Approximating Large Convolutions in Digital Images

Tapas Kanungo[a] David M. Mount[b] Nathan S. Netanyahu[c] Christine Piatko[d]
Ruth Silverman[e] Angela Y. Wu[f]

ABSTRACT

Computing discrete two-dimensional 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, computes a valid convolution in time $O(kmn)$ 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 efficiently as the kernel is moved from one position to another across the image.

Keywords: Digital convolutions, correlations, digital morphology, digital geometry, approximation algorithms.

1. INTRODUCTION

A fundamental problem in image processing is that of computing *discrete convolutions*. Consider and image, which is given as a 2-dimensional $m \times n$ array I of real numeric values. We will think of an image as defining a function $I : \mathbf{Z}^2 \to \mathbf{R}$, where

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

(The transposition of g_x and g_y reflects the convention that arrays are indexed first by row then by column.) The *kernel* of the convolution is a $q \times r$ image array K. The *discrete convolution*[5] of I by K, denoted $I * K$, is defined to be

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

It is common to embed I and K within larger images to avoid wraparound effects, which will not be of concern to us here.

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.[9,10] For example, the *dilation* of a digital shape described by a 0–1 image I by a digital set K, can be expressed by computing the convolution

[a] Tapas Kanungo, Center for Automation Research, University of Maryland College Park, Maryland. Email: `kanungo@cfar.umd.edu`.
[b] David M. Mount, Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland. Email: `mount@cs.umd.edu`. The support of the National Science Foundation under grant CCR–9712379 is gratefully acknowledged.
[c] Nathan S. Netanyahu, Center for Automation Research, University of Maryland, College Park, and Center of Excellence in Space Data and Information Sciences, (CESDIS) Code 930.5, Space Data and Computing Division, NASA Goddard Space Flight Center, Greenbelt, Maryland. Email: `nathan@cfar.umd.edu`. The support of the Applied Information Sciences Branch (AISB), Code 935, NASA/GSFC, under contract NAS 5555-37 is gratefully acknowledged.
[d] Christine Piatko, Research and Technology Development Center, The Johns Hopkins University Applied Physics Laboratory, Laurel, Maryland. Email: `christine.piatko@jhuapl.edu`.
[e] Ruth Silverman, Department of Computer Science, University of the District of Columbia, Washington, DC, and Center for Automation Research, University of Maryland, College Park, Maryland. Email: `ruth@cfar.umd.edu`.
[f] Angela Y. Wu, Department of Computer Science and Information Systems, The American University, Washington, DC. Email: `awu@american.edu`.

$I*(-K)$, and then thresholding this image so that all positive values are mapped to 1. Binary convolutions are all is also useful in template matching[5] 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. Let $-K$ denote the set of points $\{-p \mid p \in K\}$. We can interpret $I * K(p)$ as placing a copy of $-K$ at location p of the image, and then computing the weight of the pixels that are overlapped by $-K$.

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 be reasonably large. The question is whether we can improve on the qr factor, especially when qr is large.

Some methods are based on the idea of decomposing convolutions involving large kernels as a combination of a number of convolutions involing small kernels.[7,8] Other methods involve using table-lookup to avoid the cost of multiplication.[3,11] Burt proposed a technique based on the use of quadtrees.[2] However, these methods only improve running times by a constant factor. A common approach for computing convolutions for large kernels is first to compute the Fourier transform of the image and kernel, denoted 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.[5] 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 a $m \times n$ 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 operator is a convex polygon or can be approximated by one. If k is small, then this can be significantly faster than existing approaches for large kernels.

2. DEFINITIONS AND NOTATION

We begin with a number of definitions. Let $\mathbf{Z}^2$ denote the set of ordered pairs of integers, called *grid points*. Given $g \in \mathbf{Z}^2$, define $\pi(g)$ to be an open unit square centered at g, called g's *pixel*. The set $\pi(g)$ over all $g \in \mathbf{Z}^2$ subdivides the real plane into a collection of unit squares with pairwise disjoint interiors.

Let $\mathbf{R}^2$ denote the set of ordered pairs of reals. Given a set $P \subset \mathbf{R}^2$ and $t \in \mathbf{R}^2$, let $t + P$ denote the translate of P by t, that is,

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

We will call this the *placement* of P at t. Let $-P = \{-p \mid p \in P\}$, and define $t - P$ to be $t + (-P)$. Given a set $P \subset \mathbf{R}^2$, we are interested in ways of mapping P to a set of grid points, called a *digitization*. We say that a digitization $D(P) \subseteq \mathbf{Z}^2$ 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 disjoint of P the corresponding grid point is not in the digitization. More formally,

$$\begin{aligned} \pi(g) \subseteq P \quad &\Rightarrow \quad g \in D(P) \\ (\pi(g) \cap P = \emptyset) \quad &\Rightarrow \quad g \notin D(P). \end{aligned}$$

Pixels that partially overlap P either may or may not be in a valid digitation. An example of a valid digitization is the *midpoint digitization*, denoted $D^m(P)$ 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 a midpoint digitization.

