Approximating Large Convolutions in Digital Images

David M. Mount, Tapas Kanungo, *Senior Member, IEEE*, Nathan S. Netanyahu, *Member, IEEE*, Christine Piatko, Ruth Silverman, and Angela Y. Wu, *Senior Member, IEEE*

***Abstract*—Computing discrete two-dimensional (2-D) convolutions is an important problem in image processing. In mathematical morphology, an important variant is that of computing binary convolutions, where the kernel of the convolution is a 0–1 valued function. This operation can be quite costly, especially when large kernels are involved. In this paper, we present an algorithm for computing convolutions of this form, where the kernel of the binary convolution is derived from a convex polygon. Because the kernel is a geometric object, we allow the algorithm some flexibility in how it elects to digitize the convex kernel at each placement, as long as the digitization satisfies certain reasonable requirements. We say that such a convolution is *valid*. Given this flexibility we show that it is possible to compute binary convolutions more efficiently than would normally be possible for large kernels. Our main result is an algorithm which, given an $m \times n$ image and a k-sided convex polygonal kernel K, computes a valid convolution in $O(kmn)$ time. Unlike standard algorithms for computing correlations and convolutions, the running time is independent of the area or perimeter of K, and our techniques do not rely on computing fast Fourier transforms. Our algorithm is based on a novel use of Bresenham's line-drawing algorithm and prefix-sums to update the convolution incrementally as the kernel is moved from one position to another across the image.**

***Index Terms*—Approximation algorithms, correlations, digital convolutions, digital geometry, mathematical morphology.**

I. Introduction

A FUNDAMENTAL problem in image processing is that of computing *discrete convolutions* [5], [9], [12]. Consider an image, which is given as a two-dimensional (2-D) $m \times n$ array I of real numeric values, and a $q \times r$ image array K, called the *kernel* (sometimes called a *template* or *structuring element* in the literature). The *discrete convolution* [9] of I with K, denoted by $I * K$, is defined to be

$$(I * K)[x, y] = \sum_a \sum_b I[a, b] \cdot K[x - a, y - b].$$

Manuscript received June 17, 1999; revised August 29, 2001. This work was supported by the National Science Foundation under Grant CCR-9712379. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Uday B. Desai.

D. M. Mount is with the Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20740 USA (e-mail: mount@cs.umd.edu).

T. Kanungo is with IBM Almaden Research Center, San Jose, CA 95120 USA (e-mail: kanungo@almaden.ibm.com).

N. S. Netanyahu is with the Department of Mathematics and Computer Science, Bar-Ilan University, Ramat-Gan, Israel and also with the Center for Automation Research, University of Maryland, College Park, MD 20740 USA (e-mail: nathan@cs.biu.ac.il).

C. Piatko is with The Johns Hopkins University Applied Physics Laboratory, Laurel, MD 20707 USA (e-mail: christine.piatko@jhuapl.edu).

R. Silverman is with the Center for Automation Research, University of Maryland, College Park, MD 20740 USA (e-mail: ruth@cfar.umd.edu).

A. Y. Wu is with the Department of Computer Science and Information Systems, The American University, Washington, DC 20016 USA (e-mail: awu@american.edu).

Publisher Item Identifier S 1057-7149(01)10562-2.

It is common to embed I and K within larger images to avoid wraparound effects. We will assume that any such transformations have already been applied and ignore wraparound.

A *binary convolution* is a special case of a discrete convolution where K is a 0–1 valued function. Binary convolutions are of particular interest in computational morphology and digital geometry [21], [22]. For example, the *dilation* of a digital shape, described by a 0–1 image I, by a digital kernel K described by another such image can be expressed by computing the convolution $I*(-K)$, and then thresholding this image so that all strictly positive values are mapped to 1. Here, $-K = \{-p | p \in K\}$ denotes the reflection of K with respect to the origin. See also [1] for an asymptotically efficient algorithm for computing dilations of digital sets. Binary convolutions are also useful in template matching [9] in binary images, through the use of the related correlation operation. Our results apply to computing binary correlations as well. Binary convolutions have the following geometrical interpretation. We can interpret $I * K(p)$ as placing a copy of $-K$ at location p of the image, and then computing the number of pixels of I that are overlapped by $-K(p)$.

One problem with computing discrete convolutions is that the operation can be quite expensive when the kernel of the convolution is large. A naive algorithm for computing the convolution considers each placement of the $q \times r$ kernel, and computes the weighted sum in $O(qr)$ time. Since there are mn possible placements, this results in an algorithm whose running time is $O(mnqr)$. Here we assume that $q \leq m$ and $r \leq n$, but these quantities may still be large. The question is whether we can improve on the qr factor, especially when qr is large.

A number of approaches for improving the efficiency of convolution computation have been proposed. Kim and Kim proposed a simple method based on the observation that in many commonly used kernels the number of distinct nonzero elements is small [15]. Perhaps the most common approach for improving the efficiency of convolution computation is based on decomposing a convolution involving a large kernel into a sequence of convolutions involving small kernels [18]. In the case of binary morphology, search algorithms have been proposed to decompose the kernels as dilations of two-point structuring elements [26]. When the kernel shape is restricted to shapes formed by intersections of half-planes at multiples of 45°, it has been shown that the kernel can be decomposed as dilations of kernels from a basis set [13]. This sort of decomposition been extended to 3×3 structuring elements for both digitally convex [19] and nonconvex cases [20]. The basic two-point search algorithm was also extended to grayscale operations [4]. In all cases, it is assumed that the kernel is described by an eight-way directional chain code. Hence, many natural shapes such as triangles with arbitrarily sloped sides are not decomposable in this manner.

There has also been extensive work on related decomposition methods for grayscale kernels [8], [24].

The concepts of singular value decomposition (SVD) and small generating kernel (SGK) have been used to speed up the grayscale convolution processing time by decomposing the kernel into separable filters and then decomposing each separable filter as a sequence of SGK filters [16]. The speedup is obtained by using the kernels corresponding to the larger eigenvalues. The SVD/SGK methods, however, are useful only in the case of grayscale convolutions.

Other methods involve using table lookup to avoid the cost of multiplication [6], [25]. Burt proposed a technique based on the use of quadtrees [3]. However, these methods improve running times only by a constant factor. A common approach to computing convolutions for large kernels is first to compute the Fourier transforms of the image and kernel, denoted by I' and K'. Then the convolution $I * K$ can be approximated in $O(mn)$ time by computing the elementwise product $I' \cdot K'$, and then inverting the transform [9]. This approach requires only $O(mn \log(mn))$ time, which is a significant savings. However, for morphological and other discrete applications, it has the inelegant property of converting an exact discrete problem into a continuous problem.

