output.

Of significance is the fact that this arrangement can be made to accept high-speed image data, even faster than the video rate acquisition speed. Binary data could be taken from a video camera, for example. One image frame can be followed immediately by another, and this circuit will perform the same transformation on both images. Because of this feature, this circuit can be used as a stage in a pipeline to perform chained morphological operations.

A problem is that the number of delay elements and the complexity of the multiplexer can become prohibitive. For example, a worst-case, 2-point structuring element is $\{(0, 0), (N - 1, M - 1)\}$. This would require NM elements in the delay line. If $N = M = 4000$, we would need 16,000,000 delay elements. Actually, for some applications, this may be possible with today's technology. This may be compared to the Cytocomputer, which utilizes $2M + 3$ delay elements per logical pipeline stage.

But a very real problem becomes evident if the complexity of the multiplexer is considered. The solution is to utilize a programmable tapped-delay memory, as shown in Figure 2. The control unit must set the amount of delay. In practice, this may be implemented with a random-access read/write memory, commonly known as a RAM. This memory unit can be viewed as a circular buffer. Input pixels are written into sequential memory locations, beginning again at the start of the memory after the end is reached. Appropriately delayed pixels are read from the memory at the same rate as new pixels are written.

With a sufficiently fast memory, the output pixels can be read before being overwritten by new input pixels. If the RAM has only one I/O port, it must be capable of operating at twice the rate of the data flow through it. This is because "read" and

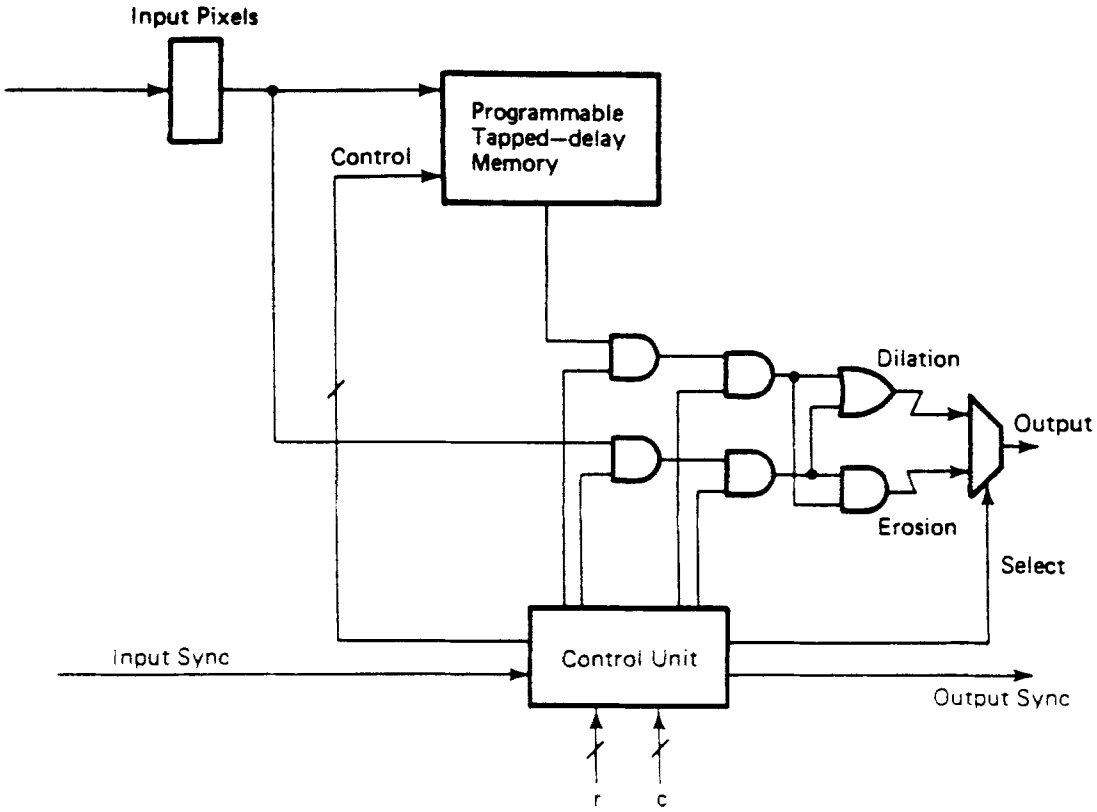


Figure 2. This illustrates the block diagram for a pipeline architecture with a programmable delay memory.

"write" operations are both required for each input pixel. Also, a dual-port RAM could be used to advantage here. For either case, the memory must contain at least $NM - 1$ memory cells, where each cell corresponds to an individual delay element.

All of this analysis has assumed the very worst case structuring element. If certain constraints are made on the structuring element, the circuit may be made simpler. For example, if we make the constraint $|r| \leq q < N - 1$, then a worst case structuring element is $\{(0, 0), (q, M - 1\}$, and at most $M(q + 1)$ memory cells are needed. Similarly, the constraint $|c| \leq s < M - 1$ may be added. In that case a worst-case structuring element is $\{(0, 0), (q, c)\}$, and the RAM is required to have only $qs - 1$ memory cells.

5.3 *n*-point Canonical Structuring Element

While the dilation of a chain of 2-point structuring elements can be equivalent to a large rather complex composite structuring element, some structuring elements *cannot* be decomposed into a chain of 2-point sets. For these applications, more points are needed in the decomposed structuring elements. For example, it can be shown that a chain of 3-point elements can be found which is equivalent to an arbitrary convex element. Here we consider the more general case of n-point, canonical structuring elements, in which each structuring element is composed of the origin and exactly $n - 1$ other points. We present first the theory, and then an implementation.

