



Fast Grayscale Morphology for Circular Window

Yuji Moroto  and Nobuyuki Umetani 

The University of Tokyo



Figure 1: Example of a bokeh filter using dilation, a type of morphological operation. By filtering at the maximum value within the circular window, highlights are enhanced, creating a beautiful expression that differs from actual lens blur. As a blur effect, circular windows are typically preferred over rectangle windows due to their isotropic feature.

Abstract

Morphological operations are among the most popular classic image filters. The filter assumes the maximum or minimum value within a window and is often used for light object thickening and thinning operations, which are important components of various workflows, such as object recognition and stylization. Circular windows are preferred over rectangular windows for obtaining isotropic filter results. However, the existing efficient algorithms focus on rectangular or binary input images. Efficient morphological operations with circular windows for grayscale images remain challenging. In this study, we present a fast grayscale morphology heuristic computation algorithm that decomposes circular windows using the convex hull of circles. We significantly accelerate traditional methods based on Minkowski addition by introducing new decomposition rules specialized for circular windows. As our morphological operation using a convex hull can be computed independently for each pixel, the algorithm is efficient for modern multithreaded hardware.

CCS Concepts

• **Computing methodologies** → **Image processing**; **Parallel algorithms**; **Computer vision**;

1. Introduction

Since the inception of digital image processing, morphological operations have become an integral part of various image and video processing operations. The filter calculates the maximum or minimum values inside the window around each pixel, and combinations of these filters can produce various effects, such as *opening*

and *closing*, which are popular noise reduction tools. Typical video editing software, such as Adobe After Effects, provides morphological effects to extract smooth matte in chromakeys or to calculate the outline of illustrations. Max pooling operations in deep neural networks are morphological operations.

Although morphological operations are often applied to binary

images for segmentation tasks, they are frequently used for grey scale or RGB images. For example, as shown in Figure 1, dilating each channel in a color image results in a blurred image. Although the morphological operation does not correspond to an actual optical lens model, it is useful for artistic bokeh expressions. In image-filtering operations, particularly blurring, circular windows are preferred to rectangular windows because the result is more isotropic. However, performing morphological operations using circular windows for grayscale images is challenging. Efficient algorithms for larger window sizes are in high demand as the resolution of images and videos continues to increase.

Although grayscale morphological operations with *rectangular windows* can be computed efficiently in constant time, regardless of the window size [VH92, GW93, DD11], few studies have addressed the problem of efficient grayscale morphological operations with *circular windows*. Methods targeting arbitrary window shapes require approximately $O(R)$ times per pixel, where R is the window radius [DT96, UW08]. Furthermore, there is a method that efficiently computes large arbitrarily shaped windows using Minkowski addition, decomposing a large window into several smaller windows, and repeating the morphological operations [VV97, HBF00]. However, circular windows are difficult to decompose and must be approximated as regular polygons (e.g., hexadecagons).

This paper presents a method that performs grayscale morphology operations on circular windows many times faster than the existing baseline approaches [UW08, VV97, VKM07]. Our method specializes in circular windows, and uses Minkowski addition to achieve acceleration. Specifically, we present a decomposition approach for a circular window that considers the convex hulls of elements. Our method can accurately reproduce approximately 87% of the circular windows with different radii without any error. The remaining windows have considerably small approximation errors, making our approach sufficiently practical for circular morphologies. Our algorithm can be computed independently on a pixel-by-pixel basis, making it compatible with new hardware with multi-threaded or vector arithmetic units.

2. Related Work

Morphological operations simply replace the value of a pixel with the minimum or maximum value in a window around the pixel. Morphological operations were among the earliest filters developed in image processing, and Kirsch et al. [KCRU57] discussed several 3×3 morphological operators. The use of the maximum value in the window is called *dilation*, and the use of the minimum value is called *erosion*.

Haralick et al. [HSZ87] defined the composition of the morphological operations for both binary and grayscale images. Dilation after erosion is called *opening*, and it removes noise without changing the thickness of the object. Filling the voids in an object by erosion after dilation is called *closing*. The process of detecting contours by differentiating dilated and eroded images is called the *gradient*. Morphological operations are integral parts of several classical image-processing methods.

2.1. Binary Morphology

The morphology of a rectangular window can be separated into horizontal and vertical one-dimensional morphologies, and the binary morphology can be easily computed at a constant time per pixel by preparing a counter.