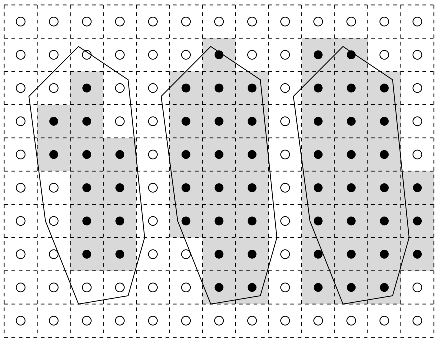

Figure 1. Valid digitizations.

Given an image I and any subset of grid points, $G \subset \mathbf{Z}^2$, 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,

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

K is called the *kernel* of the convolution. A *valid convolution* of I by K is defined in the same way, but where $D^m(t-K)$ is replaced by any valid digitization of $t - K$. Note that because valid digitizations are not unique, different placements may be digitized differently. The concepts of midpoint and valid convolutions are generalizations of the binary convolution, except that the kernel of the convolution is assumed to be modeled by a convex polygon. We assume that the convolution C is defined over the same m by n rectangle as the image, but our algorithm can easily be generalized to compute a representation of C over any expanded region.

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 each element of the convolution is the weight of some translate of K. We assume that the image is presented as a two dimensional rectangular array with m rows and n columns. The pixels of the image form a subdivision into squares of the *image rectangle*:

$$R = \{(x, y) \mid 0.5 \leq x \leq n + 0.5, \quad 0.5 \leq y \leq m + 0.5\}.$$

All grid points outside of this rectangle are assumed to have value 0. We assume that the kernel K is presented by a counterclockwise cyclic list of its vertices.

3. THE ALGORITHM

Let's begin with an overview of the algorithm for computing a valid convolution. Let k denote the number of sides of the kernel K, and recall that m and n denote the height and width, respectively, of the image. The algorithm consists of a number of stages. The first involves preprocessing the kernel K, and results in a representation of K as the weighted sum of a set of $O(k)$ *primitive shapes.* Each primitive shape is either an axis-aligned rectangle or a right triangle with a horizontal and vertical side. The nonorthogonal side of a right triangle is called its *slanted side*.

The second stage involves preprocessing the image. We create at most $O(k)$ 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 partitions the image rectangle into a collection of thin regions, called *canonical* *strips*. We digitize each strip using a midpoint digitization and preprocess each by a method to be described later. The resulting structure has the property that, in constant time, it is possible to count the total weight of a trapezoid bounded by a strip and two lines that are either horizontal and vertical. We will show that this structure can be built in time and space of $O(mn)$ for each side of K.

After these preprocessing phases, 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*. These digitizations will be defined in such a way that they are pairwise disjoint, and their union is a valid digitization of K. 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 when placement is translated by one unit distance, either horizontally or vertically. This is done in constant time, by applying the structure described in the previous paragraph. Thus, the entire convolution for one shape can be computed in $O(mn)$ time, by computing an initial weight, and then shifting the shape pixel-by-pixel over the entire image. By applying this to each primitive shape and then computing the weighted sum of these convolutions, the convolution by K can be computed in $O(kmn)$ total time. The various elements of the algorithm are explained in detail in each of the following subsections.

3.1. Decomposition into Primitive Shapes

As mentioned above, the first stage of the algorithm involves decomposing K into $O(k)$ *primitive shapes*. The decomposition 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. For each vertex of K imagine shooting two bullets horizontally and vertically away from the interior of K, until either hitting the bounding box B or hitting a prior bullet 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.

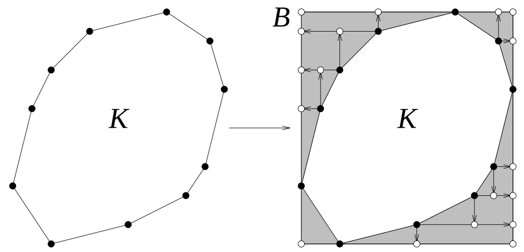

Figure 2. Decomposing the kernel into primitive shapes.