In this paper we consider a significantly different approach. We consider the problem of computing binary convolutions where the kernel of the convolution is derived from a convex polygon. We introduce the notion of a *valid digitization* of a geometric shape. We present formal definitions later, but intuitively, a digitization is *valid* if pixels lying entirely inside the shape are in the digitization and pixels lying entirely outside the shape are not in the digitization. We then define the notion of a *valid convolution*, which is based on using valid digitizations of the kernel to perform the convolution. Different placements of the kernel are allowed to use different digitizations. We show that with this added flexibility it is possible to compute digitizations for convex polygonal kernels in time that is independent of the area or perimeter of the kernel. In particular, we show that a valid convolution of an $m \times n$ image with a k-sided convex polygonal kernel can be computed in $O(kmn)$ time and $O(mn)$ space. This type of convolution is of interest in morphology applications, where the kernel can be approximated by a convex polygon, or decomposed into a small number of convex polygons. If k is small, this can be significantly faster than existing approaches for large kernels. The most closely related work to ours is that of box-filtering [17]. However, box-filtering is limited to rectangular shapes. Our approach is a generalization of the box-filtering method to nonrectangular convex polygons. The notion using continuous mathematics in interpreting discrete morphological operations has been considered elsewhere [11], [23]. A preliminary version of this paper appeared in [14].

II. Definitions and Notation

We begin with some definitions. Let $\mathbb{Z}^2$ denote the set of ordered pairs of integers, called *grid points* and let $\mathbb{R}^2$ denote the set of ordered pairs of reals. Given $g \in \mathbb{Z}^2$, define $\pi(g)$ to be a semi-open unit square centered at g, called g's *pixel*

$$\pi(g) = \{(x, y) | g_x \le x < g_x + 1,\ g_y \le y < g_y + 1\}.$$

The collection of $\pi(g)$ for all $g \in \mathbb{Z}^2$ subdivides the real plane into a collection of pairwise disjoint semi-open unit squares. Let us think of the $m \times n$ image as defining a function $I\colon \mathbb{Z}^2 \to \mathbb{R}$, where

$$I(g) = \begin{cases} I[m+1-g_y,\ g_x], & \text{if } 1 \le g_x \le n \text{ and } 1 \le g_y \le m, \\ 0, & \text{otherwise.} \end{cases}$$

This mapping reflects the convention that image arrays are typically indexed by row and column from the upper left corner. Henceforth, our indexing will be done assuming the standard (x, y)-coordinate system.

Given a set $P \subset \mathbb{R}^2$ and $t \in \mathbb{R}^2$, let $t + P$ denote the translate of P by t, that is

$$t + P = \{t + p | p \in P\}.$$

We will call this translate the *placement* of P at t. Let $-P = \{-p | p \in P\}$, and define $t - P$ to be $t + (-P)$.

Given a set $P \subset \mathbb{R}^2$, a *digitization* is a mapping $D(P)$ of P to a set of grid points. A digitization $D(P) \subseteq \mathbb{Z}^2$ is *valid* if for every pixel that lies entirely within P the corresponding grid point is in the digitization, and for every pixel that is entirely outside of P the corresponding grid point is not in the digitization. More formally,

$$\pi(g) \subseteq P \Rightarrow g \in D(P)$$

and

$$(\pi(g) \cap P = \emptyset) \Rightarrow g \notin D(P).$$

Pixels that partially overlap P may or may not be in a valid digitization. An example of a valid digitization is the *midpoint digitization*, denoted by $D^m(P)$, which consists of all grid points that lie within P. (Fig. 1 shows valid digitizations of three different placements of the same polygon P. The one in the center is the midpoint digitization.)

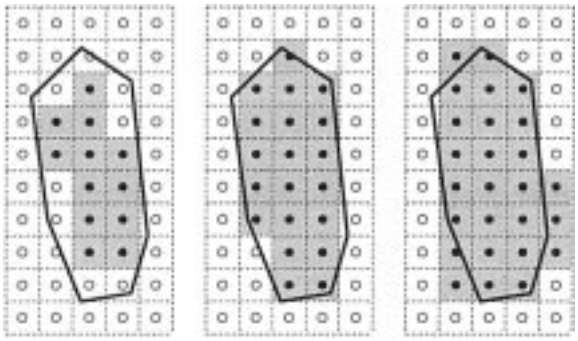


Fig. 1. Valid digitizations.

Given an image I and any set of grid points $G \subset \mathbb{Z}^2$, define the *weight* of G relative to I to be the sum of the image values of G

$$w(G) = \sum_{g \in G} I(g).$$

Let K be a convex polygon in the plane, and let I be an image. We define the *midpoint convolution* of I by K to be an $m \times n$

image C, where $C(t)$ is defined to be the weight of the midpoint digitization of $-K$ placed at t, that is

$$C(t) = w(D^m(t-K)) = \sum_{g \in D^m(t-K)} I(g).$$

A *valid convolution* of I by K is defined in the same way, with $D^m(t-K)$ replaced by any valid digitization of $t-K$. Note that because valid digitizations are not unique, different placements may be digitized differently. A valid convolution is a variation of the standard binary convolution, subject to some flexibility in the shape of the kernel. Our main result is the following.

Theorem 1: Given an $m \times n$ image I and a k-sided convex polyon K, a valid convolution of I by K can be computed in $O(kmn)$ time and $O(mn)$ space.

Henceforth, to avoid the continual need for negations, we will assume that the kernel for the convolution is $-K$, and so the value of the convolution at each point is just the weight of some valid digitization of a translate of K. The union of the pixels of the image forms the *image rectangle* R, where

$$R = \{(x, y) | 0.5 \le x < n + 0.5,\ 0.5 \le y < m + 0.5\}.$$

All grid points outside this rectangle are assumed to have value 0. We assume that the kernel K is represented by a cyclic listing of its vertex coordinates.

III. The Algorithm

In this section we describe our convolution algorithm. We begin with an intuitive high-level description of the essential technique used by the algorithm. Recall that k denotes the number of sides of the kernel K. The first stage of the algorithm involves decomposing the kernel K into a collection of $O(k)$ simpler shapes, called *primitive shapes*, such that K can be represented as a weighted sum of these shapes. Each primitive shape is an axis-aligned rectangle or right triangle.

The second stage involves preprocessing the image. We create $k+2$ sequences of equally spaced parallel lines, where each sequence is either horizontal, vertical, or parallel to a side of K. These are called *canonical lines*. Each sequence of canonical lines decomposes the image rectangle into a collection of thin regions called *canonical strips*. We will digitize each strip using midpoint digitization and preprocess it by a method to be described later. We will show that in constant time, it is possible to compute the total weight of a parallelogram defined by a strip and two lines that are either horizontal or vertical. We will show that this structure can be built in time and space $O(mn)$ for each side of K.

The third stage of the algorithm computes the actual convolution. It is based on computing valid convolutions for each of the primitive shapes and then summing the results over all $O(k)$ shapes to get the final convolution. For any placement of a primitive shape, we will define a special valid digitization called the *canonical digitization*. We will show that for each primitive shape, the weight of a single placement of the shape can be computed in $O(mn)$ time. Then we will show that once the weight of one placement is known, it is possible to update the weight in $O(1)$ time whenever the placement is translated by a unit distance, either horizontally or vertically. This is done with the aid of the digitizations of the canonical strips. By translating the primitive shape to each point of the image, the entire convolution for each primitive shape can be computed in $O(mn)$. Finally, we sum the convolutions of all k primitive shapes, producing the convolution by K in $O(kmn)$ total time. The various elements of the algorithm are explained in detail in the following subsections.

