

Master's Paper [ROUGH DRAFT]:

2 Dimensional Min-Filters with Polygons

Paolo Codenotti¹

Advisors: László Babai¹, Pedro Felzenszwalb¹

Exam Committee: László Babai¹, Pedro Felzenszwalb¹, Janos Simon¹

April 20, 2007

¹Department of Computer Science, University of Chicago, Chicago, Illinois, USA. Email: {paoloc, laci, pff, simon}@cs.uchicago.edu

Abstract

Computing min-filters is an important operation in image processing. Min-filters are the basic building blocks for translation invariant morphological operators, which are used to perform operations such as noise suppression, image smoothing, contrast enhancement, and edge detection. Min-filters are a special case of min-convolutions, which are used for a variety of applications, including signal processing, and combinatorial optimization. The most widely used approach for computing min-filters decomposes the filtering element as a Minkowsky sum of smaller filters. However this approach can be expensive for large filtering elements[1]. There is an algorithm to perform the min-filter efficiently, but it only applies to axis parallel rectangles [2]. We present an efficient algorithm to compute the min-filter when the filtering element is a polygon for certain polygons. Because the polygon will be given as a non digital geometric object, we relax the conditions by allowing the algorithm to choose a valid digitization of the polygon for each placement. With this relaxation, we present an algorithm to compute the min-filter by arbitrary rectangles and isosceles right triangles in logarithmic time per pixel, in the size of the polygon. We also show how to compute the min-filter by arbitrary triangles in time $O\left(\frac{\log A}{\sin \gamma}\right)$ per pixel, where A is the area of the triangle, and γ is the size of the smallest angle in the triangle.

1 Introduction

Computing min-filters of images by polygons is an important tool for image processing. The min-filter of an image I by a shape S is defined to be the function that associates to every integer translate of S the minimum value I takes inside the translated shape. Min-filters are used to perform image processing operations such as noise suppression, image smoothing, contrast enhancement, and edge detection [3]. Min-filter is also an important special case of min-convolution, which are widely used in nonlinear signal processing, pattern analysis, computer vision, and combinatorial optimization [4].

1 dimensional min-filters by convex objects (intervals) can be computed in constant time per pixel. In fact monotone matrix search algorithms (see [5, 6]) can be used to compute a 1 dimensional min-convolution when one of the two functions is convex [4]. Min-filter by a convex shape is a special case min-convolution with a convex function. In two dimensions Gil and Werman present an algorithm to compute the min-filter by axis parallel rectangles in constant time per pixel [2]. Axis parallel rectangles can be seen as the simplest generalization of intervals to two dimensions. We present a simple and efficient algorithm to compute the min-filter of an image by rectangles that are not axis parallel. Then we generalize this to isosceles right triangles, and finally to arbitrary triangles. The algorithms are based on the idea of covering a shape with smaller copies of itself for a dynamic programming approach.

For triangles where the size of the angles is large enough (does not depend on the area of the triangle), the running time is logarithmic in the area of the triangle for each pixel in the image. The running time to compute the min-filter with non axis parallel rectangles and

isosceles right triangles is also logarithmic per pixel, in the area of the shape. Triangles are of special interest because any polygon with k sides (convex or non) can be triangulated by $k - 2$ triangles. In section 8, we explain how efficient algorithms for triangles could be used to design efficient algorithms for arbitrary polygons.

The rest of the paper is organized as follows. In section 2 we lay down some definitions. We give the formal statement of our results in section 3. We talk about related work in section 4. Then we present algorithms for some special cases: rectangles in section 5, and isosceles right triangles in section 6. In section 7 we show how to apply the ideas we used in these special cases to arbitrary triangles. Finally we present directions for future work in section 8.

Acknowledgements

I would like to thank Pedro Felzenszwalb, who introduced me to this problem, and with whom these results are joint work. I would like to thank my advisor Laszlo Babai for his continued support throughout my educational career.

2 Definitions

Let $\overline{\mathbb{R}} = \mathbb{R} \cup \{\infty\}$, $\mathbb{R}_+ = (0, \infty)$, and let $[n] = \{0, 1, \dots, n - 1\}$. For any shape $P \subseteq \overline{\mathbb{R}}^2$, we will denote by $Int(P)$ the set of integer points inside P , that is, $Int(P) = P \cap \mathbb{Z}^2$. Let us look at a polygon P , and let $P[i] = P \oplus i = \{j + i | j \in P\}$ for any $i = (i_1, i_2) \in \overline{\mathbb{Z}}^2$. We will

interchangeably use the notation $P[i]$ or $P[i_1, i_2]$, where (i_1, i_2) or i are understood to be in $\overline{\mathbb{R}}^2$. Polygons will be specified in canonical position, such that $P \subseteq \mathbb{R}_+^2$, and $P[i_1, i_2] \not\subseteq \mathbb{R}_+^2$ for integers $i_1 < 0$ or $i_2 < 0$. Let n_1 and n_2 be the largest integers satisfying:

$$P[i] \subseteq (0, n) \times (0, n) \forall 0 \leq i_1 \leq n_1, 0 \leq i_2 \leq n_2$$

The sets $P[i]$ for $i \in [n_1] \times [n_2]$ are the integer translates of P that fit inside $(0, n) \times (0, n)$.

Let P be a polygon, and let f be a function $f : [n]^2 \rightarrow \overline{\mathbb{R}}$. We define the min-filter $h : [n]^2 \rightarrow \overline{\mathbb{R}}$ of f by P , $h = f \otimes P$ as follows:

$$h(i) = \min_{j \in \text{Int}(P[i])} f(j).$$

Note that in fact h is only defined over $[n_1] \times [n_2]$, however it can be extended to $[n]^2$ assuming the value of f outside of $[n]^2$ is ∞ .

The problem we are considering is a special case of computing the min-convolution of two functions. The min-convolution h of two functions $f, g : [n]^d \rightarrow \overline{\mathbb{R}}$ is defined by:

$$h(i) = (f \otimes g)(i) = \min_j \{f(j) + g(i - j)\},$$

where values outside the range of f and g are defined to be infinite. Computing the min-filter of f by P is equivalent to computing the min-convolution of f with the function g_P defined by:

$$g_P(i) = \begin{cases} \infty & \text{if } i \notin \overline{P} \\ 0 & \text{if } i \in \overline{P} \end{cases} \quad (1)$$

Where \overline{P} denotes a flipped version of P (rotated by π).

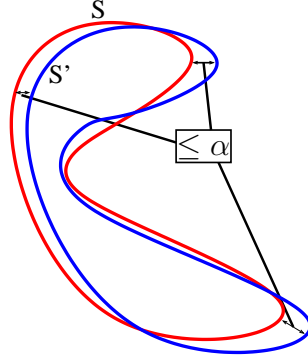


Figure 1: S' is an α -approximation to S .