For morphology operations with circular and elliptical windows for binary images, constant-time computation is possible using a distance transform. Distance transform [RP66] is a process that assumes a binary image as input and outputs an array of distances (i.e., called a distance map) to the nearest 0 value of each pixel. Yamada et al. [Yam84] presented an algorithm for computing the Euclidean distance transform [Dan80], which computes the Euclidean distance as a distance function. Dilation can be computed in constant time by setting the pixels whose Euclidean distance from each black pixel is less than or equal to R to black.

2.2. Grayscale Morphology with Rectangular Windows

Although the binary morphology can be computed efficiently using simple AND and OR operations on a binary array, the grayscale morphology is difficult because the minimum and maximum values must be computed. The naïve implementation of grayscale morphology with a window width of W pixels requires only $W - 1$ comparisons per pixel.

The seminal works of van Herk [VH92] and Gil and Werman [GW93] presented efficient methods for computing grayscale morphology. This algorithm is known as the vHGW algorithm. This algorithm corresponds to a rectangular kernel and is a constant-time computation requiring only 3 comparisons in the horizontal and vertical directions for each pixel. This algorithm was further improved by Gil and Kimmel [GK02] to reduce the number of comparisons per pixel to an average of less than 1.5 per dimension, although its implementation was more complex, with more conditional branches. This constant-time algorithm with rectangular windows can be applied to binary operations (e.g., add, mul, etc.) other than min/max, and is currently being studied under the name sliding window aggregation [THS15].

2.3. Arbitrary-shaped Windows

For 1D arrays, the method is based on the fact that morphological operations can be performed with $O(1)$ per pixel for fixed window lengths, and can be applied by decomposing arbitrarily shaped windows into lines [UW08]. In the case of a circular window, this can be computed as $O(H)$ per pixel, where H is the window height. Our method has a complexity of approximately $O(H)$; however, their method must adjust the line morphology H times if there are H lines of windows, whereas ours is a constant order of magnitude improvement because we can integrate multiple lines. Vaz et al. [VKM07] proposed a method for approximately computing $O(H)$ by interpreting a circular window as a superposition of several rectangular windows. Our method works faster because it efficiently reduces the computational cost of the diagonal linear component, although it may involve a small approximation.

Another method for arbitrarily shaped windows is to use

Minkowski addition [Min01] to decompose the window into multiple morphological operations. This method decomposes a window into smaller windows typically called structural elements (SEs). This decomposition allows morphological operations to be computed more efficiently than the morphological operations in the original window. Vanrell et al. [VV97] demonstrated that a circle can be efficiently decomposed into 3×3 windows by approximating it as a hexadecagon. Hashimoto et al. [HBF00] proposed a method to decompose any shape into 3×3 windows. However, the decomposition of circular windows is not always possible for some radii, while radii 2 and 4 are decomposable, and radii 3 and 5-50 are not. Even if it can be decomposed, the decomposition algorithm requires an exponential time from a few seconds to tens of seconds. The number of windows after decomposition is approximately $O(H)$, resulting in a computation time per pixel of approximately $O(H)$. Our method uses the fast greedy method to decompose the input window into fewer than H windows by changing the decomposition rule to a window consisting of two elements instead of decomposing it into 3×3 windows.

2.4. Median Filter

The median filter determined the median values within a window. Morphological operations are related to the median filter in that they use statistical information within the windows. There are two main approaches to median filtering: the sorting network method [Ada21] and the histogram-based method [Gre17]. Moroto et al. [MU22] presented another approach using a wavelet matrix, and discussed its extension to polygonal windows. However, it remains difficult to apply the same approach to morphology operations with circular windows because the computation becomes slow when the polygon has many edges.

2.5. Lens Blur Filter

The lens-blur filter computationally emulates the blur in a defocused photograph. Another filter that performs statistical operations on polygonal windows is lens blur. It uses a flat polygon or circle as the blur kernel, and is available in Adobe Photoshop and other software packages. The lens blur can be used as a filter to calculate the sum or average of the values in a window. There are methods to speed up the process by using subtraction, which is the inverse of addition, by approximating the kernel as a polygon and considering the difference in each direction [MHU21], or by using the fact that the flat kernel is sparse in Laplacian space and performing a convolution in Laplacian space [LSR18]. Our method shares the common approach of approximating a circle as a polygon.