5.3.1 Theory—dilation. The theory here is a generalization of that for 2-point sets. We first look at the strictly causal case, then the strictly noncausal case, and finally the general case.

Proposition 7: Consider a structuring element composed of n points, where one of the points is the origin and the renaming points are arbitrary. Denote this set of points by $\{(r_i, c_i) | 0 \leq i \leq n - 1\}$. If for each point (r_k, c_k) the sum $(Mr_k + c_k)$ is positive or zero (causal), then we can take $n - 1$ delays of the input raster image, mask off the pixels that would wrap around to new image lines, OR all of these with the original image, and window by the duration of the input to produce the dilation.

Proof: An image can be effectively shifted within an image window by delaying and masking for wraparound. As we saw in Proposition 5, when the sum $(Mr_k + c_k)$ is positive or zero for a point (r_k, c_k), we can form a union of the image with a shifted version of itself by delaying the raster input, masking that for wraparound, ORing with the nondelayed input, and windowing by $R[W](t)$. When there are $n - 1$ points outside the origin, and each represents a causal delay, we can form the dilation of the image by this element by independently performing the appropriate delays, masks, and ORs for each point.

Proposition 8: Consider again a structuring element composed of n points, where one of the points is the origin. For every point (r_k, c_k), if the sum $(Mr_k + c_k)$ is negative (noncausal), then we can again take $n - 1$ delays of the input image. The point (r_p, c_p), for which the sum $|Mr_k + c_k|$ is maximum over all k determines the greatest delay needed, and this corresponding image becomes the primary image. All other delays are measured with respect to this image. For all other delayed images, and for the original image, wraparound masks must be applied; the resulting pixels are ORed with this primary image; and the delayed image window $R[W](t - |Mr_p + c_p|)$ is applied to derive the dilation.

Proof: As we saw in Proposition 5, when the sum $(Mr_k + c_k)$ is negative for a point (r_k, c_k), we can form a union of the image with a shifted version of itself by delaying the raster input, masking the original image for wraparound, ORing these together, and windowing by a shifted image window. The final union will be output in raster-scan format, but will be delayed relative to the original input. If there are $n - 1$ points outside the origin, and for each of them $(Mr_k + c_k)$ is negative, then the problem is a similar one. If we find the point (r_p, c_p) for which the sum $|Mr_k + c_k|$ (overall k) is maximum, then we have determined the maximum delay, $(Mr_p + c_p)$. Since this is a negative integer, this represents a negative (noncausal) delay of the original raster input. We form this noncausal delay by naming this shifted version the primary image, and measuring all delays relative to it. If all of the other delays are taken and masked for wraparound, we can OR the resulting shifted images with this primary image, and then window with $R[W](t - |Mr_p + c_p|)$ to form the desired dilation.

Now we proceed to the general case. Up to now, every structuring element has required only causal delays or only noncausal delays—never both. If both are permitted, a complication arises, since we need to accommodate delays in opposite directions, relative to some primary image. This will require the addition of extra delay elements.

Proposition 9: For an arbitrary, canonical, n-point structuring element, an entire image can be shifted into a long-shift register. The output of that will be the primary image, and will enter a second, long-shift register. By obtaining delays relative to this primary image from the two shift registers, we are able to form the dilation. Masks must be applied

to the outputs of the two shift registers; this is then ORed with the primary image, and an image window must be applied. This window is the duration of the output of the first shift register.

Proof: By entering the entire image into a long-shift register, and by centering the image window on the output of this register, we can form relative delays in both directions from the primary image. If noncausal points are present, then the correspondingly delayed images are available from the first shift register. Likewise, delayed images corresponding to causal points are available from the second long-shift register. Combining the results of Propositions 7 and 8, we see that the desired dilation is formed as described above.

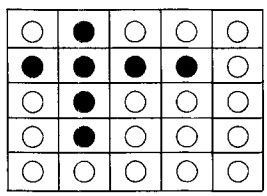
5.3.2 Theory—Erosion. We proceed directly to the general case.

Proposition 10: For a structuring element containing the origin and $n - 1$ arbitrary points, we can form the erosion of an image by shifting the entire image into a long-shift register. The output of that is the primary image, and it enters a second long-shift register. We obtain delays relative to this primary image from the two shift registers. Suitable masking must be applied to prevent wraparound, and these results are ANDed with the primary image. Finally, an image window is applied to produce the desired erosion.

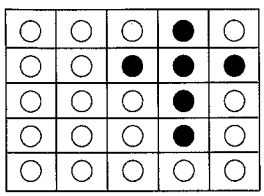
Proof: Recall that erosion differs from dilation only in that intersections replace unions, and that shifts are reversed in direction. Then, using the results derived so far, we have Proposition 10.