Let k denote the number of sides of K and let r denote the number of such shapes. Observe that each time a bullet path is shot, it splits some region into two subregions. Since there are at most $2k$ bullets shot (at most two per vertex), and including the bounding box B, it follows that $r \leq 2k+1 = O(k)$. Let $K_1, K_2, \ldots, K_r$ denote these shapes. Let $\mathrm{bnd}(K_i)$ denote the boundary of K_i.

First we observe that K can be expressed as a weighted sum of these shapes. The following lemma establishes this fact for all nonboundary points. The bounding box B is assigned a weight of $+1$, and all the other primitive shapes are assigned a weight of -1. 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 $p \in \mathbf{R}^2$, define the weight of p to be $W(p) = \sum_{i=1}^{r} w_i \cdot K_i(p)$.

LEMMA 2. *For all $p \in \mathbf{R}^2$, $p \notin \cup_{i=1}^{r} \mathrm{bnd}(K_i)$,*

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

Proof. Points outside of B are clearly not in K and have weight 0. Points within the interior of K have a weight of 1 since they lie inside B but outside all the other primitive shapes. Because the interiors of the primitive shapes other than B are pairwise disjoint and cover $B \backslash K$, a point of $B \backslash K$ that does not lie on the boundary of any primitive shape has weight $1 - 1 = 0$. $\square$

Points that lie on the boundary of the primitive shapes are messier to deal with, since the result holds only if each is assigned to a unique primitive shape. We will deal with this by a tie breaking rule, which we term the *nonintegrality assumption*. We assume that for each $t \in \mathbf{Z}^2$, the coordinates of the vertices of $t + K$ are not integers. With this assumption, there are no grid points that lie on the horizontal or vertical boundaries of the primitive shapes. This can be handled by assuming that K is given with arbitrary (possibly integer) coordinates, and to assume that each integer coordinate has been perturbed by adding an infinitessimal value $\epsilon > 0$. This can be implemented symbolically without actually modifying the coordinates of K. When making a comparison between a coordinate and integer quantity, if the values are not equal, then the result of the comparison is unchanged. If the values are equal, then we break the tie as if the coordinate value has been increased infinitessimally. Slanted sides bounding primitive shapes may also pass through grid points. These can also be handled by a symbolic perturbation. But this is not needed, because our definition of valid digitization provides the freedom to either select such a point or not.

3.2. Canonical Lines and Canonical Digitizations

For each $t \in \mathbf{Z}^2$, and each primitive shape of K_i, we define a special digitization of the placement $t + K_i$, called the *canonical digitization* and denoted $D^c(t + K_i)$. This will be done in such a way that the weighted sum of the 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 set of grid points lying within the intersection of $t + K_i$ and the image rectangle R. (By our nonintegrality assumption, no grid points lie on the boundary of this set.) If K_i is a right triangle, then main issue is how to digitize its slanted side. 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 is less than 1, then we call this a *low-slope triangle*, and otherwise we call it a *high-slope triangle*. Let us consider the high-slope case. The low-slope case is handled in an 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 sorted sequence set of lines that intersect the image rectangle R, whose slope is s, and whose x intercept is an integer multiple of 0.5. Such a sequence is illustrated in Fig. 3 below. 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 reader may wonder about the reason for the choice of 0.5 as the separation distance. This will be discussed later.)

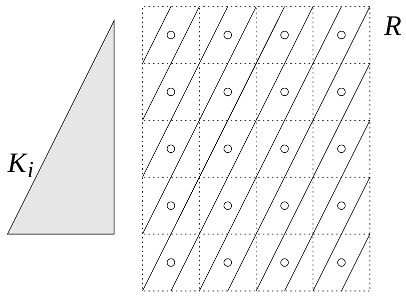

Figure 3. Canonical lines.

LEMMA 3. *For any slope s, the number of canonical lines and number of canonical strips 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 2. By easy calculations, it can be seen that any line of slope s intersects R if and only if its x-intercept lies within the interval

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

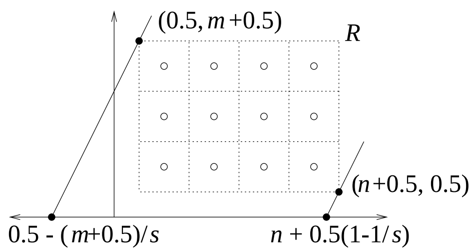

Figure 4. The range of canonical lines.