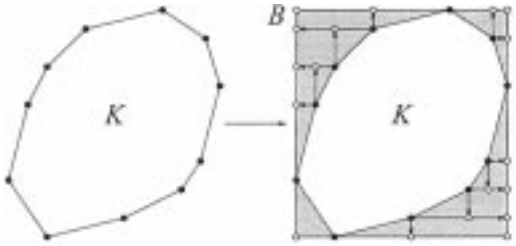


Fig. 2. Decomposing the kernel into primitive shapes.

A. Decomposition Into Primitive Shapes

As mentioned above, the first stage of the algorithm involves representing K as a weighted sum of $O(k)$ *primitive shapes* and the bounding box B. The representation is constructed by first enclosing K in an axis-aligned bounding box B, and then decomposing the difference $B \backslash K$ into a collection of rectangles and right triangles. We visit the vertices of K in cyclic order and for each vertex we imagine shooting two bullets horizontally and vertically away from the interior of K, until hitting either the bounding box B or a previous bullet's path (see Fig. 2). It is easy to see that these bullet paths subdivide $B \backslash K$ into a set of rectangles and right triangles with pairwise disjoint interiors. Together with B, these form the set of primitive shapes.

Let k denote the number of sides of K and let r denote the number of such shapes. Observe that each time a bullet is shot, it splits some region into at most two subregions. Since $2k$ bullets are shot (two per vertex of K), and we started with the bounding box B, it follows that $r \le 2k + 1 = O(k)$. Each bullet shoot can be done in $O(1)$ time, since the result depends only on the location of the bounding box and possibly the result of the bullet paths of the previous vertex. The *primitive shapes* are denoted $K_1, K_2, \ldots, K_r$.

Shapes $K_1, K_2, \ldots, K_r$ have pairwise disjoint interiors. Let $\mathrm{bnd}(K_i)$ denote the boundary of K_i. If a grid point in B falls on the boundary between two or more of these shapes, we assign it uniquely to one of them as follows. Consider the vector $v_\epsilon = (\epsilon, \epsilon^2)$ for an infinitesimal $\epsilon > 0$. A point p on $\mathrm{bnd}(K_i)$ is assigned to K_i if and only if $p + v_\epsilon$ is in the interior of K_i.[1] Intuitively, this means that each shape is semi-open with the lower-left parts of the boundary belonging to K_i (see Fig. 3). Note that this is consistent with our convention that pixels are closed on their lower-left sides. Let us apply this convention to the convex kernel K as well. Because the definition of a valid digitization provides the freedom to include a grid point on the boundary of K or not, there is no loss of generality in applying this convention to K.

We assert next that K can be expressed as a weighted sum of these shapes and B. The bounding box B is assigned a weight

[1]The reason for squaring the second component of the vector is so that the vector's angle with respect to the x-axis decreases with ϵ. Because the polygon is bounded by straight line edges, for all sufficiently small $\epsilon > 0$, the point $p + v_\epsilon$ cannot lie on the boundary of K_i.

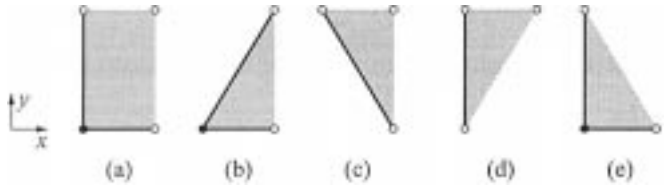


Fig. 3. Examples of semi-open shapes.

of $+1$, and all the other primitive shapes $K_1, K_2, \ldots, K_r$ are assigned a weight of -1. Let $K_0 = B$. Let w_i denote the weight of K_i. Let $K_i(p)$ be 1 if $p \in K_i$ and 0 otherwise. For all $p \in \mathbb{R}^2$, define the weight of p to be $W(p) = \sum_{i=0}^{r} w_i K_i(p)$.

Lemma 1: For all $p \in \mathbb{R}^2$,

$$W(p) = \begin{cases} 1, & \text{if } p \in K \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Proof: Points outside B are clearly not in K and have weight 0. Points in the interior of K have weight 1 since they lie inside B but outside all the other primitive shapes. Because the primitive shapes other than B are pairwise disjoint and cover $B \backslash K$, each point of $B \backslash K$ lies in B and exactly one shape K_i and hence has weight $1 - 1 = 0$. □

B. Canonical Lines and Canonical Digitizations

For each $t \in \mathbb{Z}^2$, and each primitive shape of K_i, we define a special digitization of the placement $t+K_i$, called the *canonical digitization* and denoted by $D^c(t + K_i)$. This will be done in such a way that the weighted sum of these digitizations defines a valid digitization of the placement $t + K$.

Consider any primitive shape K_i. If K_i is a rectangle then define $D^c(t + K_i)$ to be the midpoint digitization, that is, the set of grid points lying in the intersection of $t+K_i$ and the image rectangle R. If K_i is a right triangle, then the main issue is how to digitize its slanted side (the one that is not axis parallel). To do this we introduce the notion of a *canonical line*. We consider two cases depending on the slope of K_i's slanted side. If the absolute value of the slope of the slanted side of the triangle is less than 1, we call K_i a *low-slope triangle*; otherwise, we call it a *high-slope triangle*. Below we consider the high-slope case. The low-slope case is handled in a symmetrical manner, by swapping the roles of the x- and y-axes.

Let s denote the slope of the slanted side of K_i. Consider the sequence of lines (sorted, from left to right) that intersect the image rectangle R, have slope s, and have x-intercept an integer multiple of 0.5 (see Fig. 4). Observe that the horizontal distance between two consecutive canonical lines is 0.5. (In the low-slope case, y-intercepts are used instead, and the vertical spacing is 0.5.) These lines subdivide R into a collection of thin regions, called *canonical strips*. (The reason for the choice of 0.5 as the separation distance will be discussed later.)

Lemma 2: For any primitive shape K_i and an $m \times n$ image rectangle R, the number of canonical lines and number of canonical strips generated by K_i is $O(m + n)$.

Proof: Assume for concreteness that s is positive and $s \geq 1$. The proofs for the other cases follow from simple symmetry. Recall the definition of the image rectangle R from Section II. By considering the lines passing through the upper left corner $(0.5, m+0.5)$ and the lower right corner $(n+0.5, 0.5)$ of R, it follows that a line of slope s intersects R only if its x-intercept lies within the interval

$$X = \left[0.5 - \frac{m+0.5}{s},\ n + 0.5\left(1 - \frac{1}{s}\right)\right]$$

[see Fig. 5(a)]. Since $s \geq 1$ it follows that the length of this interval is at most $n + m$. □

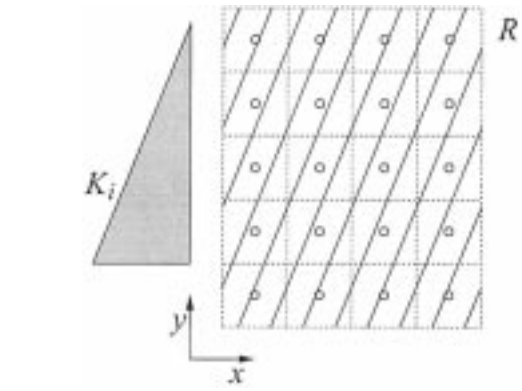


Fig. 4. Canonical lines.