Now we define two notions of approximation of shapes. An α -approximation of a shape S is a shape \tilde{S} such that every point that is well inside S (α away from the boundary), is in \tilde{S} , and every point well outside S (α away from the boundary) is not in \tilde{S} . Note that this is equivalent to saying that S and \tilde{S} have hausdorff distance at most α .

Definition 1 (see figure 1) Given a shape S , an α -approximation of S is a shape \tilde{S} such that the following conditions hold:

- $x \in S$, and $d(x, \partial S) \geq \alpha \Rightarrow x \in \tilde{S}$,
- $x \notin S$, and $d(x, \partial S) \geq \alpha \Rightarrow x \notin \tilde{S}$.

Where $d(x, y)$ denotes the distance between two objects, and ∂S denotes the boundary of the shape S . We do not impose any conditions on points that are α -close to ∂S .

Let us look at the integer points in our space as the centers of pixels, then we can make another definition. More precisely, given a point $i = (i_1, i_2)$, the pixel whose center is i is

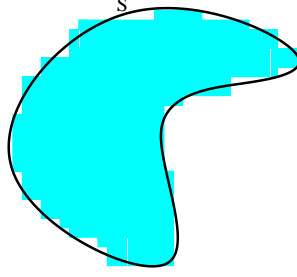


Figure 2: The colored pixels are a valid digitization of S .

the set of points

$$p[i] = \{(j_1, j_2) \mid -1/2 \leq i_1 - j_1 < 1/2 \text{ and } -1/2 \leq i_2 - j_2 < 1/2\}.$$

A valid digitization of a shape S is a set of pixels that includes all pixels completely contained in S and does not include any pixel completely outside S .

Definition 2 *Given a shape S , a valid digitization of S is a set of pixels \tilde{S} such that the following conditions hold:*

- *if a pixel p is completely contained in S , then $p \in \tilde{S}$,*
- *if a pixel p is completely outside S , then $p \notin \tilde{S}$.*

We do not impose any conditions on pixels on the boundary of S (see figure 2).

It is easy to see that every $1/2$ -approximation of S is a valid digitization, and every valid digitization is a $1/\sqrt{2}$ -approximation. Therefore we will use these two notions fairly interchangeably.

We now define two notions similar to $Int(P)$ for a polygon P . Let $Int^*(P)$ be the set of integer points inside a valid digitization of P , and $Int_\alpha(P)$ be the set of integer points inside

an α -approximation of P . Note that Int^* and Int_α are not actually functions, since there is more than one valid digitization or α -approximation of any given shape.

Definition 3 An α -approximated *min-filter* of a function f by a polygon P is defined by:

$$g(i) = \min_{j \in Int_\alpha(P[i])} f(j).$$

We denote g by $f \otimes_\alpha P$.

Note that each entry $(f \otimes_\alpha P)(i)$ will not be the min of the integer points inside $P[i]$, but the min of the integer points in some α -approximation to $P[i]$. There can be several α -approximations to $P[i]$, and the definition does not specify which one will be used. In particular, $Int_\alpha(P[i])$ may not be a translate of $Int_\alpha(P[j])$ for $i \neq j$.

Similarly, we can define a digitized min-filter.

Definition 4 A digitized *min-filter* of a function f by a polygon P is defined by:

$$g(i) = \min_{j \in Int^*(P[i])} f(j)$$

We denote g by $f \tilde{\otimes} P$.

This time each entry $(f \tilde{\otimes} P)(i)$ will be the min of integer points inside a valid digitization of $P[i]$.

When taking a min-filter of a function $f : [n]^2 \rightarrow \overline{\mathbb{R}}$, we can think of f as representing a digital image, where $f(i)$ corresponds to the color of the pixel with center i . Note that we need not restrict our domain to $[n]^2$, but we can define min-filters for a function $f : I \rightarrow \overline{\mathbb{R}}$ over an arbitrary domain I .

3 Results

The main results of this paper are efficient algorithms to compute the α -approximated min-filters of a function f by large rectangles and triangles. We state these results for functions $f : I \rightarrow \overline{\mathbb{R}}$ over some domain I . We will prove these results for $I = [n]^2$, however the same proofs apply to rectangles, and other shapes of the domain.

Theorem 1 *Given a rectangle R , and a function $f : I \rightarrow \overline{\mathbb{R}}$, we can compute $f \otimes_{\alpha} R$ in time $O(\alpha^2 \log A)$ per pixel, where A is the area of R . In particular, we can compute the digitized min-filter of f by R in time $O(\log A)$ per pixel.*

Theorem 2 *Given an isosceles right triangle T , and a function $f : I \rightarrow \overline{\mathbb{R}}$, we can compute $f \otimes_{\alpha} T$ in time $O(\alpha^2 \log A)$ per pixel, where A is the area of T . In particular, we can compute the digitized min filter of f by T in time $O(\log A)$ per pixel.*

Theorem 3 *Given a triangle T , and a function $f : I \rightarrow \overline{\mathbb{R}}$, we can compute $f \otimes_{\alpha} T$ in time $O(\frac{1}{\sin \gamma} \alpha^2 \log A)$ per pixel, where A is the area of T , and γ is the measure of the smallest angle in T . In particular, we can compute the digitized min-filter in time $O(\frac{1}{\sin \gamma} \log A)$ per pixel.*

The memory required is $O(n^2)$ for the first two algorithms, and $O(\frac{1}{\sin \gamma} n^2)$ for the algorithm for arbitrary triangles. The algorithms compute as intermediate results the min-filter by smaller similar shapes. It can be useful for some applications such as shape recognition to store the intermediate results. The memory required to store the intermediate results would be the same order as the running time of the algorithms. Moreover, although these algorithms compute approximate min-filters, they compute exact min-filters in some other

lattice. Therefore the approximate min-filter we compute has additional structure, all approximations from which we are taking the min are translates of each other.