3. Background

The window of the morphology operation \mathcal{K} contains a set of integer coordinates (taps) to offset the indices of the input array. For example, the dilation result \mathcal{J} from the input image \mathcal{I} can be calculated as follows:

$$\mathcal{J}[x][y] = \max_{(dx,dy) \in \mathcal{K}} \mathcal{I}[x+dx][y+dy]. \quad (1)$$

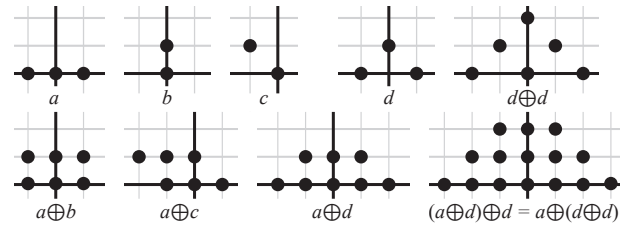


Figure 2: Examples of Minkowski addition of multiple structural elements. The Minkowski addition has the associative property.

3.1. Window Decomposition using Minkowski Addition

The composition of the two morphology operations corresponds to a morphology operation with a larger window, as computed by Minkowski addition of the two windows [Ser84]. The Minkowski addition follows the associative law: For example, let K_1, K_2, K_3 be the shapes of the windows. The associative law results in $(K_1 \oplus K_2) \oplus K_3 = K_1 \oplus (K_2 \oplus K_3)$, where \oplus denotes a morphological convolution operation. In particular, a morphological operation with a large window can be decomposed into several sequential operations using smaller windows. Smaller windows that typically appear after decomposition are often called *structuring elements* (SE). Figure 2 shows several examples of Minkowski addition.

The efficiency of the morphological operations can be significantly improved by decomposing a large input window into smaller SEs [HBF00]. For example, dilating a $W \times H$ rectangular window assumes approximately $O(WH)$ time per pixel in a naïve calculation, while decomposing the window into W -sized horizontal and H -sized horizontal windows assumes approximately $O(W+H)$ time.

Determining the decomposition of an arbitrary window shape is an NP-complete problem [SPR97], which is extremely difficult. Hashimoto et al. [HBF00] presented a technique to determine whether decomposition existed, but the algorithm grew exponentially for large window sizes. As their approach finds a decomposition with 3×3 SEs, the search space is limited. This study presents a method for instantly determining the exact decomposition of most circular windows. Furthermore, the proposed algorithm determines an approximate decomposition even when an exact solution is unavailable.

3.2. vHGW Algorithm for Rectangular Windows

The vHGW algorithm [VH92, GW93] computes the maximum (minimum) value inside and within rectangular windows for a 2D grayscale image $O(1)$ times per pixel, regardless of the window size.

First, we describe how the algorithm determines the minimum and maximum within an interval of size W for an input 1D grey scale array \mathcal{I} . The algorithm computes the intermediate buffers \mathcal{F}

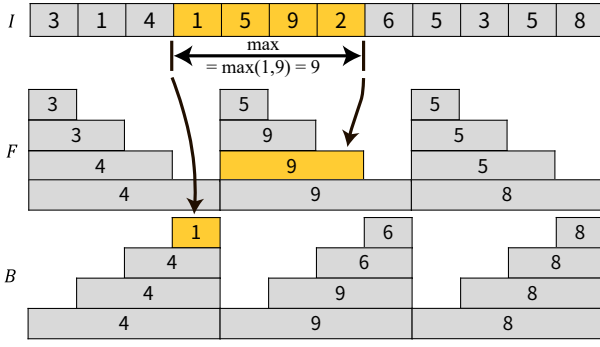


Figure 3: Example of using the vHGW algorithm to find the maximum in a window of length 4. The array is divided by the length of the window, and the cumulative maximum is computed from the beginning for the array \mathcal{F} and from the end for the array \mathcal{B} . Immediately \mathcal{F} and \mathcal{B} are computed, the maximum of an interval of length 4 can be obtained at any point by assuming the larger element of \mathcal{F} and \mathcal{B} .