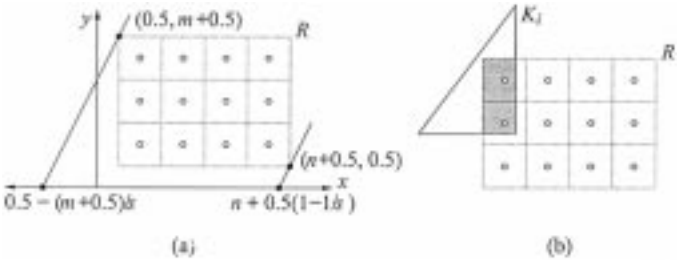


Fig. 5. (a) Range of canonical lines and (b) the intersection of a triangle K_i with R.

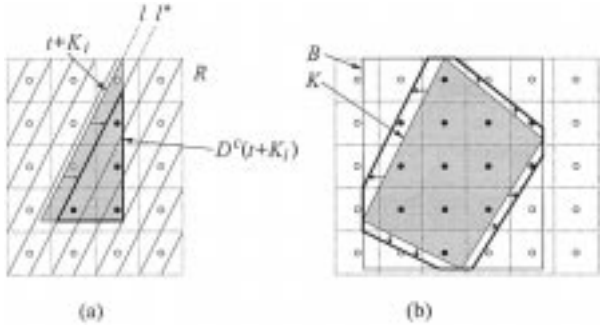


Fig. 6. Canonical digitization.

The *canonical digitization* of a right triangle $t + K_i$ is defined as follows. Consider the line ℓ supporting the slanted side of K_i. If ℓ does not intersect the image rectangle R, then the intersection of $t+K_i$ is either empty or a rectangle [see Fig. 5(b)]. In the latter case, the canonical digitization is defined as in the rectangle case to be the set of grid points lying within this rectangle. Otherwise, let us assume for the sake of concreteness that the triangle lies to the right of the slanted line. (The other case is symmetrical.) Select the nearest canonical line ℓ^* that lies on or to the right of ℓ [see Fig. 6(a)]. This can be accomplished in constant time by computing the x-intercept of ℓ, and then rounding to the next larger integer multiple of 0.5. In general, ℓ is rounded toward the interior of the triangle it supports.

The canonical digitization of $t+K_i$ is defined to be the set of grid points that lie within the intersection of the bounding rectangle of $t+K_i$ and the image rectangle R, and are either on or to the right of ℓ^*. [These are shown as black points in Fig. 6(a).] Notice that by rounding to the canonical line lying toward the interior of the triangle, the canonical digitization consists of a subset of the grid points lying in $t+K_i$. Thus it is a subset of the midpoint digitization of $t+K_i$. [For example, in Fig. 6(a) two grid points in the midpoint digitization have been excluded from the canonical digitization.]

Given the canonical digitization for a single primitive shape, we define the canonical digitization for K as the weighted sum of the canonical digitizations for primitive shapes $t+K_i$. More formally, recalling the weights w_i introduced above, we define the canonical digitization of $t+K$ to be the weighted sum of the digitizations of $t+K_i$, that is

$$D^c(t+K) = \sum_{i=0}^{r} w_i \cdot D^c(t+K_i).$$

In other words, a pixel lies in the canonical digitization if the weighted sum of sets containing this pixel is 1, and does not lie in it if this weighted sum is 0. [An example is shown in Fig. 6(b). The grid points belonging to the final digitization are shown as black points in this figure.] Since the primitive shapes lie outside K, observe that the sides are rounded away from the interior of K, and hence the points of the canonical digitization of K are a superset of the points in the midpoint digitization of K. Next we show that the result is a valid digitization of K.

Lemma 3: For any convex polygon K, and any vector $t \in \mathbb{Z}^2$, the canonical digitization $D^c(t+K)$ is a valid digitization of $t+K$.

Proof: By the definition of a valid digitization, it suffices to show the following for each grid point $g \in \mathbb{Z}^2$. Recall that $\pi(g)$ is the pixel (semi-open unit square) centered at g.

1) If $\pi(g)$ lies entirely within $(t+K)\cap R$ then g is assigned a weight of 1.
2) If $\pi(g)$ is entirely outside of $(t+K)\cap R$ then g is assigned a weight of 0.
3) Otherwise g is assigned a weight of either 0 or 1.

Recall that a grid point g is in the midpoint digitization of a shape if and only if g lies in that shape. To establish 1), observe that if $\pi(g)$ lies entirely within $(t+K)\cap R$ then its midpoint g does as well. Every grid point in $(t+K)\cap R$ is given an initial weight of 1 because it lies within the bounding rectangle $t+B$. Furthermore, because each of the canonical digitizations is a subset of the midpoint digitization, no canonical digitization for any primitive shape can contain this point. Thus, its total weight is 1.

To establish 2), consider a pixel $\pi(g)$ that is disjoint from $(t+K)\cap R$. If g lies entirely outside the bounding box $t+B$, or outside the image bounding box R, it is not allocated to any canonical digitization, and so it is given a weight of 0. Otherwise, g lies within $(t+B)\cap R$ and outside $(t+K)\cap R$, and hence lies in a unique primitive shape $t+K_i$. We assert that g is in the canonical digitization of some shape $t+K_i$. Observe that if this is true, then because the primitive shapes are disjoint, g does not lie in the canonical digitization of any other primitive shape. But g does lie within B, and hence it is assigned a weight of $1-1=0$.

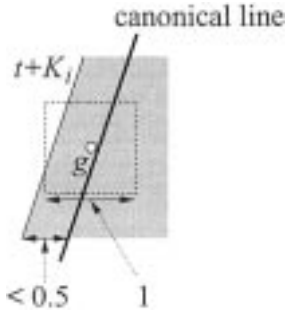


Fig. 7. Proof of Lemma 3.

To establish this assertion, suppose that g lies within the bounding box $t+B$, but outside $t+K$, and g is not in the canonical digitization of any primitive shape. We derive a contradiction. Clearly g is in a unique primitive shape $t+K_i$, but not in the canonical digitization of $t+K_i$. For the sake of concreteness, let us consider the case of a high-slope side where K_i lies to the right of its slanted side (see Fig. 7). Since g is not in the canonical digitization, it lies to the left of the associated canonical line. However, by construction, the canonical line is to the right of the slanted side of $t+K_i$ by a horizontal displacement of at most 0.5. Therefore the slanted side of $t+K_i$ lies to the left of g by a horizontal displacement of less than 0.5. Since the pixel $\pi(g)$ is of width 1 and g is its midpoint, it follows that the slanted side of $t+K_i$ intersects $\pi(g)$. However, this contradicts hypothesis 2). The low-slope case is proved symmetrically, using vertical distances.