4 Related work

The standard convolution in $[n]^d$ can be computed using Fast Fourier Transform methods in $O((2n)^d \log(n^d))$ time. In particular, in two dimensions the standard convolution can be computed in $O(N \log(N))$ time, where N is the number of grid points. The binary convolution is the notion equivalent to min-filter in the setting of standard convolutions. A binary convolution is a standard convolution where one of the two functions is $\{0, 1\}$ valued. Mount et al. show how to compute approximate binary convolutions by a polygon with k sides in time $O(kN)$, where N is the number of grid points [7]. Their notion of approximate binary convolution is equivalent to the notion of approximate min-filter we use. Their algorithm works in two steps. First they express the convolution by the polygon in terms of convolutions by primitive shapes: axis parallel rectangles and right triangles. Then they compute the convolution by the primitive shapes using prefix sums along strips in the image. Unfortunately both steps in their algorithm do not apply to the min-filter setting, because they rely on the fact that summation has an inverse (whereas min does not). On the other hand our techniques do not apply to the standard convolution setting because we use the fact that $\min\{A \cup B\} = \min\{\min A, \min B\}$, which is not true for sums. In fact the challenges in computing standard convolutions are quite different from those in the min-convolution setting.

Gil and Werman present an algorithm to compute the min-filter by axis parallel rectangles in constant time per pixel [2]. Their main observation is that the min-filter by an axis parallel rectangle can be expressed in terms of 1 dimensional min-filters with intervals, which can be computed in constant time per pixel. This decomposition however does not work for more complicated shapes, in fact it does not extend to non axis parallel rectangles. We present new algorithmic techniques that allow us to efficiently compute min-filters with large non axis parallel rectangles, and triangles.

The most widely used approach for the computation of min-filters is to decompose the filtering element (in our case this is the polygon P) as a Minkowsky sum of smaller filtering elements [3]. In fact, it is easy to see that $f \otimes (A \oplus B) = (f \otimes A) \oplus (f \otimes B)$, here \otimes is min-filter, and \oplus denotes the Minkowsky sum. A lot of work has focused on finding the optimal decomposition of filters into different sets of basic building blocks [1, 7]. This approach works well if the filtering element is small, but may not scale well, especially for filtering elements where the boundary is small compared to the area (such as convex shapes) [7]. For example, the number of small factors needed to decompose a large rectangle is order of the length of the diagonal of the rectangle. So an algorithm that used this decomposition would have running time order of the length of the diagonal per pixel. By contrast, our result for rectangles has running time logarithmic in the length of the diagonal per pixel. Moreover, this approach does not work for arbitrary polygonal shapes, because the intermediate results are always represented by a digital image.

More generally, min-filters can be related to erosions and dilations in mathematical mor-

phology. An erosion is what we called a min-filter, and a dilation is a max-filter. Note that any algorithm that computes min-filters can also compute the max-filter by changing the sign of the values of the image. Therefore we present efficient algorithms for erosions and dilations by large shapes. Erosions and dilations are of particular interest because they can be combined into more complex morphological operators. In 2 dimensions, any increasing translation invariant operator can be represented as a union of erosions [3, 8]. Morphological operators created from erosions and dilations are used to solve various problems in signal processing and computer vision. Very simple compositions of erosions and dilations with small sets are used for noise suppression, image smoothing, and contrast enhancement. Erosions and dilations with larger shapes are used to detect components and edges in images [3]. We suspect that efficient algorithms for min-filters with large shapes will have applications in shape recognition, and other operations in vision [9, 10, 11].

As we have seen in section 2, the min-filter is a special case of min-convolution. In one dimension, monotone matrix search algorithms (see [5, 6]) can be used to compute min-convolutions when one of the two functions is convex (an interval) [4, 12]. As noted in [4], the best known algorithms to compute a general min-convolution in arbitrary dimensions have running time $O(N^2/\log N)$, where N is the number of points in the space (in our case $N = n^2$). Babai and Felzenszwalb present an algorithm to compute the min-convolution when one of the two functions takes on a constant number of distinct values in $O(N^{\frac{3}{2}}(\log N)^{\frac{1}{2}})$. As shown in the definitions section, min-filters are min-convolutions where one of the two functions takes values from $\{0, \infty\}$, therefore this algorithm applies to min-filters. There

is significant interest in the efficient computation of min-convolutions when one of the two functions belongs to a special class of functions, and in particular when one of the two functions is convex. Felzenszwalb and Huttenlocher presented an algorithm to compute the min-convolution when one of the two functions is a parabola, and for certain other classes of functions in $O(dN)$ time, where N is the number of grid points, and d is the dimension of the shape [9].

5 Rectangles

5.1 Axis parallel rectangles

First let us consider the case of axis parallel rectangles. Gil and Werman show an algorithm to compute the min-filter with an axis parallel rectangle in time $O(n^2)$ [2]. We present an algorithm that runs in time $O(n^2 \log A)$, where A is the area of the rectangle. Although this result is not as good as the one presented in [2], we will use the main idea of this algorithm to compute the min-filter with non axis parallel rectangles and triangles, so it is instructive to look at this special case. Moreover the algorithm we present computes the min-filter by smaller similar rectangles as well, which might be useful for some applications in shape recognition [10].

Let R be a rectangle of side lengths L and M , with the lower left vertex at an integer point (the origin, since polygons are given in canonical position). We can reduce the problem of finding the min-filter by R to finding the min-filter by smaller rectangles. The rectangle R

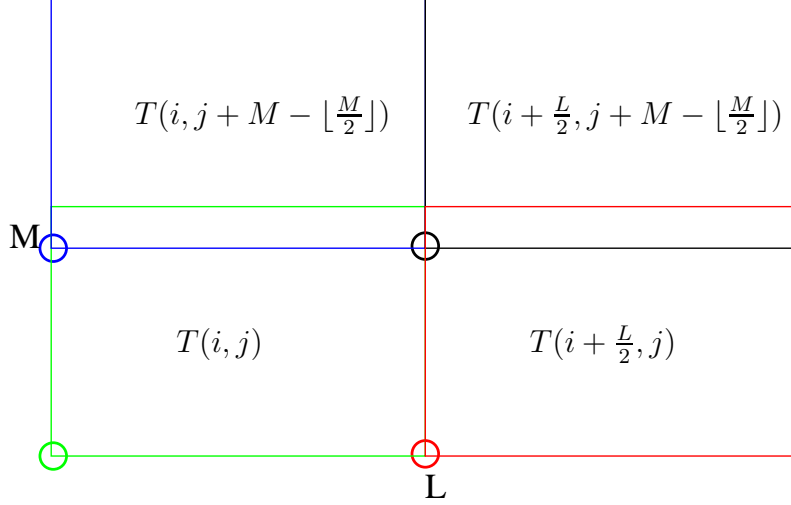


Figure 3: A rectangle of side lengths L and M covered by integer shifts of a smaller rectangle T . In the illustration, L is an even integer, and M is not. The circles indicate the lower left corners of the smaller rectangles.