5.3.3 Implementation. Note that the 3-point structuring element

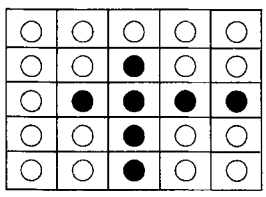
$$\{(0, 0), (N - 1, M - 1), (-[N - 1], -[M - 1])0\}$$



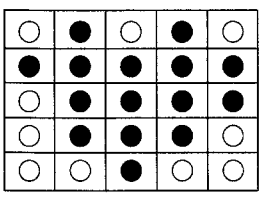
A



$A_{(0,2)}$



$A_{(1,1)}$



$A \oplus \{(0,0),(1,1),(0,2)\}$

is the worst case, in terms of the total number of delay elements required. This requires $2NM - 1$ delay elements. (In practice, this would be a very unusual structuring element.) For any $n > 2$, a hardware implementation must accommodate this as its worst case if completely arbitrary structuring elements are permitted.

Figure 3 shows a conceptual realization for a general, 3-point canonical structuring element. For clarity, multiplexers are shown. Note that nearly twice the number of delay elements are needed from previous implementations. Also, a multiplexer is needed for each point outside the origin *for each delay line*. This is because for a truly general structuring element, all $n - 1$ points outside the origin may represent either entirely causal or entirely noncausal delays, and a multiplexer is needed for each point. The complexity of the control unit is similarly increased.

Figure 4 shows the more realistic implementation, where programmable tapped-delay memories replace the delay lines and multiplexers. In practice, several simple RAMs may be used, as were used in Figure 2.

The implementation for an n-point structuring element is a generalization of this one. The 2^n delay lines are still needed, requiring a total of $NM - 1$ memory cells in each RAM. Note that if constraints are placed on $|r_k|$ and $|c_k|$, the sizes of the memories can be accordingly reduced.

Example: Here we perform the dilation of a 4×3 image with a worst-case structuring element, $\{(0, 0), (3, 2), (-3, -2)\}$.

Here is the same data in raster-scan form. Row 1 is the original image data, $R[A](t)$. Row 2 is the primary image, and is the output of the first long shift register, $R[A](r - 11)$. Row 3 is the output of the second long shift register, $R[A](t - 22)$. Row 4 is the mask $R[M](t)$ that is ANDed with row 3 to prevent wraparound. Row 5 is the mask for row 1. Row 6 is the delayed image window, $R[W](t - 11)$. Row 7 is the final image. It is produced by this operation:

5.4 The Pipeline

5.4.1 Discussion. We have now seen possible implementations of two morphological operations, dilation and erosion, using a shift-register approach which can accommodate raster-scan input of image data. Since our implementations both *accept* and *produce* image data in raster-scan format, these implementations are well suited for use as individual stages in a pipeline. This pipeline is well suited for morphological operations that employ chained