Finally, to show 3), consider a pixel $\pi(g)$ that intersects the boundary of $(t+K)\cap R$. If g lies outside the bounding box $(t+B)\cap R$, it is assigned a weight of 0. Otherwise, g will be assigned an initial weight of 1 because it lies inside the bounding box. We claim that g can be in the canonical digitization of at most one other primitive shape. This is because g can lie in at most one primitive shape, and hence it lies in the midpoint digitization of at most one primitive shape. Because the canonical digitization of a shape is a subset of the midpoint digitization, g lies in at most one canonical digitization. If g is in some such canonical digitization, its final weight is 0, and otherwise its weight is 1. This establishes 3), and completes the proof. □

The reason for rounding lines toward the interior of the primitive shape triangles and the choice of 0.5 as the separation distance between canonical lines is apparent from the proof. The proof of part 2) relied on the fact that the horizontal distance is half the width of a pixel. The proof works as long as the distance between canonical lines is at most 0.5. By reducing the spacing between the canonical lines it is possible to produce a digitization that is arbitrarily close to the midpoint digitization. However this would result in proportionally more canonical strips, and hence the algorithm's running time and storage costs would increase proportionally. Note that the proof of 3) relied on the fact that canonical digitizations are subsets of the associated midpoint digitizations. If this were not the case, a pixel whose center is in K but which is intersected by two sides

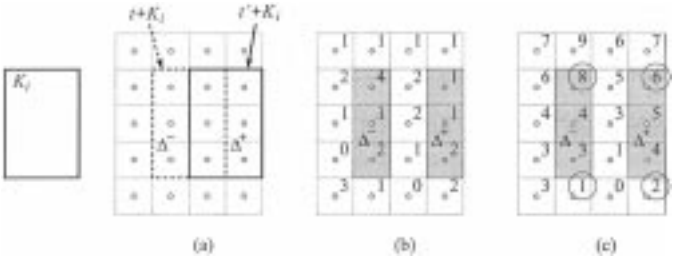


Fig. 8. (a) The two translates $t + K_i$ (dotted) and $t' + K_i$ (solid) and the rectangles Δ^- and Δ^+ which form the symmetric difference, (b) the initial image pixel weights, and (c) the weights after computing prefix sums within each column.

of K might conceivably be assigned to the canonical digitizations of two difference primitive shapes. The resulting weight of the associated grid point would be $1-1-1=-1$, and this does not correspond to any valid digitization of K.

C. Updating Canonical Digitizations

The main algorithmic tasks needed to compute the digitization are 1) computing the weight of the canonical digitization of a single placement of a primitive shape, and 2) updating the weight of the canonical digitization when the shape is shifted by one unit distance, either horizontally or vertically. The first task can be accomplished by applying any standard algorithm for digitizing convex polygons [7]. The second task will be addressed in the remainder of this section.

1) Rectangular Shapes: We first consider the case of a rectangular primitive shape K_i, since it is the simplest. For concreteness we consider the case of a translation to the right by one unit. (The other unit shifts are handled similarly.) Let t and t' be two placement vectors where that $t' = t + (1, 0)$. We assume that the weight of the canonical digitization of $t + K_i$ is known, and we want to compute the weight of the canonical digitization of $t' + K_i$. First, observe that the symmetric difference between $t+K_i$ and $t'+K_i$ is the union of two congruent rectangles Δ^- and Δ^+, each of unit width, where Δ^- lies to the left of $t'+K_i$ and Δ^+ lies to the right of $t+K_i$ [see Fig. 8(a)]. The weight of $t' + K_i$ is equal to the weight $t + K_i$ minus the weight of Δ^- and plus the weight of Δ^+. Since we are dealing with canonical digitizations, this means

$$w(D^c(t' + K_i)) = w(D^c(t + K_i)) - w(D^c(\Delta^-)) + w(D^c(\Delta^+)).$$

Observe that if the width of K_i is less than 1 then Δ^- and Δ^+ overlap. However, this is not a problem because the weights in the region of overlap will cancel when we add one and subtract the other.

The incremental change in weight can be computed in constant time once we know the weights of the two unit-width rectangles Δ^- and Δ^+. To do this we preprocess the image as follows. For each column and each grid point we store the total weight of the image points in that same column that have equal or smaller y-values. This is called a *prefix sum*.

For example, in Fig. 8(b) we show the weights of the pixels of the image. Observe that the total weight of Δ^- is $2+1+4=7$. In Fig. 8(c) we show the result after computing the prefix sums for each column. The weight of Δ^- is the difference between the prefix sums of the topmost grid point in the rectangle and the grid point just below the bottommost grid point, that is, $8-1=7$. Doing the same for Δ^+ we find that its weight is $6-2=4$, and hence the total change of weight between placements t and t' is $4-7=-3$. The weight of $t+K_i$ is 12 and hence after only three arithmetic operations (after preprocessing) we determine that the weight of $t' + K_i$ is $12-3=9$.

The prefix sums for each column can be computed by a simple scan in $O(m)$ time, implying that all the prefix sums can be computed in $O(mn)$ total time. The weight of the canonical digitization of any rectangle of unit width can be computed in constant time by rounding its x-coordinates to determine the grid column that it spans, and then rounding its y-coordinates to determine the elements of the prefix sum whose difference is to be taken. A vertical unit-length translation is handled similarly, but it results in two rectangles of unit vertical height. The preprocessing for this case consists of computing prefix sums for each of the rows.

2) Triangular Shapes: Horizontal Translation: Next we consider how to update the weight of the canonical digitization of the placement of a right-triangle primitive shape K_i. We will assume that the slanted side of K_i has a slope that is positive and at least 1. The cases for negative and/or low slopes are handled similarly. Let us first consider the case of a horizontal translation; we will consider vertical translations later. As before, let $t + K_i$ denote the current placement of K_i, whose weight we know, and let $t' + K_i$ be the new placement, whose weight we wish to compute. Let $t' = t + (1, 0)$ [see Fig. 9(a)].

In the rectangular case, we reduced the problem to that of computing the difference of weights of two rectangles of unit width. In this case we will see that the problem reduces to computing the difference of weights of a rectangle and a parallelogram. Consider two shapes Δ^+ and Δ^- [see Fig. 9(b)]. Δ^+ is a rectangle of unit width that lies to the right of $t+K_i$, and Δ^- is a parallelogram of unit width lying to the left of $t'+K_i$, with horizontal top and bottom sides and slanted sides that are parallel to the slanted side of K_i. These two shapes overlap one another, but observe that the symmetric difference of $t+K_i$ and $t'+K_i$ is equal to the symmetric difference of Δ^+ and Δ^-. Thus, we have

$$w(D^c(t' + K_i)) = w(D^c(t + K_i)) - w(D^c(\Delta^-)) + w(D^c(\Delta^+)).$$

The horizontal width of Δ^+ is one unit, and hence it spans exactly one column of grid points. Its weight can be computed in constant time using the same method described earlier for

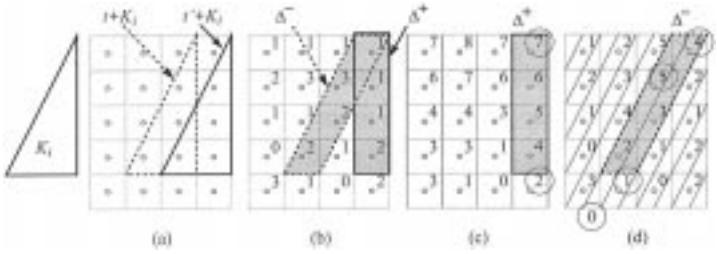


Fig. 9. Updating weights for the horizontal translation of a triangle.