is covered by four congruent rectangles of side lengths $L/2$ and $M/2$. These rectangles may not be at integer coordinates, but we can also cover R by four slightly larger rectangles each with the lower left vertex at an integer point. Let T be the rectangle in canonical position with side lengths $L - \lfloor \frac{L}{2} \rfloor$ and $M - \lfloor \frac{M}{2} \rfloor$. Then for any $(i, j) \in [n]^2$, the set $R[i, j]$ can be covered by the four sets $T[i, j]$, $T[i + \lfloor \frac{L}{2} \rfloor, j]$, $T[i, j + \lfloor \frac{M}{2} \rfloor]$, and $T[i + \lfloor \frac{L}{2} \rfloor, j + \lfloor \frac{M}{2} \rfloor]$ (see figure 3). Note that we allow the covering elements to overlap, since this does not have an effect for computing the min. Let $h = f \otimes T$, then we can express the min-filter of f by R as:

$$(f \otimes R)(i, j) = \min\{h(i, j), h(i + \lfloor \frac{L}{2} \rfloor, j), h(i, j + \lfloor \frac{M}{2} \rfloor), h(i + \lfloor \frac{L}{2} \rfloor, j + \lfloor \frac{M}{2} \rfloor)\}. \quad (2)$$

We can use this recursive formula for a dynamic programming algorithm. To compute

the min-filter by a rectangle R of side lengths L and M , we first compute the sequences $L = L_0, L_1, \dots, L_k$, and $M = M_0, M_1, \dots, M_k$, where $L_{q+1} = L_q - \lfloor \frac{L_q}{2} \rfloor$, $M_{q+1} = M_q - \lfloor \frac{M_q}{2} \rfloor$, and $M_k, L_k < c$, for some absolute constant c ($c = 2$ will do for the analysis, although it might be more efficient to have larger c in practice). Let R_q be the rectangle in canonical position with side lengths L_q and M_q . We first compute the min-filter by R_k by exhaustive search, then we compute the min-filter by R_q by reverse induction on q . Note that we are able to compute $f \otimes R_k$ by lemma 1, since R_k is bounded and we can test if a point is inside $R_k[i, j]$ in constant time.

Lemma 1 *Given a bounded shape S (contained in a circle of constant radius R), and given that we can determine whether a given point is in S in constant time, if we are given any point in S , then we can list all integer points in S in constant time.*

Proof: see appendix. ■

Let $g_q = f \otimes R_q$, then by equation (2) we have:

$$g_{q-1}(i, j) = \min\{g_q(i, j), g_q(i + \lfloor \frac{L}{2} \rfloor, j), g_q(i, j + \lfloor \frac{M}{2} \rfloor), g_q(i + \lfloor \frac{L}{2} \rfloor, j + \lfloor \frac{M}{2} \rfloor)\}. \quad (3)$$

More precisely, given a rectangle R of side lengths L and M , a function f , and an integer n , the algorithm for computing the min-filter is as follows.

Min-filter(R, L, M, n, f)

1. $L[0]:=L$; $M[0]:=M$; $q:=0$;
2. while $M[q]>c$ OR $L[q]>c$ do //compute the sequence of side lengths
3. $q:=q+1$;
4. $M[q]:=M[q-1] - \text{floor}(M[q-1]/2)$;
5. $L[q]:=L[q-1] - \text{floor}(L[q-1]/2)$;
6. end while
7. $k:=q$;
8. Let $R[q]$ be the rectangle with side lengths $M[q]$ and $L[q]$
9. Compute the min-filter by $R[k]$ by exhaustive search
10. for $q = k-1$ to 0 do
11. for $i = 1$ to n do
12. for $j = 1$ to n do
13. Compute $g[q](i, j)$ using the formula in equation (3)
14. return $g[0]$.

The initialization (lines 1-8) takes time $O(k)$. The first step in the induction (line 9) takes time $O(n^2)$. Each iteration of the for loops (line 13) takes time $O(1)$. The total running time of the loops (lines 10-13) is $O(kn^2)$, so the overall running time is $O(kn^2)$. Note that $k = O(\log(\max\{L, M\})) = O(\log A)$, where A is the area of R . So we can express the running time as $O(n^2 \log A)$. Note that in the dynamic programming we only need to remember the previous step, therefore we can over-write g_q once we computed g_{q-1} . So the memory required by the algorithm is $O(n^2)$. However for certain applications (such as shape

recognition), it might be useful to store the result of the min-filter by smaller similar shapes.

Then the memory required would be $O(n^2 \log A)$.

5.2 Non axis parallel rectangles

Now let us show how to compute the min-filter by a non axis parallel rectangle R with the lower left vertex not necessarily at an integer point. In this case we obtain the following result, for the relaxed definition of min-filter.

Theorem 4 *Given a rectangle R , and a function $f : [n]^2 \rightarrow \overline{\mathbb{R}}$, we can compute $f \otimes_\alpha R$ in time $O((\alpha n)^2 \log A)$, where A is the area of R . In particular, we can compute the digitized min-filter of f by R in time $O(n^2 \log A)$.*

Proof: The algorithm works in two stages. First it computes the exact min-filter in a different coordinate system, using the algorithm presented in the previous section. Then it uses the result of the first step to compute $f \otimes_\alpha R$.

First we show the algorithm to compute a $\frac{1}{\sqrt{2}}$ -approximate min-filter. Consider a new coordinate system, with origin in the lower left corner of R , and axes in the direction of the sides of R (see figure 4). We define L to be the set of integer points in the new coordinate system that are inside $(-1, n) \times (-1, n)$ in the original coordinate system. We chose this set so that every point in $[n]^2$ is close to a point in L . Note that $|L| = O(n^2)$. Recall that we defined $R[i]$ to be R shifted by $i \in \mathbb{Z}^2$. Similarly, we denote the shift of R by a point $i \in L$ by $R\{i\}$. Let $f \otimes' R$ be the min-filter of f by R over L , more precisely:

$$h(i) = (f \otimes' R)(i) = \min_{j \in \text{Int}(R\{i\})} f(j) \quad (4)$$

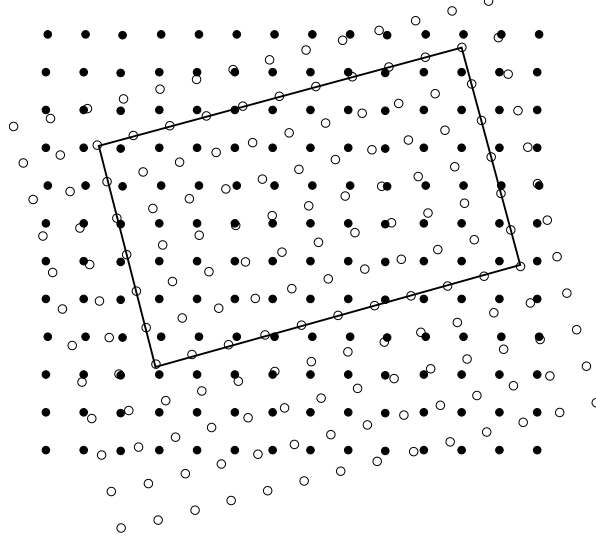


Figure 4: The integer points in the new coordinate system for the rectangle are represented by empty dots, \mathbb{Z}^2 is represented by the filled in dots.