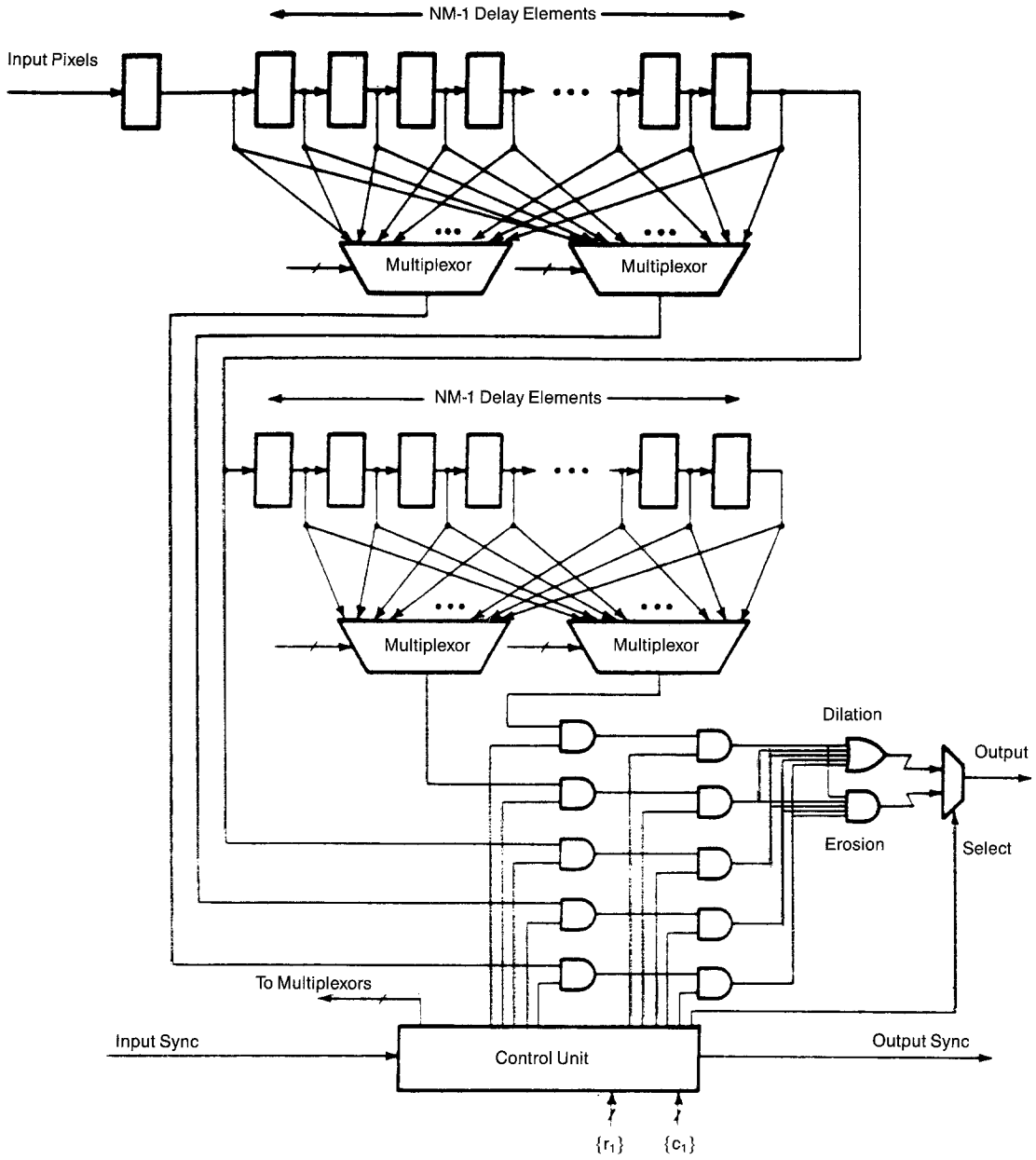


Figure 3. This illustrates a conceptual pipeline architecture for a general three-point structuring element.

structuring elements, since each stage can implement the transformation by one structuring element. An example of this is shown in Figure 5. An analysis follows.

5.4.2 Analysis. We assume the use of general, canonical, n-point structuring elements. We still assume an $N \times M$ image window. Each stage in the pipeline will introduce a delay in the raster-scan image data. This delay is fixed. Suppose that the limitations

$$r_i \leq q \leq N - 1$$
$$c_i \leq s \leq M - 1$$

are imposed on each point in a structuring element. (This still supports a general structuring element if we set $q = N - 1$ and $s = M - 1$.) With these constraints, a worst-case structuring element is

$$\{(0, 0), (q, a), (-q, -s)\}$$

both in terms of amount of delay and the number of storage elements required. Each of the two shift registers in every pipeline stage should then contain $|qM + s|$ delay elements. Therefore the total delay through a single stage is $(|daM + s|) + 1$ time delays because of the single delay element at the start of each stage. With S stages, this implies that the delay, from the first pixel entering the pipeline to the last pixel leaving, is $S[|qM + s| + 1] + NM - 1$ time units. This is if a single image were run through the pipeline.

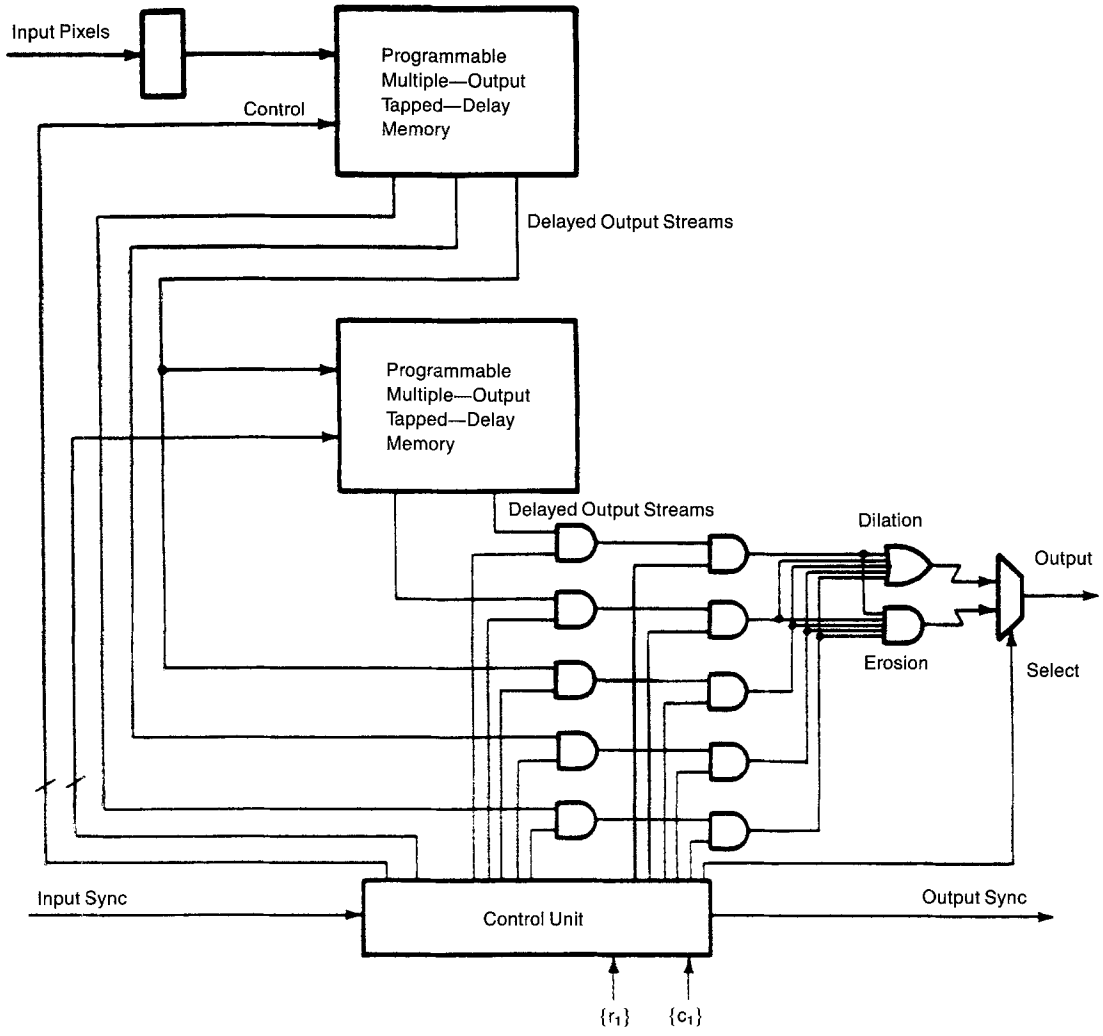
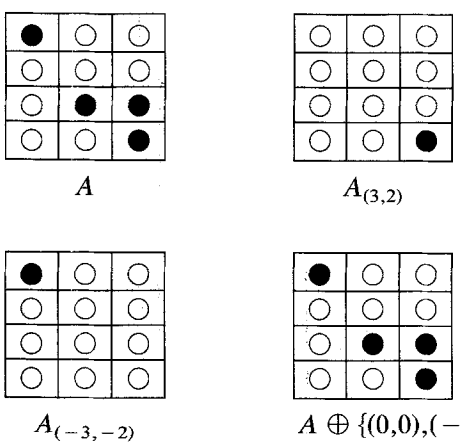