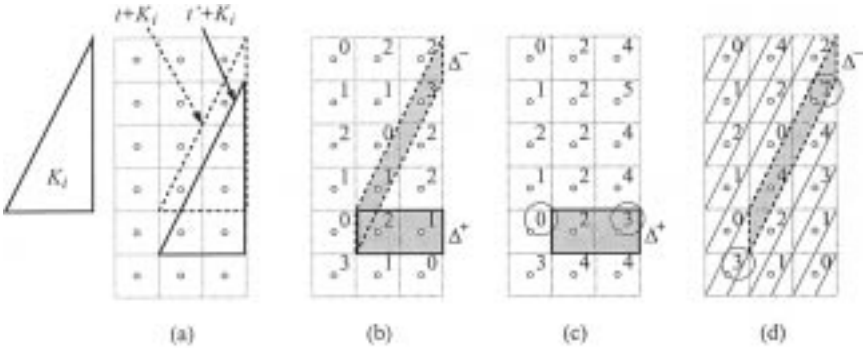


Fig. 10. Updating weights for the vertical translation of a triangle.

rectangles. [For example, the weight Δ^+ in Fig. 9(c) is $7-2=5$.]

Next, we consider the canonical digitization of Δ^-. Observe that the horizontal distance between the slanted sides in the canonical digitization of this parallelogram is exactly one unit. This follows from the facts that the original slanted lines of the triangle before and after translation are separated by a distance of one unit, and both slanted lines are rounded in the same direction to an x-intercept that is a multiple of 0.5. Thus Δ^- spans exactly two canonical strips. It suffices to compute the sum of the weights of the two parallelograms resulting from the intersection of Δ^- and these two canonical strips.

As before, we assume that the image has been preprocessed by computing the prefix sums within each canonical strip. [This is shown in Fig. 9(d).] Once these prefix sums have been computed, we compute the difference between the prefix sum of the topmost grid point in each of the two parallelograms and the topmost grid points in the strip lying immediately below each parallelogram. (Details will be discussed in Section III-C4.) If there is no pixel below the parallelogram, then the value 0 is used.

In Fig. 9(d), the weight of the left parallelogram is $5-0=5$ and the weight of the right parallelogram is $4-1=3$ and hence the weight of Δ^- is $5+3=8$. Finally the difference between Δ^+ and Δ^- is $5-8=-3$. The weight of $t+K_i$ is 8, and hence using only six arithmetic operations we determine that the weight of $t'+K_i$ is $8-3=5$.

3) Triangular Shapes: Vertical Translation: The last case to be considered is the incremental change in the weight of a right triangle primitive shape, again with a high slope, but in the case of a vertical translation. Assume that the triangle is translated vertically downward by one unit. Let $t+K_i$ denote the current placement of K_i, whose weight we know, and let $t'+K_i$ be the new placement, whose weight we wish to compute. Let $t'=t-(0,1)$ [see Fig. 10(a)]. As in the horizontal translation case, the change in weight can be expressed as the difference of the weights of a parallelogram and a rectangle. Consider two shapes Δ^+ and Δ^-. Δ^+ is a rectangle of unit vertical width lying beneath $t+K_i$. Δ^- is a parallelogram of unit vertical width lying above $t'+K_i$, with vertical left and right sides and slanted sides that are parallel to the slanted side of K_i [see Fig. 10(b)]. As before, these two shapes overlap one another, but the symmetric difference of $t+K_i$ and $t'+K_i$ is equal to the symmetric difference of Δ^+ and Δ^-.

The vertical width of Δ^+ is one unit, and hence it spans exactly one row of grid points. Its weight can be computed in constant time, using the same method described earlier for rectangles, but this time using prefex sums along the rows. [For example, the weight Δ^+ in Fig. 10(c) is $3-0=3$.]

Let s denote the slope of the slanted side of K_i. In the horizontal translation case, the slanted parallelogram was of unit width and hence spanned exactly two canonical strips. In this case the slanted parallelogram is of unit vertical width and hence of horizontal width $1/s$. By hypothesis this is a high slope primitive shape, and hence $s \geq 1$. Because each canonical strip is of horizontal width 0.5, it follows that (depending on the vagaries of rounding) the slanted parallelogram Δ^- spans either 0, 1, or 2 canonical strips. (By the way, this is why we need to distinguish between the high-slope and low-slope cases. If s were less than 1, then the parallelogram might span an arbitrarily large number of canonical strips.)

The processing is exactly the same as in the horizontal translation case, except that we determine how many canonical strips (0, 1, or 2) are spanned by Δ^- by rounding. We then compute their total weight using the prefix sums for these strips and return the total. For example, in Fig. 10(d), Δ^- spans only one canonical strip, and its total weight is computed by taking the difference between the topmost grid point and the grid just be-

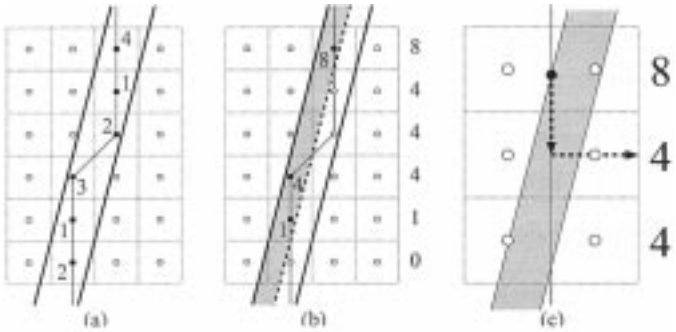


Fig. 11. Technical issues in the algorithm.

neath the parallelogram, for a weight of $7 - 3 = 4$. Combining this with the weight of Δ^+ it follows that the total weight change is $3 - 4 = -1$. Since the weight of $t + K_i$ is 8, the weight of $t' + K_i$ is $8 - 1 = 7$. The number of arithmetic operations needed to compute the updated weight is no greater than the horizontal translation case.

4) Technical Issues: There are a couple of technical issues that were not fully discussed in the previous sections. The first involves how prefix sums are computed. We consider the case of high slopes. Low slopes follow from a symmetrical argument, and horizontal and vertical strips have already been discussed. Each canonical strip is of width 0.5. We begin by pairing consecutive strips together to form a collection of disjoint *double strips* each of unit width. We can compute the grid points of the image rectangle R that lie within each wide strip by applying an appropriate modification of any standard line digitization algorithm, for example, Bresenham's midpoint algorithm [2], [7] [see Fig. 11(a)].