Note that h is defined over points in L , but Int still refers to points in \mathbb{Z}^2 . Note further that we can define h over all points in L , assuming that the value of f over points not in $[n]^2$ is ∞ .

Claim 1 $\forall i \in [n]^2 \quad \exists i' \in L$ s.t. $R\{i'\}$ is a $1/\sqrt{2}$ approximation to $R[i]$.

Proof: Every point in $(-1, n) \times (-1, n)$ is at distance at most $1/\sqrt{2}$ from a point in L . In particular, this holds for points in $[n]^2$. Given a point $i \in [n]^2$, we can find the coordinates of i in the new coordinate system, and round them to the nearest integer. Call this new point i' , and note that $i' \in L$. Then $R\{i'\}$ is a $1/\sqrt{2}$ -approximation of $R[i]$, since $R\{i'\}$ is a congruent copy of $R[i]$ translated by the vector $v = i' - i$ (see figure 5). But the norm of v is the distance between i' and i , so it is less than $1/\sqrt{2}$.

■

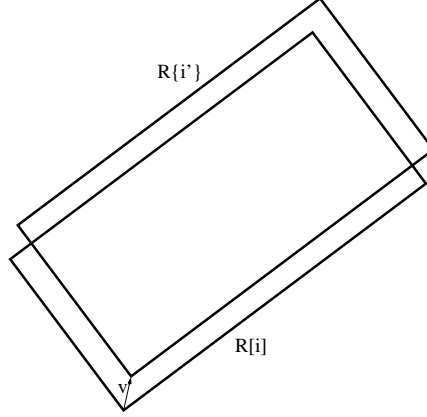


Figure 5: $R\{i'\}$ is a $\|v\|$ -approximation to $R[i]$.

After computing $f \otimes' R$, we compute $f \otimes_{\frac{1}{\sqrt{2}}} R$ in the original coordinate system as follows:

$$(f \otimes_{1/\sqrt{2}} R)(i) = f \otimes' R(i'). \quad (5)$$

Where i' is the closest point to i in L . By definition,

$$(f \otimes' R)(i') = \min_{x \in R\{i'\}} f(x),$$

and Claim 1 proves that $R\{i'\}$ is a $1/\sqrt{2}$ -approximation of $R[i]$. Therefore equation (5) correctly computes $f \otimes_{1/\sqrt{2}} R$. This last operation takes constant time for each $i \in [n]^2$. So the overall running time is $O(n^2 \log A + n^2) = O(n^2 \log A)$. Note that we were able to compute the min-filter with constant size non axis parallel rectangles by exhaustive search by lemma 1 (this is necessary for the algorithm to efficiently compute $f \otimes' R$).

Now we show how to compute an α -approximate min-filter by a rectangle for arbitrary α . We apply the same algorithm we used for $\alpha = 1/\sqrt{2}$, but we scale the new coordinate system by a factor of $\alpha\sqrt{2}$. That is, points in L will be at distance $\alpha\sqrt{2}$ from each other (see

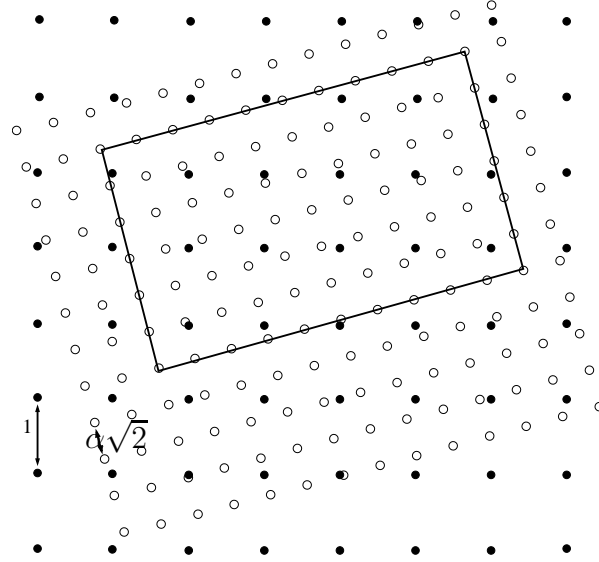


Figure 6: In the new coordinate system (empty points) integer points are at distance $\alpha\sqrt{2}$ from each other.

figure 6). Now every point in $[n]^2$ is at distance at most α from a point in L . The analysis of the rest of the algorithm goes through as before, with an approximation of α instead of $1/\sqrt{2}$. The first step of the algorithm (computing $f \otimes' R$) will take time $O((\alpha n)^2 \log A)$, since the L has size $O((\alpha n)^2)$. The second step in the algorithm will take time $O(n^2)$. So, as claimed, the total running time is $O((\alpha n)^2 \log A)$. ■

Note that we could not apply the Gil-Werman algorithm directly in the new coordinate system, because the function f takes values at integer points in the old coordinate system. The Gil-Werman decomposition into 1 dimensional min-filters does not work in this setting. On the other hand, the algorithm we presented in section 5.1 works no matter on what points f is defined, as long as it is easy (constant time) to determine what points are inside a given small (constant size) rectangle.

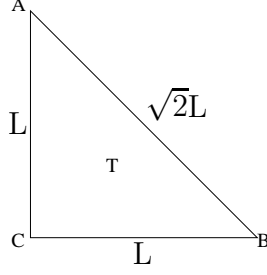


Figure 7: An isosceles right triangle T with side length L and right angle at C .

6 Isosceles right triangles

Theorem 5 *Given an isosceles right triangle T , and a function $f : [n]^2 \rightarrow \overline{\mathbb{R}}$, we can compute $f \otimes_{\alpha} T$ in time $O((\alpha n)^2 \log A)$, where A is the area of T . In particular, we can compute the digitized min filter of f by T in time $O(n^2 \log A)$.*

We will first prove this result for T an axis parallel isosceles right triangle. Then we will extend the result to the non axis parallel case using the same technique as in the previous section. Let us label the vertices of T by A , B , and C , with $\widehat{C} = \pi/2$, and $\overline{AC} = \overline{BC} = L$ (see figure 7).