and \mathcal{B} as follows:

$$\mathcal{F}[x] = \begin{cases} \mathcal{I}[x] & \text{if } x \equiv 0 \pmod{W}, \\ \mathcal{I}[x] \odot \mathcal{F}[x-1] & \text{otherwise,} \end{cases} \quad (2)$$

$$\mathcal{B}[x] = \begin{cases} \mathcal{I}[x] & \text{if } x \equiv -1 \pmod{W}, \\ \mathcal{I}[x] \odot \mathcal{B}[x+1] & \text{otherwise,} \end{cases} \quad (3)$$

where \odot is the binary operation corresponding to the desired morphological operation, such as maximum or minimum. Arrays \mathcal{F} and \mathcal{B} are the results of cumulative operations in the forward and backward directions, respectively, within chunks, where the input array is divided into chunks of length W . Note that if the length of the input is not a multiple of W , the remainder will be filled with dummy values (e.g., 255 for a min operation on 8-bit values). Figure 3 shows an example of how \mathcal{F} and \mathcal{B} are constructed for the input \mathcal{I} .

Using these intermediate computations, an array of maximum/minimum values \mathcal{J} in an interval of length W from the i th element is computed as follows:

$$\begin{aligned} \mathcal{J}[x] &= \bigodot_{\hat{x}=x}^{x+W-1} \mathcal{I}[\hat{x}], \\ &= \mathcal{B}[x] \odot \mathcal{F}[x+W-1]. \end{aligned} \quad (4)$$

The number of comparison operations performed during the construction of \mathcal{F} and \mathcal{B} is at most $2N$, where N is the length of the input \mathcal{I} . Because N comparison operations are performed during the construction of \mathcal{J} , the maximum value of the fixed-length window can be obtained in approximately $3N$ comparisons.

This algorithm is memory-efficient because it uses only $O(W)$ space. We reuse the buffers of size W for \mathcal{F} and \mathcal{B} repeatedly for each chunk. The morphology of the rectangular window for the images can be computed by applying the algorithm to the horizontal and vertical directions separately, as described in Section 3.1.

4. Method

In this study, we present a method for decomposing large circular windows into a set of SEs with two points, where the Minkowski additions (Section 3) of these SEs reconstruct the original circular windows. We define a circular window of radius R as:

$$K_R = \{(x, y) \in \mathbb{Z}^2 : x^2 + y^2 \leq R^2\}. \quad (5)$$

Figure 4(a) shows a sample window for $K_{\sqrt{27}}$. This definition accurately captures the shape of a circle, contrary to the hexadecagon approximation proposed by Vanrell et al. [VV97].

Algorithm Our decomposition algorithm is based on our simple but powerful heuristic in that the boundary edges of the circular windows produce good SEs. Specifically, we computed the convex hull of the upper half of the circular window and extracted the integer coordinates of the edge directions as SEs (see Figure 4(b)). This simple strategy decomposes most of the circular windows of different radii. For example, out of 10000 circular windows with radii ranging from $\sqrt{1}$ to $\sqrt{10000}$, 8699 can be decomposed exactly without errors. If the convex hull decomposition has more than two identical edges, the computation can be accelerated by grouping them (Section 4.1).

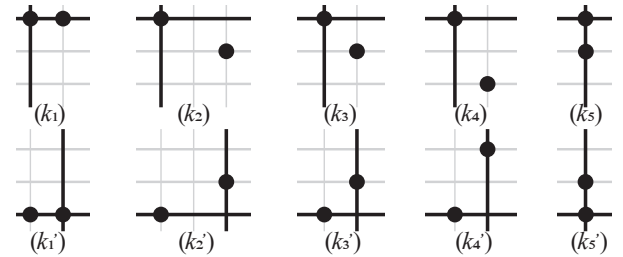
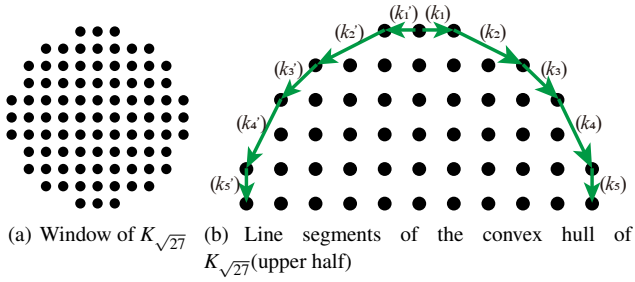
Empirically, we observed that significant errors can occur only when the circular window pattern has only one element in the top row (see Section 4.2). In these cases, the reconstructed patterns may be sparse. This problem can easily be alleviated by slightly changing the radius or adding another element to the top row. Consequently, the resulting decomposition has only a small error that is hardly visible if an error exists.