As we walk along this digitized line from bottom to top, we can determine which grid points lie in the left canonical strip and which to the right and update a prefix sum counter for each such strip. In Fig. 11(b), we show this for the left strip. Finally we store the prefix sums for each canonical strip as a vector with one entry for each row of the image, even if this row does not contribute a grid point to the canonical strip. This is shown on the right of Fig. 11(b).

The time to apply this to each double strip is proportional to the number of grid points in the strip. Because the double strips are of unit width, each grid point of R occurs in one double strip, and hence the total time to compute the prefix sums is proportional to the image size, which is $O(mn)$. The total space used is also $O(mn)$ by the same argument. Note that the only essential difference for the low-slope case is that prefix sums are computed and stored by columns, rather than by rows. Hence, we have the following.

Lemma 4: The preprocessing for all canonical strips for a given slope can be done in $O(nm)$ time and $O(nm)$ space.

The second technical issue is how to determine which prefix sum values are used in computing the weights of shapes Δ^- and Δ^+. For the rectangular cases, this simply involves rounding the coordinates of the appropriate side to the next smaller integer and accessing the associated prefix value. In the case of horizontal translation, the y-coordinates of the top and bottom sides of the parallelogram are simply rounded to the appropriate integer values. Because prefix sums for high-slope canonical strips are stored for each row, we can access the appropriate prefix sum values in constant time. The case of vertical translation is somewhat more involved. This is because the left and right sides of the parallelogram are vertical. Consider the case of a parallelogram for a high-slope primitive shape [as shown in Fig. 11(c)]. To access the appropriate element of the prefix sum, we compute the y-coordinate of the intersection of the vertical side with the top edge of the canonical strip. We then round this down to the next smaller integer, and access the prefix sum value associated with this row.

Combining the discussion of this and previous sections, we have the following result.

Lemma 5: After preprocessing has been completed, if the weight of the canonical digitization of a placement of a primitive shape $t + K_i$ is known, then the weight of the canonical digitization of any unit-length translation of the primitive shape, either horizontally or vertically, can be computed in constant time.

As mentioned earlier the constant factors in the time are quite small. For each of the k sides of the kernel the preprocessing involves digitizing a line of this slope, and visiting each pixel of the image once in order to compute the prefix sums. The number of primitive shapes is at most $2k + 1$, and each update step essentially involves rounding coordinates to determine the appropriate prefix sums to access, and then applying up to six arithmetic operations to these sums.

D. Canonical Convolution Algorithm

We now give the complete description of the algorithm for computing the approximate convolution. As mentioned before, the algorithm operates by computing the weights of the canonical digitizations for each of the $O(k)$ primitive shapes, and then computing the weighted sum over all these shapes. Lemma 3 states that the resulting sum is a valid digitization, and hence the resulting convolution, called the *canonical convolution*, is a valid convolution. Here is the entire algorithm, which is given the input image $I[1, \ldots, m; 1, \ldots, n]$ and the kernel polygon K with k sides.

1) Using the method of Section III-A, subdivide K into $r = O(k)$ primitive shapes denoted $K_1, K_2, \ldots, K_r$.
2) Compute the prefix sums for the rows and columns of the image in $O(mn)$ time. Initialize the convolution result image $C[1, \ldots, m; 1, \ldots, n]$ to 0.
3) For i from 1 to r, perform the following steps:
   a) if K_i is a right triangle shape, compute the prefix sums for the canonical strips for the slanted edge of K_i;
   b) by brute force, compute the weight of the canonical digitization of the placement of K_i in the lower left corner, $K_i + (1, 1)$ [see Fig. 12(a)];
   c) for y from 2 to m, do the following:
      i) compute the weight of the canonical digitization of $K_i + (1, y)$ by updating the weight of $K_i + (1, y - 1)$ through a vertical translation of one unit [see Fig. 12(b)];
      ii) for x running from 2 to n, compute the weight of the canonical digitization of $K_i + (x, y)$ by updating the weight of

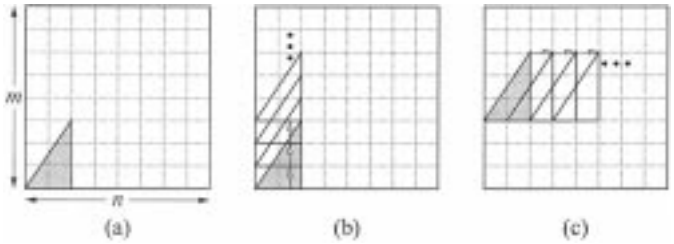


Fig. 12. Complete algorithm structure.

$K_i + (x-1, y)$ through a horizontal translation of one unit [see Fig. 12(c)];

d) as each new digitization $K_i+(x, y)$ weight is computed in steps b) and c), add the result to the appropriate index in the convolution matrix in C.

The correctness of this procedure has been established in the previous discussion. The running time of step 1) is $O(k)$. Step 2) can be performed in $O(mn)$ time. By Lemma 4, Step 3a) can be performed in $O(mn)$ time. Step 3b) can be performed, by any algorithm for digitizing convex polygons [7], [10]. The running time of such an algorithm is proportional to the number of pixels covered by $K \cap R$, and this can be at most $O(mn)$. By Lemma 4, steps 3c-i) and 3c-ii) take $O(1)$ time each, and since they are performed $O(mn)$ times, the total time for each iteration of the loop in step 3) is $O(mn)$. Since this loop is repeated for each of the $O(k)$ primitive shapes, step 3) takes total time $O(kmn)$. Hence, the total running time is $O(kmn)$.

As mentioned in Lemma 4, the space requirements are $O(mn)$ per slope. Since we can discard the prefix sums computed in step 3a) after their use in step 3c), we need to keep only three copies of the prefix sums at any time (one for the slanted slope, one for the rows, and one for the columns). Thus the total space requirements are $O(3mn) = O(mn)$. This establishes our main result, Theorem 1.

IV. Conclusions

We have presented an efficient algorithm for computing approximate (valid) convolutions for binary kernels that are modeled as convex k-sided polygons. The algorithm runs in $O(kmn)$ time on an $m \times n$ image, irrespective of the area or perimeter of the kernel. Our approach is based on a special type of digitization of the kernel, called a canonical digitization, which varies from one placement to the next. We have shown that canonical digitizations can be updated efficiently through the use of prefix sums. Although we have showed that the constants hidden by the O-notation are reasonably small, this method would not be competitive with existing convolution algorithms for small or rectangular kernels. However, applications involving large convex kernels should benefit from this approach.

Some interesting open problems are suggested by this work. One question is whether these techniques can be generalized to convolutions involving kernels that are multi-valued (grayscale) or to nonconvex simple polygons. In theory, such a kernel could be subdivided into single-valued, convex parts. However, Lemma 3, which establishes the validity of the canonical digitization, does not generalize immediately to collections of convex polygons.