The algorithm is based on a similar divide and conquer idea to the one for rectangles. By corollary 1, the triangle T can be covered by three isosceles right triangles of side length $2/3L$, each sharing a vertex with T (see figure 8). Let S be the triangle similar to T of side length $L - \lfloor \frac{L}{3} \rfloor$. Then the triangle $T[i, j]$ will be covered by $S[i, j]$, $S[i + \lfloor \frac{L}{3} \rfloor, j]$, and $S[i, j + \lfloor \frac{L}{3} \rfloor]$.

Corollary 1 *Triangles T_A , T_B , and T_C each similar to T scaled by a factor of $\frac{2}{3}$, and each sharing the corresponding vertex of T , cover T . That is, every point x in T is in at least one*

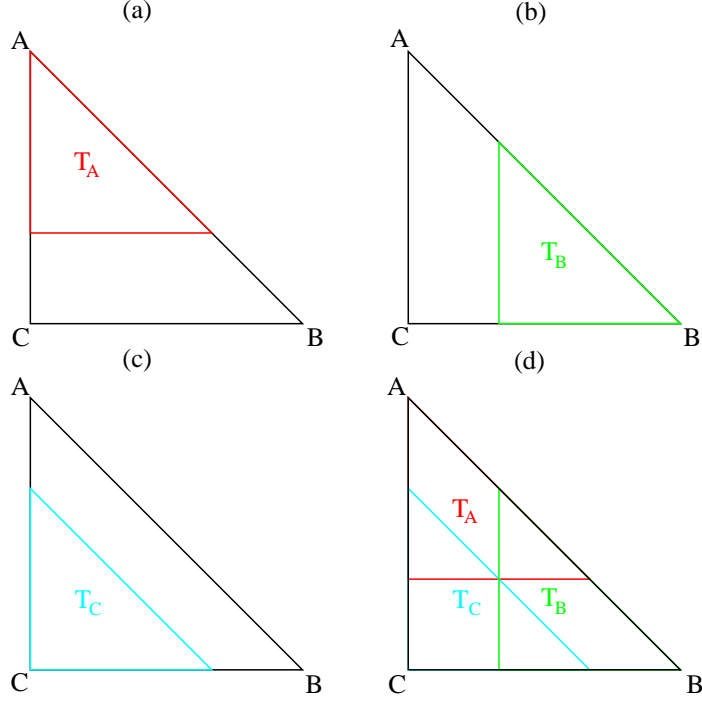


Figure 8: Triangles T_A , T_B and T_C cover triangle T , and are similar to T but scaled by a factor of $2/3$.

of the smaller triangles (see figure 8).

Proof: see appendix. ■

As we did in the case of rectangles, we use this intuition to design an efficient dynamic programming algorithm. First we compute the sequence of lengths $L = L_0, L_2, \dots, L_k$, where $L_{q+1} = L - \lfloor \frac{L_q}{3} \rfloor$, and $L_k \leq c$ for some absolute constant c . The algorithm will compute the min-filter by the right isosceles triangle of side length L_q . Let T_q be the isosceles right triangle of side length L_q , and let $g_q = f \otimes T_q$. Then we have:

$$g_{q-1}(i, j) = \min\{g_q(i, j), g_q(i + \lfloor \frac{L}{3} \rfloor, j), g_q(i, j + \lfloor \frac{L}{3} \rfloor)\} \quad (6)$$

Similarly to the algorithm presented in section 5, this algorithm will compute g_k by exhaustive search (this can be done efficiently by lemma 1), and then iteratively compute g_q starting from $q = k - 1$ to $q = 0$, using equation 6. The analysis of the algorithm is analogous to the analysis for axis parallel rectangles, and gives us the desired running time of $O(n^2 \log A)$, where A is the area of T .

For non axis parallel isosceles right triangles, we can apply the same trick we used in section 5.2: we compute the exact min-filter in a different coordinate system to get an algorithm for approximate min-filter in the original basis. Specifically, given a (not necessarily axis parallel) isosceles right triangle T , and a function f , we can compute the α -approximate min-filter of f by T in time $O((\alpha n)^2 \log A)$. Therefore we can compute the digitized min-filter of f by T in time $O(n^2 \log A)$.

7 General triangles

By corollary 1, any triangle T can be covered by smaller triangles T_A , T_B , and T_C , similar to T , scaled by a factor of $2/3$, each sharing the corresponding vertex with T . This would suggest that we can apply an algorithm similar to the one in section 6 to arbitrary triangles. However in order to apply this algorithm we need to be able to precompute the min inside smaller triangles. The algorithm relies on the fact that the number of triangles we need to consider at each level does not grow. In other words, we need the same small triangle to be

used in the decomposition of several larger triangles, otherwise the number of small triangles would grow exponentially in the number of levels, and the running time could be as bad as exhaustive search. The algorithm in the last section gets around this problem by carefully picking smaller triangles in the lattice. In this section we show an algorithm that does a similar thing for arbitrary triangles. This algorithm will be more efficient for triangles with no small angles. We use the algorithm described in section 6 on a lattice different from the integer grid.

Theorem 6 *Given a triangle T , and a function $f : [n]^2 \rightarrow \overline{\mathbb{R}}$, we can compute the α -approximated min-filter of f by T in time $O(\frac{1}{\sin \gamma}(\alpha n)^2 \log A)$, where A is the area of T , and γ is the smallest angle in T . In particular, we can compute the digitized min-filter of f by T in time $O(\frac{1}{\sin \gamma} n^2 \log A)$.*

Proof: We will show the result for $\alpha = 1/\sqrt{2}$. The scaling trick from section 5.2 will give us arbitrary α for an additional factor of α^2 in the running time. The algorithm will proceed in three steps. First it will create a lattice L_T based on the triangle T . Secondly, it will apply an algorithm like the one described in section 6 to compute the min-filter of f by T over the lattice L_T . Then we use this min-filter over L_T to compute an approximate min-filter in the original integer grid.

Let \widehat{C} be the smallest angle in T , and let a and b be the sides incident to C . Assume w.l.o.g. that a is the longest side. We construct the lattice with basis vectors v_a and v_b , along the directions of sides a and b respectively. The lengths of the basis vectors are: $\|v_a\| = 1$, and $\|v_b\| = \frac{b}{a}$. The lattice generated by the basis vectors v_a and v_b starting from point v is

defined to be the of points of the form $v + \alpha v_a + \beta v_b$, for α and β integers. We start our lattice from the vertex C in the triangle. More precisely, the lattice we create (which we denote by L_T) is defined by:

$$L_T = \{C + \alpha v_a + \beta v_b \in \mathbb{R}^2 \mid \alpha \in \mathbb{Z}, \beta \in \mathbb{Z}\}. \quad (7)$$

Definition 5 *The min-filter of f by T on a lattice L is defined as:*

$$(f \otimes_L T)(v) = \min_{j \in \text{Int}(T[v])} f(j) \quad (8)$$

For $v \in L$, and where $\text{Int}(T[v])$ is the set of points in \mathbb{Z}^2 inside the shift of T by v .