Figure 4(b) presents the decomposition when $R^2 = 27$, where the decomposition can be computed by extracting the edges of the convex hull. Because the computational complexity of the morphology operation is proportional to the number of elements in the SEs, the computational complexity is significantly reduced compared with performing the morphology directly on the original circle window. The time required to decompose the SE is also $O(R)$ to scan the convex hull of the circle.

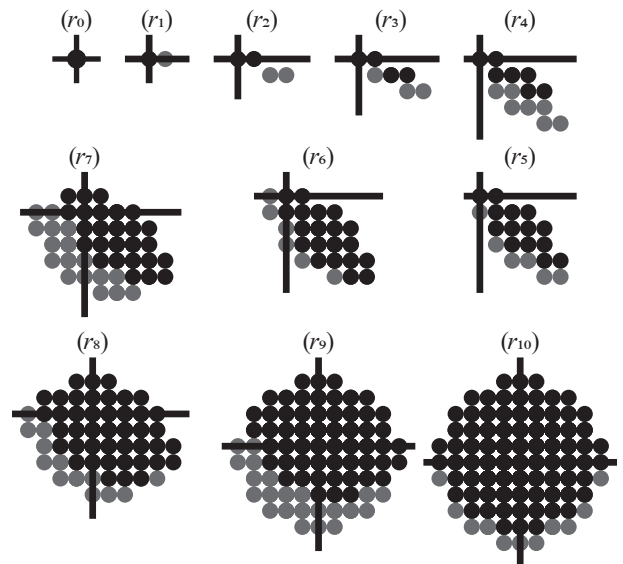
4.1. Speed Up by Grouping Identical Segments

We extracted the smallest edges when decomposing a convex hull. For example, the convex hull of $K_{\sqrt{30}}$ (see Figure 5) contains long line segments $\{(2,0), (3,3), (0,2)\}$, which the elements (i.e., taps) touch these line segments. The decomposition of these long segments should yield the 7 smallest line segments $\{(1,0), (1,0), (1,1), (1,1), (1,1), (0,1), \text{ and } (0,1)\}$ to reconstruct the input window.

Subsequently, we grouped the line segments that appeared more than thrice. For example, line segments $(1,1), (1,1), (1,1)$ were grouped into fewer SEs $\{(1,1), (2,2)\}$. If there are n SEs with $(x, y) \in \mathbb{Z}^2 : \gcd(x, y) = 1$, we group $1, 2, 4, \dots, 2^k, r$ SEs using the Minkowski addition. Note that $k = \lfloor \log_2(n) - 1 \rfloor$ and r is the remainder; that is, $r = n - (1 + 2 + 4 + \dots + 2^k)$. For example, we divided the 5 SEs into groups of $\{1, 2, 2\}$ SEs as well as the 15 SEs into groups of $\{1, 2, 4, 8\}$ SEs. This grouping did not change the results and reduced the number of SEs that contributed to the acceleration.



(c) Line segments of the convex hull are used as SEs. Note that the positions of the origin are adjusted to balance the up- and down-movements.



(d) Morphological operations are performed on the decomposed SEs to restore the original circular window. $r_1 = k_1 \oplus r_0, r_2 = k_2 \oplus r_1, \dots, r_6 = k'_1 \oplus r_5, \dots, r_{10} = k'_5 \oplus r_9 = K_{\sqrt{27}}$.

Figure 4: The circular window is decomposed by extracting the line segments of the convex hull of the window. The original window $K_{\sqrt{27}}$ is exactly reconstructed by the Minkowski additions of the decomposed structural elements (SEs) as $K_{\sqrt{27}} = k_1 \oplus k_2 \oplus \dots \oplus k_5 \oplus k'_1 \oplus k'_2 \oplus \dots \oplus k'_5$.

Next, although this is optional we applied the vHGW algorithm (see Section 3.2) to the long horizontal line segment for further acceleration. For example, $K_{\sqrt{30}}$ contains a long horizontal line segment $(4, 0)$, which is decomposed into $\{(1, 0), (2, 0), \text{ and } (1, 0)\}$ after the previous grouping procedure. A line element of length n is decomposed into $\lfloor \log_2 n \rfloor + 1$ SEs, and because morphology operations assume time proportional to the number of SEs, they will assume approximately $O(\log n)$ per pixel. Here, the horizontal morphology is processed in $O(1)$ time per pixel by using the vHGW algorithm.