Another question is: If valid convolutions are used to approximate morphological operations (such as dilation), what can be said about the properties of the resulting shapes, as compared with their exact counterparts, and what is the magnitude of the resulting discrepancy in practice. As mentioned at the end of Section III-B, by placing canonical lines closer together it is possible in increase the accuracy of the digitization to any desired level, but since this would result in more canonical strips, a proportional increase in computation time and space would be paid.

Acknowledgment

The authors would like to thank Dr. A. Rosenfeld for his comments and for making them aware of the box-filtering literature. They would also like to thank the anonymous referees for making a number of valuable suggestions, which improved the clarity of this paper.

References

[1] A. Amir, A. Efrat, P. Indyk, and H. Samet, "Efficient algorithms and regular data structures for dilation, location and proximity problems," in *Pro. 40th Annu. IEEE Symp. Foundations Computer Science*, 1999.

[2] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Syst. J.*, vol. 4, pp. 25–30, 1965.

[3] P. Burt, "Fast filter transforms for image processing," *Comput. Graph. Image Process.*, vol. 16, pp. 20–51, 1981.

[4] O. I. Camps, T. Kanungo, and R. M. Haralick, "Grayscale structuring element decomposition," *IEEE Trans. Image Processing*, vol. 5, pp. 111–120, 1996.

[5] D. E. Dudgeon and R. M. Mersereau, *Multidimensional Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[6] E. Fiume, "Coverage masks and convolution tables for fast area sampling," *Graph. Models Image Processing*, vol. 53, pp. 25–30, 1991.

[7] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*. Reading, MA: Addison-Wesley, 1990.

[8] P. D. Gader, "Separable decompositions and approximations of greyscale morphological templates," *CVGIP: Image Understand.*, vol. 53, pp. 288–296, 1991.

[9] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Reading, MA: Addison-Wesley, 1992.

[10] D. Hearn and M. P. Baker, *Computer Graphics: C Version*. Englewood Cliffs, NJ: Prentice-Hall, 1997.

[11] H. J. A. M. Heijmans, *Morphological Image Operators*. New York: Academic, 1994.

[12] A. K. Jain, *Fundamentals of Digital Image Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[13] T. Kanungo and R. M. Haralick, "Vector-space solution for a morphological shape-decomposition problem," *J. Math. Imag. Vis.*, vol. 2, pp. 51–82, 1992.

[14] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu, "Approximating large convolutions in digital images," in *Proc. Vision Geometry VII*, vol. 3454, R. A. Melter, A. Y. Wu, and L. J. Latecki, Eds., 1998, pp. 216–227.

[15] J. Kim and Y. Kim, "Efficient 2-D convolution algorithm with the single-data multiple kernel approach," *Graph. Models Image Process.*, vol. 57, pp. 175–182, 1995.

[16] S. U. Lee, "Design of SVD/SGK convolution filters for image processing," Univ. Southern California, Los Angeles, Tech. Rep. 950, 1980.

[17] M. J. McDonnell, "Box-filtering techniques," *Comput. Graph. Image Process.*, vol. 17, pp. 65–70, 1981.

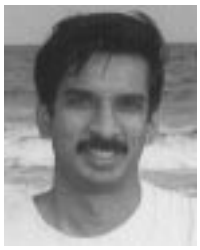
[18] D. P. O'Leary, "Some algorithms for approximating convolutions," *Comput. Vis., Graph., Image Process.*, vol. 41, pp. 333–345, 1988.

[19] H. Park and R. T. Chin, "Optimal decomposition of convex morphological structuring elements for 4-connected parallel array processors," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16, pp. 304–313, 1994.

[20] ——, "Optimal decomposition of arbitrarily-shaped morphological structuring elements," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, pp. 2–15, 1995.
[21] J. Serra, *Image Analysis and Mathematical Morphology*. New York: Academic, 1982.
[22] ——, *Image Analysis and Mathematical Morphology Vol. 2: Theoretical Advances*. New York: Academic, 1988.
[23] K. Sivakumar and J. Goutsias, "Binary random fields, random closed sets, and morphological sampling," *IEEE Trans. Image Processing*, vol. 5, pp. 899–912, June 1996.
[24] P. Sussner and G. X. Ritter, "Decomposition of gray-scale morphological templates using the rank method," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19, pp. 649–658, June 1997.
[25] G. Wolberg and H. Massalin, "Fast convolution with packed lookup tables," in *Graphics Gems IV*, P. Heckbert, Ed. New York: Academic, 1994, pp. 447–464.
[26] X. Zhuang and R. M. Haralick, "Morphological structuring element decomposition," *Comput. Vis., Graph., Image Process.*, vol. 35, pp. 370–382, 1986.

David M. Mount received the Ph.D. degree in computer science from Purdue University, West Lafayette, IN, in 1983.

He is a Professor with the Department of Computer Science at the University of Maryland, College Park, with a joint appointment to the Institute for Advanced Computer Studies. His primary research interests include the design, analysis, and implementation of data structures and algorithms for geometric problems, particularly problems with applications in image processing, pattern recognition, information retrieval, and computer graphics. He is an Associate Editor for *Pattern Recognition*.



Tapas Kanungo (SM'01) received the Ph.D. degree from the University of Washington, Seattle.

He is a Research Staff Member at the IBM Almaden Research Center, San Jose, CA. Previously, he was a Co-director of the Language and Media Processing Lab at the University of Maryland, College Park. His current interests are in information extraction and retrieval and document analysis.

Nathan S. Netanyahu (S'89–M'90) received the B.Sc. and M.Sc. degrees in electrical engineering from the Technion—Israel Institute of Technology, Haifa, and the M.Sc. and Ph.D. degrees in computer science from the University of Maryland, College Park.

He is currently a Senior Lecturer with the Department of Mathematics and Computer Science at Bar-Ilan University, Ramat-Gan, Israel, and is also affiliated with the Center for Automation Research, University of Maryland. His main research interests are in the areas of algorithm design and analysis, computational geometry, image processing, pattern recognition, remote sensing, and robust statistical estimation. He currently serves as an Associate Editor for *Pattern Recognition*.

Christine Piatko received the B.A. degree in computer science and mathematics from New York University in 1986, and the M.S. and Ph.D. degrees from Cornell University, Ithaca, NY, in 1989 and 1993, respectively.

She is currently a Computer Science Researcher with the Johns Hopkins University Applied Physics Laboratory, Laurel, MD. Her research interests include computational geometry and information retrieval.

Ruth Silverman received the Ph.D. degree in mathematics from the University of Washington, Seattle, in 1970.

She is a retired Professor from the Department of Computer Science, University of the District of Columbia, and a Visiting Professor at the Center for Automation Research, University of Maryland, College Park.

Angela Y. Wu (M'86–SM'90) received the Ph.D. degree in computer science from the University of Maryland, College Park, in 1978.

Since 1980, she has been a Professor with the Department of Computer Science and Information Systems, American University, Washington, DC.