(See Fig. 4.) Letting $2I$ denote the interval resulting by doubling the endpoints of I, it follows that the set of x-intercepts of the canonical lines for slope s is $\{i/2 \mid i \in \mathbf{Z} \cap 2I\}$. Since $s \geq 1$, the width of I satisfies

$$\mathrm{width}(I) \;=\; n + 0.5\left(1 - \frac{1}{s}\right) - 0.5 + \frac{m + 0.5}{s} \;\leq\; n + m + 0.5.$$

An interval of width w can be subdivided into at most $w + 1$ strips of unit length. Since the canonical strips are of width 0.5, it follows that the number of canonical strips is at most $2(n + m) + 3$. The number of canonical lines is at most one greater than this. □

The *canonical digitization* of $t + K_i$ 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 is a rectangle. In the latter case, the canonical digitization is defined to be the set of grid points lying within this rectangle. Otherwise, assume for concreteness that the triangle lies to the right of the slanted line. Select the nearest canonical line ℓ^* that lies on or to the right of ℓ (see Fig. 5(a)). This can be accomplished by computing the x-intercept of ℓ, and then rounding to the next larger integer multiple of 0.5. In general, ℓ is rounded towards the interior of the triangle it supports.

The canonical digitization is defined to be the set of grid points that lie within the intersection of the bounding rectangle of $t + K_i$, the image rectangle R, and either on or to the right of ℓ^*. (These are shown as black points in Fig. 5(a).) Notice that by rounding to the canonical line lying towards the interior of the triangle, the canonical digitization will generally consist of a subset of the grid points lying in $t + K_i$. Thus, it will generally be a subset of the midpoint digitization of $t + K_i$. (For example, in Fig. 5(a) there are two grid points that are in the midpoint digitization but have been excluded from the canonical digitization.)

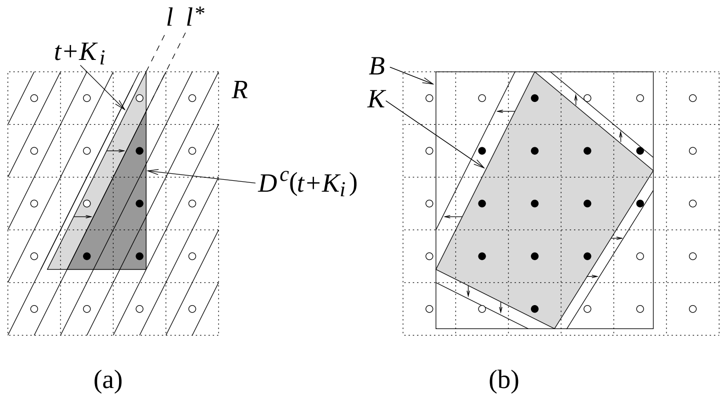

Figure 5. Canonical digitization.

The canonical digitization for the entire shape is defined by taking the weighted sum of points in the canonical digitizations of the primitive shapes. 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,

$$D^c(t + K) = \sum_{i=1}^{r} w_i \cdot D^c(t + K_i).$$

In other words, a pixel lies in the canonical digitization if the weighted sum of sets containing this pixel is 1, and lies outside if the weighted sum is 0. (An example is shown in Fig. 5(b). The grid points belonging to the final digitization are shown as black points in this figure.) Since the primitive shapes lie outside of K, observe that the sides are rounded away from the interior of K, and hence the points of the canonical digitization of K will generally be a superset of the points in the midpoint digitization of K. Next we show that the result is a valid digitization of K.

Lemma 4. *For any convex polygon K, and any vector $t \in \mathbf{Z}^2$, 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 $g \in \mathbf{Z}^2$. Recall that $\pi(g)$ is the pixel (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) If $\pi(g)$ intersects the boundary of $(t + K) \cap R$ then g is assigned a weight of either 0 or 1.

Recall by our nonintegrality assumption that no grid point lies on the horizontal or vertical boundary of any primitive shape. Also 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 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 digitizations for that shape, the canonical digitization for no primitive shape can contain this point. Thus, its total weight is 1.

To establish (2), consider a grid point g that is not in $(t + K) \cap R$. If g lies entirely outside of the bounding box $t + B$, or outside of the image bounding box R, then it is not allocated to any canonical digitization, and so it is given a weight of 0. Otherwise, the g lies within this intersection, but lies within some primitive shape $t + K_i$. We consider two cases.

First, if g lies within the canonical digitization of this shape, then we assert that g will not be in the canonical digitization of any other primitive shape (other than B) and hence it will be assigned a weight of $1 - 1 = 0$. To see this, first recall that (ignoring the bounding box B) the primitive shapes have disjoint interiors and have no grid points on their boundaries. Thus the midpoint digitizations of the shapes are pairwise disjoint. Recall that the canonical digitization is a subset of the midpoint digitization, and hence they too are pairwise disjoint, implying that g is in no other canonical digitization.