The use of the vHGW algorithm for the vertical morphology, whose components are represented by $(0, y)$, theoretically results in a smaller computational complexity. However, in practice, we found it difficult to compute the morphology operation faster with this approach. Furthermore, the complexity of the implementation increases because a separate vHGW must be prepared for the vertical direction, and additional buffer memory is required. Regarding the number of decomposed SEs, all SEs except the horizontal SEs have a y component greater than or equal to 1; thus, the number of SEs is less than or equal to $2R + 1$.

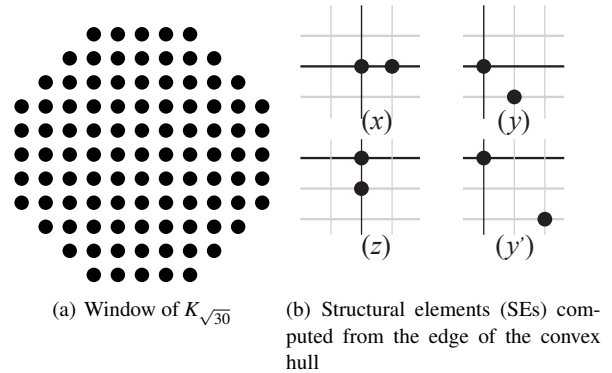


Figure 5: The line segments that form the upper right part of the convex hull of $K_{\sqrt{30}}$ consist of seven elements, x, x, y, y, z, z, z . The number can be reduced to six by decomposing as x, x, y, y', z, z .

4.2. Approximated Decomposition

Of the 10000 circular SEs from $K_{\sqrt{1}}$ to $K_{\sqrt{10000}}$, our algorithm finds 8699 exact decomposition patterns; however, the remaining 13% of the decomposed SEs do not reconstruct the original window shapes. Below, we describe two cases in which exact decomposition is impossible.

In the first case, the first row (and thus, the last column) has only one element, and the transition to the next SE is not smooth during decomposition. The decomposition of this window shape has an incorrect grid pattern and is significantly different from that of the input. Figure 6 shows the example of $K_{\sqrt{25}}$. This problem is caused by the lack of smallest SEs $\{(1, 0), (0, 1)\}$ in the decomposition.

There are two simple methods to overcome this problem. One approach is to check whether the number of elements in the first row is one, and if it is, slightly modify the radius such that there are

at least two elements in the first row. Another approach is to add new elements next to a single element in the first row before the decomposition (see Figure 6(d)). To check whether the number of elements in the first row is one, we use the condition $\|(1, \lfloor \sqrt{R} \rfloor)\| = \sqrt{1 + \lfloor \sqrt{R} \rfloor^2} < R$.

The second case of exact decomposition failure occurs when the edges of the reconstructed window are slightly jagged compared with the input. Because our method considers only the convex hull of the window, a boundary that is not touched by the convex hull cannot be faithfully reproduced. Figure 7 shows the reconstruction of the concave part of the input window, which is missing from the example in the window of $K_{\sqrt{73}}$. It is difficult to fix a jagged border. However, as far as we tested with many circular windows, the error was considerably small and not noticeable in practical applications. If readers are interested in the size of the error, please refer to the plots of the reconstructed circular windows with radii ranging from $\sqrt{1}$ to $\sqrt{10000}$ in the supplementary material.