The algorithm computes the exact min-filter of f by T for points in $L_T \cap (-1, n) \times (-1, n)$. This will be done by covering T with three triangles T_A , T_B , and T_C , each similar to T , and of size at least $2/3$ the size of T .

The algorithm for computing $f \otimes_{L_T} T$ works via the usual dynamic programming idea. First we compute the sequence of values $a = a_0, \dots, a_k$, and $b = b_0, \dots, b_k$, where $a_{q+1} = a_q - \lfloor \frac{a_q}{3} \rfloor$, and $b_q = \frac{b}{a} a_q$, and k is the smallest integer such that $a_k \leq c$, for some absolute constant c . Let T_q be the triangle similar to T with longest side of length a_q , and let $g_q = f \otimes_{L_T} T_q$. The algorithm computes g_k by exhaustive search (this can be done efficiently by lemma 1). Then we inductively compute g_{q-1} given g_q , according to the following rule:

$$g_{q-1}(v) = \min \left\{ g_q(v), g_q \left(v + \left\lfloor \frac{a_{q-1}}{3} \right\rfloor v_a \right), g_q \left(v + \left\lfloor \frac{a_{q-1}}{3} \right\rfloor v_b \right) \right\} \quad (9)$$

In order for equation 9 to even make sense, we need to show the points $v + \lfloor \frac{a_{q-1}}{3} \rfloor v_a$, and $v + \lfloor \frac{a_{q-1}}{3} \rfloor v_b$ are in the lattice. But v is in the lattice, and $\lfloor \frac{a_{q-1}}{3} \rfloor \in \mathbb{Z}$. In order to prove the correctness of the algorithm, we now need to show that $T_q^C = T_q[v]$, $T_q^B = T_q[v + \lfloor \frac{a_{q-1}}{3} \rfloor v_a]$, and $T_q^A = T_q[v + \lfloor \frac{a_{q-1}}{3} \rfloor v_b]$ cover $T_{q-1}[v]$. This is the case because T_q is a triangle similar to T_{q-1} , scaled by a factor of $\frac{a_{q-1}}{a_q} \geq \frac{2}{3}$. Moreover, T_q^A shares vertex A with T_{q-1} , T_q^B shares vertex B with T_{q-1} , and T_q^C shares vertex C with T_{q-1} . So by corollary 1, $T_{q-1} = T_q^A \cup T_q^B \cup T_q^C$.

We have shown that the dynamic programming algorithm will correctly compute the min-filter over the lattice L_T . The running time will be $O(\log A)$ times the number of lattice points inside $(-1, n) \times (-1, n)$, since there are $k = O(\log A)$ levels in the dynamic programming, and at each level the algorithm takes constant time for each lattice point.

Claim 2 *The number of lattice points inside $(-1, n) \times (-1, n)$ is $O(\frac{1}{\sin \gamma} n^2)$.*

Proof: We are looking for the size of the set

$$S = \{w \in \mathbb{R}^2 \mid w = v + \alpha v_a + \beta v_b, \text{ for some } \alpha, \beta \in \mathbb{Z} \text{ and } w \in (-1, n)^2\}. \quad (10)$$

With every point $v \in L_T$, associate the parallelogram P_v with vertices v , $v + v_a$, $v + v_b$, and $v + v_a + v_b$. For large enough n , most $v \in S$ will be such that $P_v \subseteq (-1, n)^2$. Let $U = \{v \in L_T \mid P_v \subseteq (-1, n)^2\}$, then $|S| = O(|U|)$. Each P_v has area $\sin \gamma$, since it has side length 1, and height $\sin \gamma$. Therefore $\sin \gamma |U| \leq (n+1)^2$, since all elements of U are disjoint and inside $(-1, n)^2$. Therefore $|S| = O(\frac{1}{\sin \gamma} n^2)$. ■

Therefore computing $f \otimes_{L_T} T$ takes time $O(\frac{1}{\sin \gamma} n^2 \log A)$. Once computed $f \otimes_{L_T} T$, we compute $f \otimes_{1/\sqrt{2}} T$, by:

$$(f \otimes_{1/\sqrt{2}} T)(i) = (f \otimes_L T)(i')$$

Where i' is the point in the lattice closest to i . Because both basis vectors of the lattice have norm less than 1, the distance between i and i' is at most $1/\sqrt{2}$, so this gives us an approximate min-filter. To extend this for arbitrary α , we scale the lattice by a factor of $\sqrt{2}\alpha$ in both directions, like in section 5.2.

■

8 Conclusions and Future Work

We have shown algorithms to efficiently compute approximate min-filters of images by not necessarily axis parallel rectangles and triangles. These algorithms run in time $O(n^2 \log A)$ in the case of rectangles and isosceles right triangles, where A is the area of these shapes, and $O(\frac{1}{\sin \gamma} n^2 \log A)$ for a general triangle with smallest angle γ and area A . Previously efficient algorithms for large 2 dimensional filtering elements were known only for axis parallel rectangles [2]. This algorithm used a decomposition into 1 dimensional min-filters that cannot be applied to non axis-parallel objects. We present a new algorithmic technique that gives efficient algorithms for shapes that are not axis parallel as well. Because in our setting the filtering elements are given as polygons, we relaxed the definition of min-filter to allow the algorithm to choose a digitization for each placement.

There are several directions of future work. First of all, is there an algorithm that will take time $O(n^2 \log A)$ to compute the min-filter by a triangle, independently of the shape

of the triangle? In fact even algorithms with running times that are polylogarithmic in the area of the triangle would be interesting. Even if there are such algorithms, we do not know of any lower bound except for the trivial $\Omega(n^2)$. Therefore it remains open to either find an algorithm to match the lower bound or prove a better lower bound. Lastly, we can try to extend our algorithms to any polygon by triangulation, however we need to be careful about the details because of the digitization. The output we obtain is not an exact min-filter but an approximate min-filter, so when computing the min for some shift, we actually obtain the min for some shape close to the one we wanted. This might create strips in the polygon that are not covered between digitizations of different triangles. Moreover, these strips could be different for different shifts of the polygon. We can get around this problem by covering the polygon with triangles, and “stitching” the triangles together with long thin rectangles. Working out the details of this argument is another direction for future research.