The second case is that the point g is not in the canonical digitization of $t + K_i$. We claim that this cannot happen under the hypothesis of (2). Since g lies within $(t + K_i) \cap R$, and since there are no grid points on the horizontal and vertical sides of primitive shapes, g fails to lie in the canonical digitization because it lies outside of the associated canonical line. If this is a high-slope line, then the canonical line lies within a horizontal distance of 0.5 of the supporting line for the slanted side. Since the point lies outside the canonical line, but inside the supporting line (since it is inside the primitive shape), it follows that the horizontal distance from the point to the slanted side's supporting line is at most 0.5. However, this together with the fact that g lies in $t + K_i$ implies that the associated side of K intersects g's pixel. This implies that g's pixel does not lie entirely outside of K as hypothesized. (In the case of a low slopes the argument is the same, but the vertical distance is at most 0.5.) This establishes (2).

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 $(t + B) \cap R$ then it is assigned a weight of 0. Otherwise g will be assigned an initial weight of 1 because it lies in the bounding box. We claim that g can be in the canonical digitization of at most one other primitive shape. This is because (by our nonintegrality assumption) 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, then its final weight is 0, and otherwise its weight is 1. This establishes (3). □

The reason for rounding lines towards the interior of the primitive shape triangles and the choice of 0.5 as the separation distances between canonical lines is evident from the preceding proof. The proof of fact (2) relied on the fact that the horizontal distance is half the width of a pixel. The proof of fact (3) relied on the fact that canonical digitizations are subsets of the associated midpoint digitizations. If this were not the case, then a pixel interior to K but intersected by two sides of K (say one on its left and one on its right) might be assigned to two canonical digitizations. 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.

3.3. Updating Canonical Digitizations

The main algorithmic tasks needed to compute the digitization are (1) how to compute the weight of the canonical digitization of a single placement of a primitive shape, and (2) how to update the weight of the canonical digitization, when the shape is shifted one unit distance, either horizontally or vertically. The first task can be accomplished by adapting any standard algorithm for digitizing convex polygons.[4] The second task will be addressed in the remainder of this section.

Rectangular Shapes. Let us consider the case of a rectangle primitive shape K_i, since it is the simplest. For concreteness we consider the case of a translation to the right by one unit. Let t and t' be two placement vectors such that $t'-t=(1,0)$. (The other unit shifts are handled similarly.) 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 of $(t+K_i)$ and $(t'+K_i)$ is the union of two congruent rectangles each of unit width, the left one r belonging to $t+K_i$ and the right on r' belonging to $t'+K_i$. (See Fig. 6(a).) If the width of K_i is less than one, then think of these unit width rectangles as overlapping each other. The change in the weight between the two placements is easily seen to be difference in the weights of r' and r. Thus we have

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

The incremental change in weight can be computed in constant time once we know the weights of r and r'. There is a simple way to do this. First we preprocess the image. For each column, and for each grid point we store the total weight of image points in the same column having equal or smaller y-value. Following common usage in the field of parallel algorithms, we call this a *prefix sum*. (Fig. 6(b) and (c) show the image weights and the column prefix sums, respectively.) Then to compute the weight within column rectangle, we take the difference of the prefix sum of the topmost grid point in the rectangle and of the grid point just below the bottom of the rectangle (or zero, if the rectangle extends to the lowest pixel in the column). In the figure, the weights of r and r' are $4-1=3$ and $6-2=4$, respectively. Thus the net change in weight between placements t and t' is $4-3=1$.

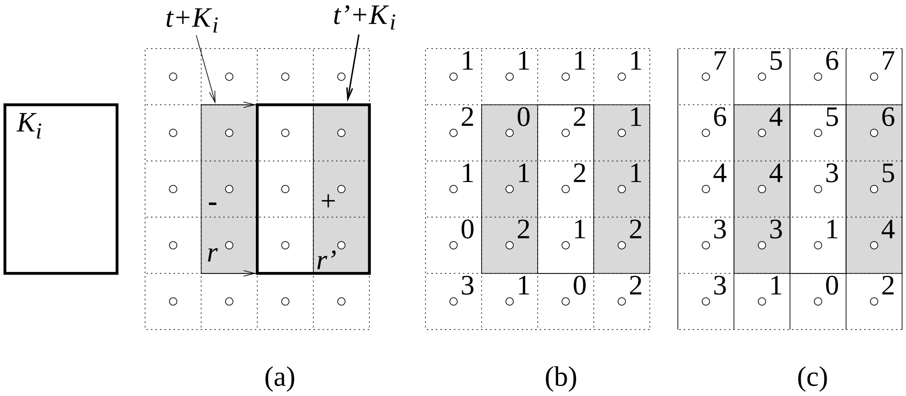

Figure 6. Updating weights for the translation of a rectangle.