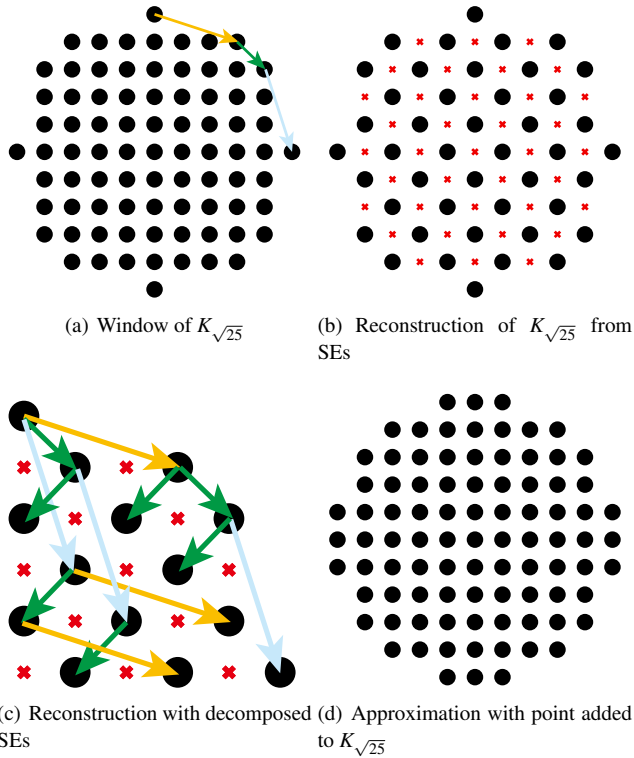


Figure 6: Decomposing and reconstructing $K_{\sqrt{25}}$ from input (a) yields a sparse grid pattern as observed in (b). This is because the decomposed SEs $\{(3,1), (1,1), (1,3)\}$ do not contain any smallest SEs $\{(1,0), (0,1)\}$ that can move directly beside or below, a point cannot move from the top to any element inside the input window, as shown in (c). To avoid this, circular windows with only one element in the first row can be rejected before decomposition. Alternatively, as in (d), we add new points in the first and last rows and columns.

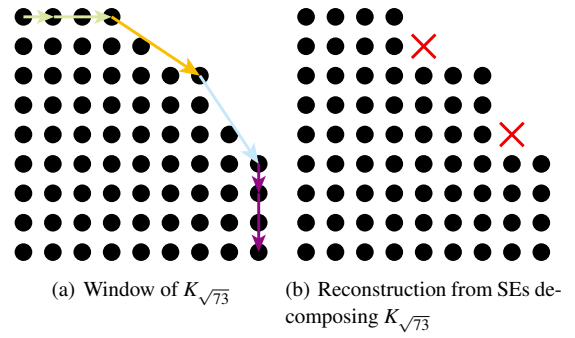


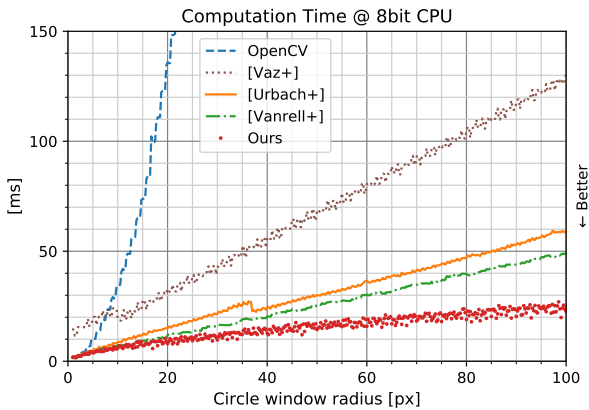
Figure 7: Because our method is a decomposition of circular windows using convex hulls, the vertices of convex hulls can be reconstructed correctly; however, other edges may not be reconstructed correctly depending on the decomposed SEs. Considering the reconstruction (b), two points are missing.

5. Results

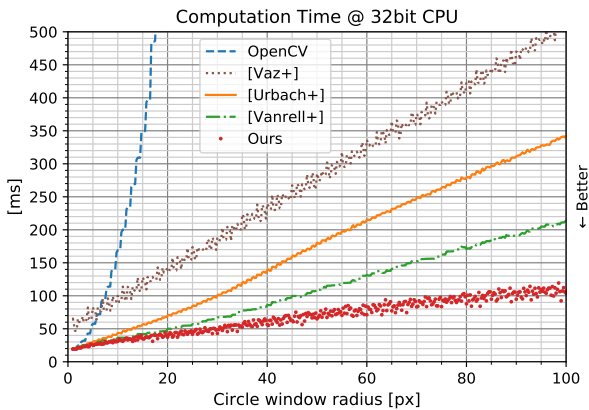
This section describes runtime measurements of the proposed algorithm. All benchmarks were run on the 4000×2162 pixel grayscale version of the input image in Figure 1 with an Intel i9-9900X CPU and an NVIDIA GeForce 3090 RTX GPU running on Ubuntu 20.04. Measurements were considered 7 times and averaged by subtracting the maximum and minimum values. The benchmark codes written in C++ and CUDA are included in the Supplementary Material. Upon acceptance, the code and data will be published to make the results more reproducible.

We compared our method to several existing methods and a popular open-source package, OpenCV, with a single-threaded CPU implementation and a parallelized implementation on the GPU. Figure 8 presents the results. The details of these methods are as follows.