Figure 4. This illustrates a pipeline architecture for a general three-point programmable delay memory.

The real benefit of the pipeline approach is seen if a large number of images are put through the pipeline in sequence or if a very long image ($N \gg M$) is shifted through the pipeline. In that case, an initial delay is seen, but after the pipeline is filled, a new, transformed image appears every NM time units.



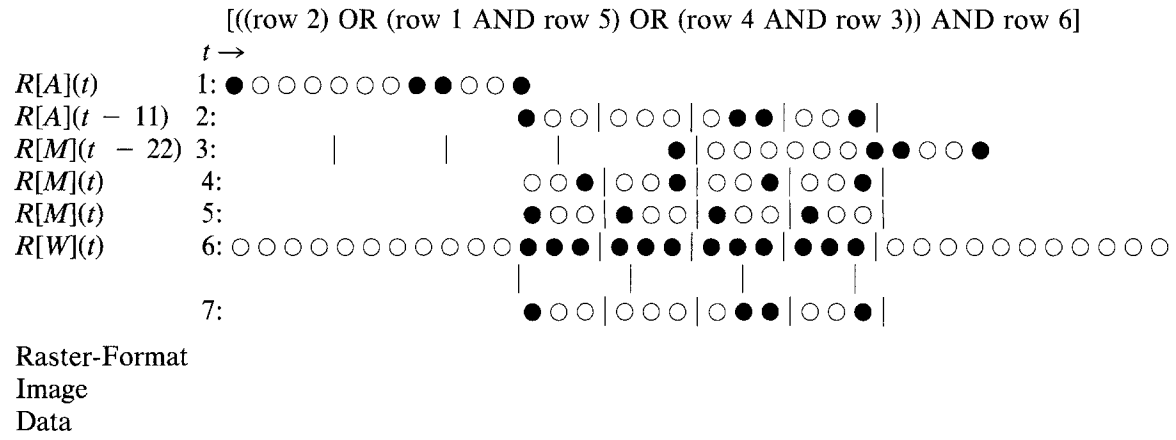
Better performance may be seen if the structuring elements are constrained to contain either causal points only or noncausal points only. With this added constraint, the delay through a single stage can be made variable, and is $|Mr_p + c_p|$, where (r_p, c_p) is the pixel causing the sum $|Mr_k + c_k|$ to be maximum for that stage. Then the worst-case delay through the pipeline is still $S[|qM + s| + 1] + NM - 1$ time units as before, but on the average fewer delays will be required.

6. Pipelined Architectures for Gray-Scale Images

Here again are the definitions of gray-scale dilation and erosion:

$$(X \oplus B)(r,c) = \max_{\substack{(i,j)\in B \\ (r,c)-(i,j)\in X}} \{X(r - i, c - j) + B(i,j)\}$$

$$(X \ominus B)(r,c) = \min_{(i,j)\in B} \{X(r + i, c + j) - B(i,j)\}$$



Instead of the logical operations of union and intersection used in the binary case, the corresponding gray-level operations are the arithmetic max and min over numerical quantities. Unlike the binary case, the operations involve direct interaction between the image and the structuring element, rather than between the image and delayed versions of itself.

We proceed directly to the noncausal case for n-point, arbitrary structuring elements. We again assume that image pixels are presented in raster-scan format. But each pixel now requires b bits of storage, where 2^b is the number of gray levels permitted. Conceptually, a pipeline stage will consist of a chain of b-bit registers, and practically as a series of programmable tapped-delay memories, which may be implemented with simple RAMs, as we have seen. Each memory address must reference b bits.

We again use the function $R{:}E^2 \rightarrow E^1$ which maps an $N \times M$ image to raster format. $R[A]$ represents a one-dimensional array of pixels, each of which is a