The prefix sums for each column can be computed by a simple scan in $O(m)$ time, implying that all the prefix sums can all be computed in $O(mn)$ total time. The weight of the canonical digitization of any rectangle of width one 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 to return.

A vertical translation by a unit length 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.

Triangular Shapes: Horizontal Translation. Next we consider how to generalize this to update the weight of the canonical digitization of the placement of right-triangle primitive shape K_i. We will assume that K_i is a right triangle whose slanted side has a slope that is positive and greater than or equal to one. The cases for negative and/or low slopes are handled similarly.

Let us consider the case of horizontal translation for now, and 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. 7(a). Assume that the figure shows the triangles after the slanted sides have been rounded to the appropriate canonical lines.) It is easy to see that the net change in weight can be expressed as the sum and difference of the two lightly shaded trapezoids shown in the figure. One trapezoid has vertical parallel sides aligned with the right side of the shape and is added. The other has parallel sides aligned with the slanted side and is subtracted. The dark shaded region shown in the figure is both added and subtracted, and so its net weight change is zero.

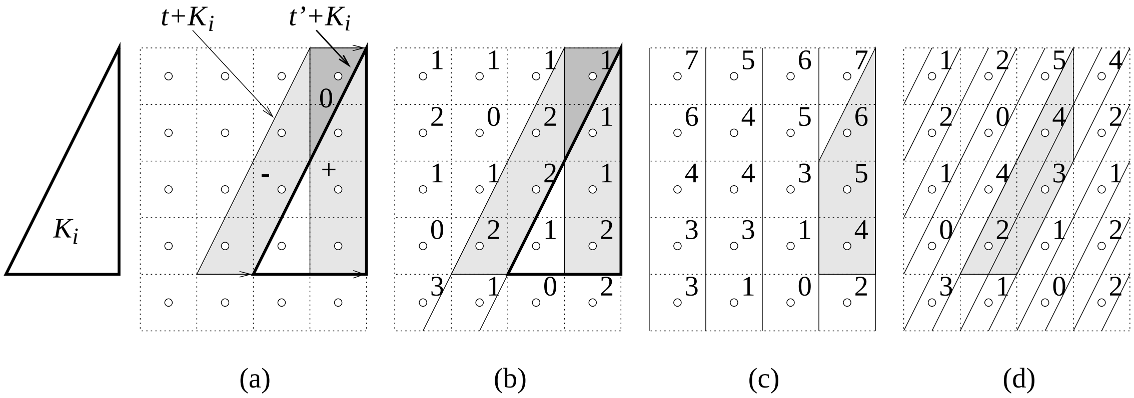

Figure 7. Updating weights for the horizontal translation of a triangle.

The horizontal width of the trapezoid along the right side is exactly one unit, and hence it spans exactly one vertical column of grid points. Its adjusted weight can be computed using the same method described earlier for rectangles. In particular, assuming that the prefix sums for the columns have been computed, it suffices to determine the topmost grid point of the column lying within the trapezoid and the topmost grid point of the column lying immediately below the trapezoid. The only added complication here is that the top side of the trapezoid is not horizontal, as it was in the rectangle case. Nonetheless, by computing the y-coordinate of the intersection point of the slanted top side of the trapezoid with the line containing the grid points and rounding to the next smaller integer value, we can compute this grid point and its prefix weight in constant time. (For example, the weight of the vertical trapezoid shown in Fig. 7(c) is $6 - 2 = 4$.)

Next we consider the canonical digitization of the slanted trapezoid. We assert that the horizontal distance between the slanted sides in the canonical digitization of this trapezoid is exactly one unit. This follows from the facts that (1) the original slanted lines of the triangle before and after translation are separated by a distance of one unit, (2) the canonical lines for high slopes are separated by a horizontal distance of 0.5, and (3) both slanted lines are rounded in the same direction.

Because the horizontal width of the slanted trapezoid is one unit, it spans exactly two canonical strips. To compute the weight of the trapezoid, it suffices to compute the sum weights of the pixels lying in the two *subtrapezoids* defined by intersection of the two canonical strips and the trapezoid. First, we preprocess the image by computing the prefix sums within each canonical strip. (This is shown in Fig. 7(d)). Once these prefix sums have been computed, we compute the difference of the prefix sum of the topmost grid point in each subtrapezoid and topmost grid point in the strip lying just below each subtrapezoid. (In the figure, the result is $(4 - 0) + (3 - 1) = 6$.)