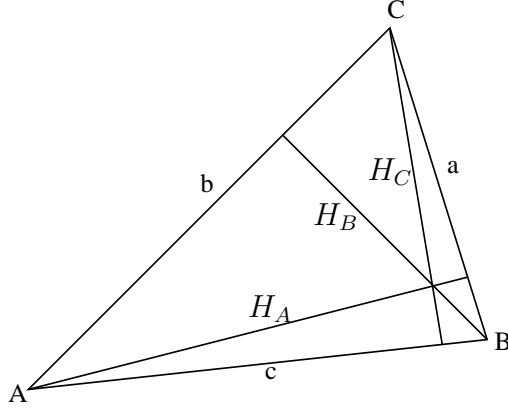


Figure 9: A triangle with vertices A , B and C , heights H_A , H_B , and H_C , and sides a , b , and c

Appendix

Lemma 1 *Given a bounded shape S (contained in a circle of constant radius R), and given that we can determine whether a given point is in S in constant time, if we are given any point in S , then we can list all integer points in S in constant time.*

Proof: We are given some point $x \in S$. Compute x' , the integer point nearest to x by rounding. Look at the set

$$T = \{x' + v \mid v \in \mathbb{Z}^2, \|v\| \leq R + 1\}.$$

Every integer point contained in S must be in T , moreover, $|T| = O(R^2) = O(1)$. Therefore we can compute $S \cap T$ in constant time by querying every point in T for membership in S . ■

Lemma 2 *(see figure 9 for the notation)*

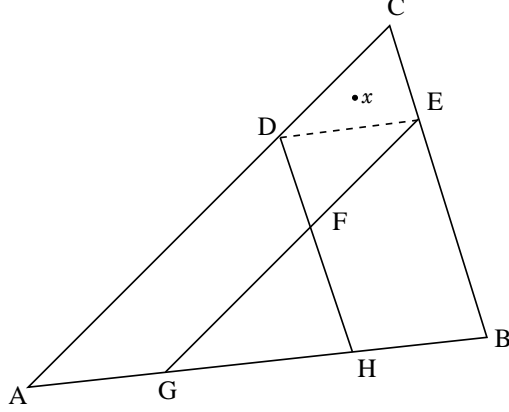


Figure 10: The construction in the proof of lemma 2

Let $T = ABC$ be a triangle, and let x be any point in T . Let H_A , H_B , and H_C be the lengths of the heights of T with respect to vertices A , B , and C respectively. Let a , b , and c be the sides opposite to vertices A , B , and C respectively, and let $h_a(x)$, $h_b(x)$, and $h_c(x)$ be the distances from x to sides a , b , and c respectively.

$$\text{if } h_a(x) \leq \frac{H_A}{3} \text{ and } h_b(x) \leq \frac{H_B}{3}, \text{ then } h_c(x) \geq \frac{H_C}{3}. \quad (11)$$

Proof: Let $D \in \overline{AC}$ at distance $\frac{b}{3}$ from C , $E \in \overline{BC}$ at distance $\frac{a}{3}$ from C , G and $H \in \overline{AB}$ at distance $\frac{c}{3}$ and $\frac{2c}{3}$ respectively from A . Let F be the intersection of EG and DH (see figure 10). Notice that DE is parallel to AB , GE is parallel to AC , and DH is parallel to BC . Therefore $\triangle (DCE)$ and $\triangle (DFE)$ are congruent, since the two triangles are similar, and they share a side. Moreover, both these triangles are similar to $\triangle (ACB)$, scaled by a factor of $\frac{1}{3}$. So the height of $\triangle (DCE)$ with respect to C is $\frac{H_C}{3}$. Therefore the height of $\triangle (DEF)$ with respect to F is $\frac{H_C}{3}$. Therefore,

$$\forall x \in CEFD, h_c(x) \geq \frac{H_C}{3} \quad (12)$$

On the other hand,

$$\left. \begin{array}{l} h_a(x) \leq \frac{H_A}{3} \Rightarrow x \in CBHD, \text{ and} \\ h_b(x) \leq \frac{H_B}{3} \Rightarrow x \in CEGA \end{array} \right\} \Rightarrow x \in CEFD \Rightarrow h_c(x) \geq \frac{H_C}{3}. \quad (13)$$

Where the last implication follows from equation 12.

■

Corollary 1 *Triangles T_A , T_B , and T_C each similar to T scaled by a factor of $\frac{2}{3}$, and each sharing the corresponding vertex of T , cover T . That is, every point x in T is in at least one of the smaller triangles(see figure 8).*

Proof: Using the notation from the lemma above, for any $D \in A, B, C$, the triangle T_D consists of all the points $x \in T$ such that $h_d(x) \geq \frac{H_D}{3}$. Therefore we are done, by lemma 2.

■

References

- [1] H. Park and R. Chin. Decomposition of Arbitrarily Shaped Morphological Structuring Elements. *IEEE Transactions on pattern analysis and machine intelligence*, 17(1), January 1995.
- [2] J. Gil and M. Werman. Computing 2-d min, median, and max filters. *IEEE Transactions on pattern analysis and machine intelligence*, 15:504–507, 1993.
- [3] P. Maragos. *Morphological Filtering for Image Enhancement and Feature Detection*. Elsevier Academic Press, 2005.
- [4] L. Babai and P. Felzenszwalb. Min-convolution with a function with small range. Technical report, University of Chicago, 2006.
- [5] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric Applications of a Matrix-Searching Algorithm. *Algorithmica*, 2:195 – 208, 1987.
- [6] A. AAgrawal and J. Park. Notes on Searching in Multidimensional Monotone Arrays. In *Proc. 29th Annual Symposium on Foundations of computer Science*, pages 497–512, 1988.
- [7] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. Approximating large convolutions in digital images. In *Proc. SPIE Vol. 3454, Vision Geometry VII*, pages 216–227, October 1998.

- [8] P. Maragos and R. Schafer. Morphological Systems for Multidimensional Signal Processing. In *Proc. IEEE*, volume 78, pages 690–710, 1990.
- [9] P. Felzenszwalb and D. Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science Technical Report, 2004.
- [10] P. Felzenszwalb and D. Huttenlocher. Efficient belief propagation for early vision. In *IEEE Computer Vision and Pattern Recognition*, pages 261–268, 2004.
- [11] P. Felzenszwalb D. Crandall and D. Huttenlocher. Spatial priors for part-based recognition using statistical models. In *IEEE Computer Vision and Pattern Recognition*, pages 10–17, 2005.
- [12] E. Eppstein. Efficient algorithms for sequence analysis with concave and convex gap costs. PhD thesis, Columbia u University, 1989.