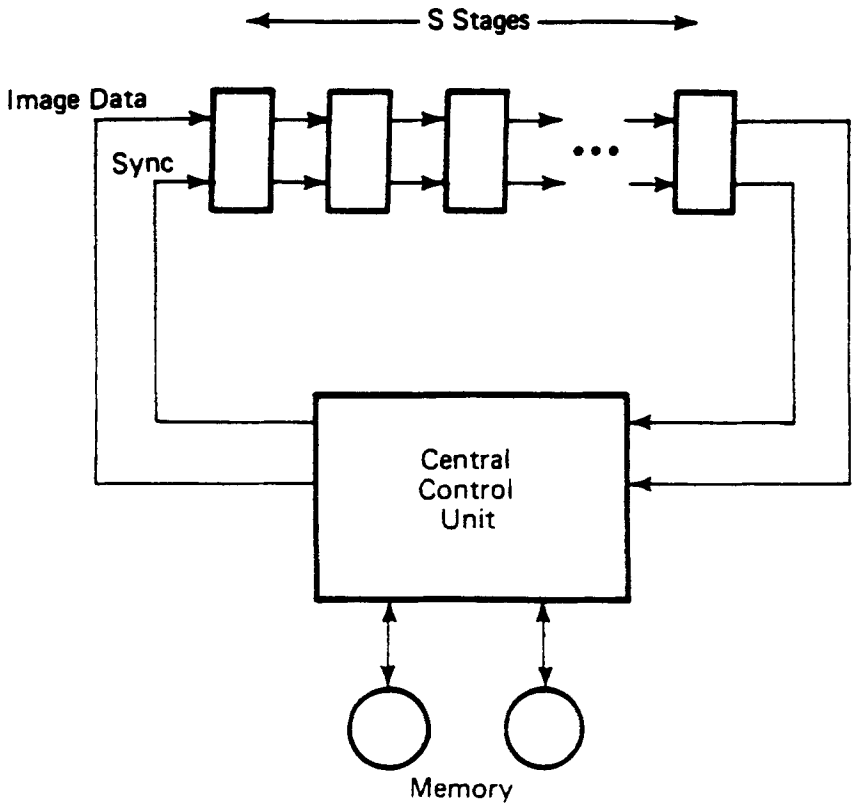


Figure 5. This illustrates a pipeline architecture for performing chained morphological operations.

numerical value. $R[A](t)$ refers to a particular pixel. $R[A](rM + c)$ is the value of the image pixel in the r^{th} row and c^{th} column of A.

As for the binary case, we use a pipeline to provide access to several image pixels simultaneously so that several numerical operations can be performed concurrently.

6.1 Gray-scale dilation

From the definition, we see that the additions of the pixels of the structuring element B to several image points are required. We assume that all of the pixels of B are always available within a pipeline stage. It remains to construct a pipeline that will continuously provide access to all appropriate image pixels.

The $(r, c)^{th}$ output pixel cannot be produced until all pixels $A(r - i, c - j)$ of image A are present in the pipeline, where (i, j) are in the indicated domain. Translated to raster format, this means that all the appropriate pixels $R[A]([r - i]M + c - j)$ must be present. This may be rewritten as $R[A]([rM + c] - [iM + j])$, which demonstrates that the pixels of A that must be available are the pixel $R[A](rM + c)$ along with a neighborhood that depends on the domain of B.

The values for i will range from a minimum value i_{min} (which may be negative), to a maximum value i_{max}, and similarly for j. If $Mi + j < 0$ for any (i, j) in B, noncausal accesses will be required.

Example: The origins for A and $A \oplus B$ are in the upper left corners, and the origin for B is at the center pixel. Since B is defined on a 3×3 domain, a neighborhood of size not greater than 3×3 about a point $A(r, c)$ is involved in the calculation of $(A \oplus B)(r, c)$.

6.2 Gray-scale erosion

From the definition, we see that the *subtractions* of the pixels of B from several image pixels are required. As we saw for dilation, the $(r, c)^{th}$ output pixel cannot be produced until all pixels $A(r + i, c + j)$ of image A are present in the pipeline. Translated to raster format, this mean that all the appropriate pixels $R[A]([r + i]M + c + j)$ must be present. This can be rewritten as $R[A]([rM + c] + [iM + j])$, where it becomes apparent, as for dilation, that the pixel $R[A](rM + c)$ must be present along with a neighborhood that depends on B. The neighborhood which must be present depends on (i, j), but in inverse order as that for dilation. When $iM + j > 0$, noncausal accesses will be required.

Example: The origins here are as in the previous example. Again, a neighborhood of size not greater than 3×3 about $A(r, c)$ is used to produce the result.

6.3 Implementation

A useful pipeline implementation must make available all of the appropriate image pixels. Since these image pixels, in general, are not all adjacent in raster form, a pipeline stage can consist of several programmable tapped-delay memories. Figure 6 illustrates such a circuit, which can perform gray-scale dilation or erosion.

Input pixels are again presented one at a time in raster-scan format. A chain of tapped-delay memories makes the appropriate image pixels available to arithmetic units. These pixels represent a neighborhood about some pixel (r, c) of the image. A quantity inside a rectangle represents the number of unit time delays caused by that element.

Separate storage registers are required for each pixel of the structuring element. Since every element of B may be involved simultaneously in a single maximum or minimum operation, the number of arithmetic units required is the same as the number of pixels in B. Each arithmetic unit is capable of both addition and subtraction.