Here is a more detailed explanation of the process. For each row of the figure and for each canonical strip the preprocessing phase computes the weighted sum of the grid points lying on or below this row in the strip. (For the low-slope case, the sums are associated with the columns of the image.) Notice that not every row has a grid point lying in the strip. This issue will be addressed in the proof of Lemma 5. To compute the points lying just below the bottom of the subtrapezoids in each strip, compute the y-coordinate of the bottom side of $K_i + t$, and then round down to the next smaller integer to get the appropriate row. To compute the topmost grid point in a subtrapezoid, first round the right vertical side of the trapezoid to the next smaller integer x-coordinate, so that it coincides with a column of grid points. Compute the y-coordinate of the intersection of this vertical line with the top line of the canonical strip. Round this y-coordinate down to the next smaller integer. It is not hard to see that any grid point lying in the subtrapezoid must lie on or below this row. This process is performed for both subtrapezoids, and the two weights are added together.

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. Suppose that the triangle is translated vertically upwards by one unit, from placement t to t', where $t' - t = (0, 1)$. (See Fig. 8(a).) Let s denote the slope of the slanted side. As in the horizontal case, the change in weight can be expressed as the weight of two trapezoids. In this case one trapezoid is aligned with the bottom of K_i and the other along the slanted side. The weight of the bottom trapezoid can be handled in a symmetrical way as was used in the horizontal case.

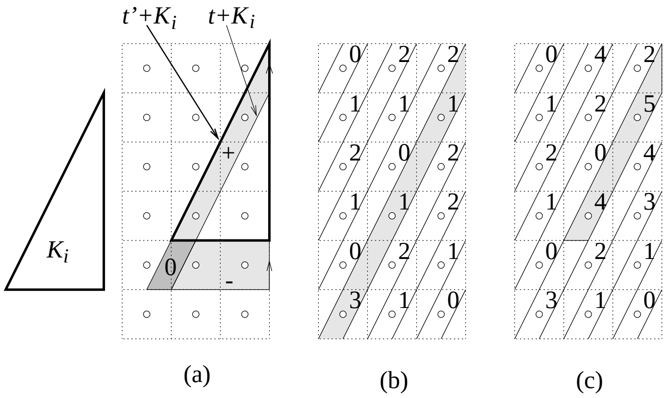

Figure 8. Updating weights for the vertical translation of a triangle.

The slanted side is somewhat different though. A convenient property of the horizontal case was that the slanted trapezoid spanned exactly two canonical strips, and hence the computation could be broken down to computing the weights of exactly two subtrapezoids. However, the horizontal distance between a high-slope line and its vertical translate of one unit is $1/s$, which is not generally an integer. We claim that this does not really pose any significant added complexity. In this case, since the horizontal distance between the two canonical lines is $1/s$, and since $s \geq 1$, it follows that the horizontal distance between the slanted lines is at most 1. Because the two lines are rounded in the same direction, after rounding, the slanted-side trapezoid will generally span either 0, 1, or 2 canonical strips. Thus the number of subtrapezoids is not fixed, but will either be 0, 1, or 2. (For example, in Fig. 8(b) there is only one subtrapezoid.) Thus, the procedure described for the horizontal case can be applied here as well. The only change is to determine actual the number of canonical strips that are spanned by the slanted sides after rounding.

Summary. The results of this section are summarized in the following two lemmas.

LEMMA 5. *The preprocessing for all canonical strips for a given slope can be done in $O(nm)$ time and $O(nm)$ space.*

Proof. We will consider the case of high slopes. Low slopes follow from a symmetrical argument, and horizontal and vertical strips have already been discussed. By Lemma 3 the number of canonical strips for any slope is $O(m+n)$. We will establish the result first for strips of horizontal width one, and then we modify this for strips of width 0.5.

For each canonical strip of width one, each row of the image intersects the strip in a single grid point (assuming consistent rounding). We can determine these grid points by computing the point at which the canonical line intersects the image rectangle R, and then applying an appropriate modification of any standard line digitization algorithm, (for example, Bresenham's algorithm[1,4]) to the canonical lines defining the boundary of the strip. As we digitize the line we can maintain the prefix sum and assign it to each pixel in the strip. Such an algorithm takes time proportional to the number of pixels processed within the strip. The total time over all strips will be equal to the number of strips plus the total number of pixels processed. Because the canonical strips for any slope cover every pixel in the image exactly once, it follows that the total time to process all the strips for this slope will be $O(mn)$. Since each pixel holds a single prefix value the space is also $O(mn)$.

We can modify this scheme for canonical strips of width 0.5 as follows. Suppose that we have already digitized the canonical strips of unit width. For each such strip, we digitize the canonical line lying midway between the sides of the strip, determining which grid points of the strip lie on the left side and which lie on the right side of this midway line. Again this can be done using any line digitization algorithm. We replace the single prefix sum value in each pixel with a pair of values, one for the left substrip and one for the right substrip. Since each row of the original strip contributed a single grid point to the strip, there will be an entry for each substrip and for each row. As before this can be performed in $O(mn)$ time and space. $\square$

The following lemma has already been established in the discussion of this section.

LEMMA 6. *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 shape, either horizontally or vertically, can be computed in constant time.*

3.4. The Canonical Convolution Algorithm

We can now complete the 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 4 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 $I[1..m, 1..n]$ and the kernel polygon K, with k sides.

(1) Using the method of Section 3.1, subdivide K into $r = O(k)$ primitive shapes, $K_1, K_2, \ldots, K_r$ and their associated weights w_i.

(2) Compute the prefix sums for the rows and columns of the image in $O(mn)$ time. Initialize the convolution image $C[1..m, 1..n]$ to 0.

(3) For i running from 1 to r, do the following steps:

   (a) By brute force compute the canonical digitization of the placement of K_i at the placement $t_0 = (1, 1)$.

   (b) If K_i is a right triangle, let s denote the slope of its slanted side. Compute the canonical lines for this slope (as defined in Section 3.2) and compute prefix sums for the resulting canonical strips using the method outlined in Lemma 5.

   (c) For the translation vector t zig-zagging along the grid points of the rows and columns of the image rectangle R, incrementally update the weight of the canonical digitization of the new placement of K_i, using the methods described in Section 3.3. From Lemma 6 we know that each update can be computed in constant time. Add each of the resulting weights, times w_i into the convolution image $C[t]$.

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. Step (3a) can be performed, by any algorithm for digitizing convex polygons.[4,6] The running time of such an algorithm is proportionl to the number of pixels covered by $K \cap R$, and this can be at most $O(mn)$. By Lemma 5, step (3b) also takes $O(mn)$ time. Step (3c) takes $O(mn)$ time, since each update takes constant time, and the shape is incrementally shifted to each of the mn grid points in the image. Since it is repeated for each of the $O(k)$ primitive shapes, step (3) takes total time of $O(kmn)$.

As mentioned in Lemma 5, the space requirements are $O(mn)$ per slope. Since we can discard the prefix sums computed in step (3b) 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.

4. CONCLUSIONS

We have presented an efficient algorithm for computing approximate convolutions for binary kernels that are modeled as a convex k-sided polygon. Our 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 showed that canonical digitizations can be updated efficiently through the use of prefix sums. There are a couple of interesting open problems that are suggested by this work. The first is whether these techniques can be generalized to convolutions involving kernels that are multi-valued or nonconvex simple polygons. In theory, such a kernel could be subdivided into convex parts. However, Lemma 4, which establishes the validity of the canonical digitization, does not generalize immediately to collections of convex polygons. Another question is if approximate convolutions are used to approximate morphological operations (such as dilation) then what can be said about the properties of the resulting shapes, as compared with their exact counterparts.

REFERENCES

1. J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4:25–30, 1965.
2. P. Burt. Fast filter transforms for image processing. *Computer Graphics and Image Processing*, 16:20–51, 1981.
3. E. Fiume. Coverage masks and convolution tables for fast area sampling. *Graphical Models and Image Processing*, 53:25–30, 1991.
4. J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, 1990.
5. R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley, Reading, MA, 1992.
6. D. Hearn and M. P. Baker. *Computer Graphics: C Version*. Prentice Hall, 1997.
7. J. Kim and Y. Kim. Efficient 2-D convolution algorithm with the single-data multiple kernel approach. *Graphical Models and Image Processing*, 57:175–182, 1995.
8. D. P. O'Leary. Some algorithms for approximating convolutions. *Computer Vision, Graphics, and Image Processing*, 41:333–345, 1988.
9. J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press Inc., New York, 1982.
10. J. Serra. *Image Analysis and Mathematical Morphology Vol. 2: Theoretical Advances*. Academic Press Inc., New York, 1988.
11. George Wolberg and Henry Massalin. Fast convolution with packed lookup tables. In Paul Heckbert, editor, *Graphics Gems IV*, pages 447–464. Academic Press, Boston, MA, 1994.