3	3	5	6	7
3	4	6	7	7
5	6	7	7	8
6	6	7	7	9
6	7	7	8	9

A

0	2	0
1	4	1
0	2	0

B

7	7	9	10	11
7	8	10	11	11
9	10	11	11	12
10	10	11	11	13
10	11	11	12	13

$A \oplus B$

3	3	5	6	7
3	4	6	7	7
5	6	7	7	8
6	6	7	7	9
6	7	7	8	9

A

0	2	0
1	4	1
0	2	0

B

−1	−1	1	2	3
−1	0	2	3	3
1	2	3	3	4
2	2	3	3	5
2	3	3	4	5

$A \oplus B$

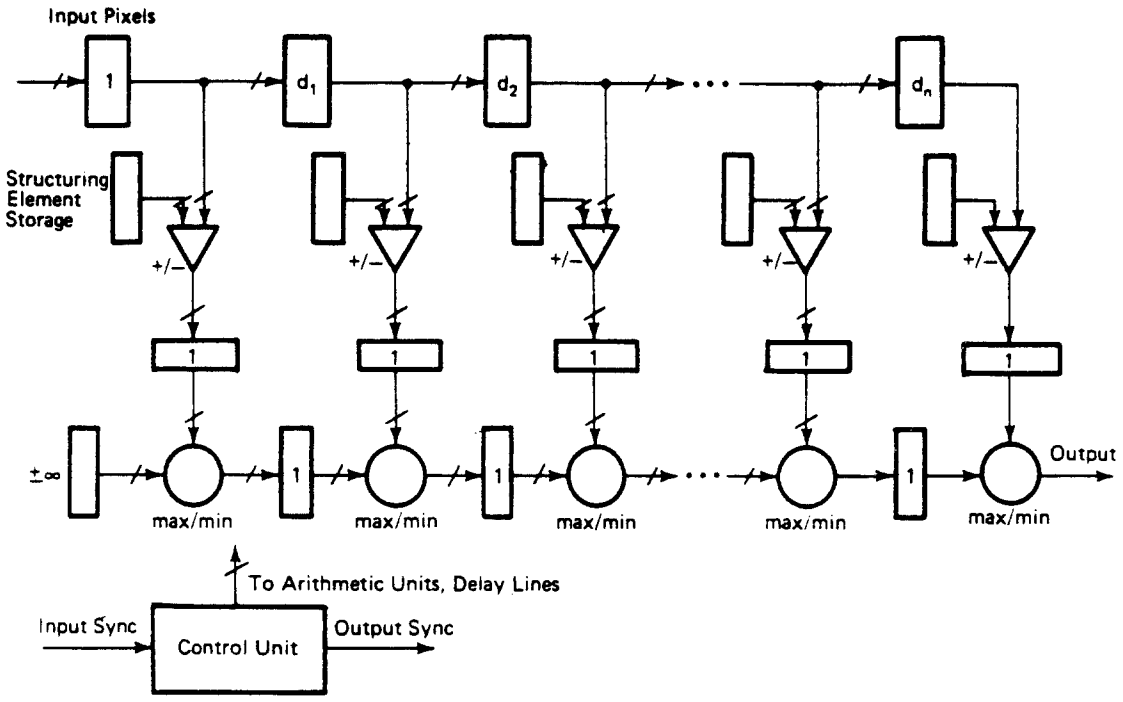


Figure 6. This illustrates the details of the pipeline architecture to perform dilation and erosion.

The wraparound problem that we first saw for binary images is also present for gray-scale images. Suitable masks may be applied, so that the min and max operations are computed only over the indicated domains. A control unit provides the appropriate mask and windowing signals, and specifies the operation of the arithmetic units. The min/max units also receive a select signal.

We now demonstrate how this circuit can perform a morphological operation. Figure 7 shows a specific example. This circuit performs gray-level dilation of an image X with the structuring B, which is made of 4 pixels as shown.

The value passed through the circuit at point P_1 is

$$B(-1,1) + R[X](t + d_1 - 1)$$

At P_2, the value is

$$\max\{B(-1,1) + R[X](t + d_1 - 3),\\ B(0,0) + R[X](t - 2)\}$$

At P_3:

$$\max\{B(-1,1) + R[X](t + d_1 - 4),\\ B(0,0) + R[X](t - 3),\\ B(1,-1) + R[X](t - d_2,-2)\}$$

Finally, the output is

$$\max\{B(-1,1) + R[X](t + d_1 - 4),\\ B(0,0) + R[X](t - 3),\\ B(1,-1) + R[X](t - d_2 - 2),\\ B(1,0) + R[X](t - d_2 - d_3 - 1)\}$$

which can be made to correspond to $(X \oplus B)(t - 3)$.

The appropriate delays d_i are calculated as follows. Letting $u = t - 3$, the output can be rewritten as

$$\max\{R[B](-M + 1) + R[X](u + d_1),\\ R[B](0) + R[X](u),\\ R[B](M - 1) + R[X](u - d_2 + 1),\\ R[B](M) + R[X](u - d_2 - d_3 + 2)\}$$

where the structuring element pixels have also been represented in raster format. Then it remains to formulate the following inequalities:

$$-M + 1 = -d_1 + 1$$
$$M - 1 = d_2 - 1$$
$$M = d_2 + d_3 - 2$$

and solve for the values of d_i.

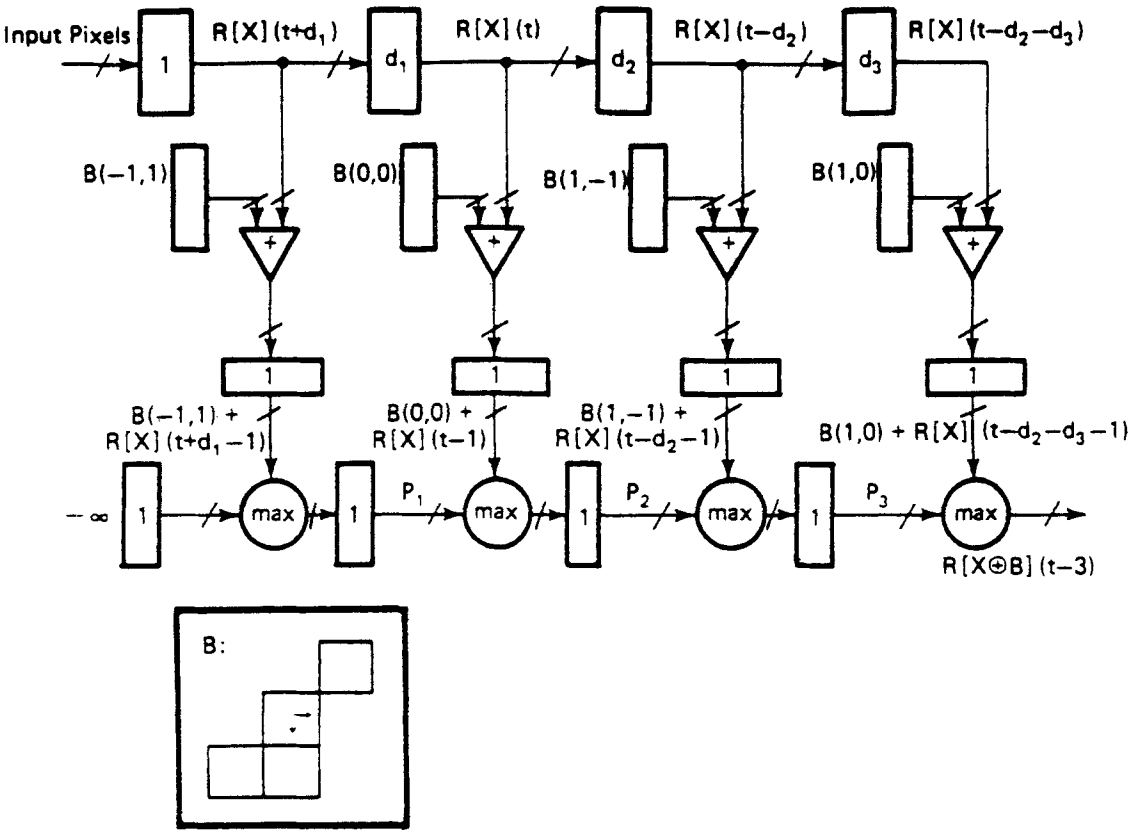


Figure 7. This illustrates the setup for the pipeline architecture to perform a dilation with a specific four-point structuring element.

7. Conclusion

Both binary and gray-scale morphology are becoming increasingly important. We have described a pipeline approach to the general implementation of chained erosions and dilations for both binary and gray-scale images. The pipeline approach to low-level image analysis can overcome some of the I/O limitations inherent in architectures that utilize arrays of separate processing elements. Several architectures were discussed, and some high-level analysis was given.

References

Gerristen FA, Verbeek PW (1984) Implementation of cellular-logic operators using 3 * 3 convolution and table lookup hardware. Computer Vision, Graphics, and Image Processing 27:115–123

Haralick RM, Sternberg SR, Zhuang X (1987) Image analysis using mathematical morphology: Part 1. IEEE Pattern Analysis and Machine Intelligence PAMI-9, (4):532-550

Kimmel MJ, Jaffe RS, Manderville JR, Lavin MA (Nov 1985) MITE: Morphic image transform engine, an architecture for reconfigurable pipelines of neighborhood processes. IEEE Computer Society Workshop for Pattern Analysis and Image Database Management, Miami Beach, Florida:493–500

Klein JC, Serra J (Apr 1973) The texture analyser. Journal of Microscopy 95(2):349–356

Kruse B (1983) State-of-the-art systems for pictorial information processing, Fundamentals in Computer Vision, OD Faugeras (ED.), Cambridge University Press, Cambridge, pp 425–442

Lougheed RM, McCubbrey DL (1980) The cytocomputer: A practical pipelined image processor. Proceedings of the 7th Annual Symposium on Computer Architecture:271–277

Matheron G (1975) Random sets and integral geometry. John Wiley, New York

Preston K, Jr (Jan 1983) Cellular logic computers for pattern recognition. Computer 16:36–47

Reeves AP (Jan 1984) Parallel computer architectures for image processing. Computer Vision, Graphics, and Image Processing 25:68–88

Rice TA, Jamieson LH (1985) Parallel processing for computer vision, Integrated Technology for Parallel Image Processing, S Levialdi (Ed.) Academic, London, pp 57–78

Serra J (1982) Image analysis and mathematical morphology. Academic Press, London

Sternberg SR (1981) Parallel architectures for image processing, Real-Time/Parallel Computing, M Onoe, K

Preston, and A Rosenfeld (Eds.), Plenum Press, New York, pp 347–359

Sternberg SR (1985) An overview of image algebra and related architectures. In Integrated technology for parallel image processing. Academic Press, London, pp 79–100

Sternberg SR (1983) Biomedical image processing. Computer, 16:23–34

Yalamanchili S, Palem KV, Davis LS, Welch AJ, Aggarwal JK (1985) Image processing architectures: A taxonomy and survey, Progress in Pattern Recognition 2 LN Kanal and A Rosenfeld (Eds.), Elsevier (North Holland), Amsterdam, pp 1–37

Zhuang X, Haralick RM (1986) Morphological structuring element decomposition. Computer Vision, Graphics, and Image Processing 35:370–382