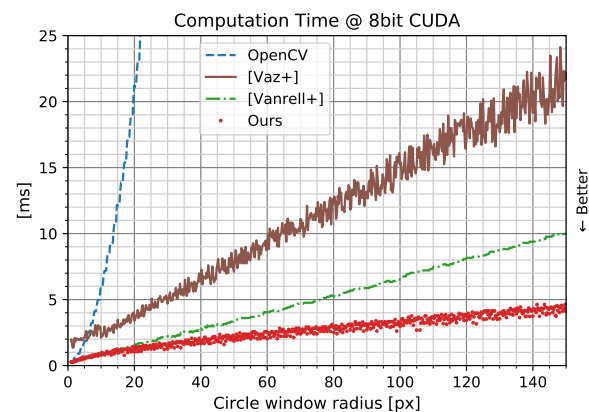
1. OpenCV is one of the world's most widely used open-source image processing libraries and supports morphological operations, including dilation and erosion. For SE, MORPH_RECT, MORPH_CROSS, and MORPH_ELLIPSE are available as templates. Efficient implementations are provided for RECT and CROSS. However, for ELLIPSE, the implementation is $O(R^2)$ time per pixel to brute-force the window (R is the window radius). However, its brute-force method is highly optimized and has a low constant multiplier.
2. Urbach et al. [UW08] proposed a morphological operation method that is more efficient than the brute-force method by partitioning windows of arbitrary input shapes into rows. For a convex hull, such as a circle, this algorithm decomposes it into $2R + 1$ one-dimensional SEs, and morphological operations can be performed in $O(R)$ time per pixel. Reusing row buffers allows efficient filtering even for large images. However, efficient parallelization on GPUs was not obvious this comparison; thus, was done on CPUs only.
3. Vaz et al. [VKM07] showed an accurate and efficient circular morphology operation by superimposing a circular window with several rectangular windows. It is particularly efficient when the



(a) Comparison in 8bit. Our method has some variability in execution time, but fastest for radii larger than 2. OpenCV uses a brute-force method; however, it slower as the radius increases, but is very fast at smaller radii. For a radius of 100, our method was approximately 2.1 times faster than the second best method.



(b) Comparison with 32-bit float. Even with high depth images, our method was efficient and 2.0 times faster at a radius of 100.



(c) Comparison on GPU with CUDA. When the radius was small, kernel startup time had a large impact, and there was slight difference in speed between the methods. Our method efficiently handled parallelization with many threads, and was approximately 2.1 times faster when the radius was 150.

Figure 8: Comparison with existing methods and the popular image processing library. The lower the value, the better.

edges of the discretized circular windows contain horizontal and vertical linear components. Their algorithm has a computational complexity of $O(R)$. However, in the case of grayscale, multiple image buffers are required, and they are a memory speed limiter.

4. Vanrell et al. [VV97] showed that the circular windows can be efficiently decomposed into 3×3 SEs by approximating them as hexadecagons. This approach decomposes the input into approximately R SEs, and by repeating the morphology operation for the number of SEs, the hexadecagonal morphology operation is performed at $O(R)$ per pixel. Because our method does almost the same except for the SE decomposition method, the implementation was almost the same as our method.

Our method has some variations in runtime because the decomposed SE varies finely with the radius of the circle; however, it is faster than any other method when the radius is approximately eight or larger. When the circle is small, our method is similar to Vanrell et al. [VV97]'s method, which approximates the circle by a hexadecagon; thus, it has the same speed. However, our method limits the SEs to 3×3 , as well as decomposes them using large SEs; therefore, if the circle is large, our method has an advantage. With 8 bits of CPU, the proposed method was approximately twice as fast at a radius of 50, more than 2 times faster at a radius of 100, and five to eight times faster at a radius of 500. At 32 bits, the speed of all methods decreased as the data size increased. The method proposed by Vanrell et al. [VV97] was considered to have high memory locality and easy data storage in a fast cache because of its row-by-row computation, and the effect of the high depth was small.

Our method and that of Urbach et al. [UW08] use temporary buffers of approximately $W \times D$ in size, excluding inputs and outputs, where W is the width of the image and D is the diameter of the circular window. Vanrell et al.'s method requires only W temporary buffers, making it memory efficient and easy to fit into a fast cache. However, our method and that of Vanrell et al. [VV97], which decomposes data into SEs of approximately half the number of rows or less, are faster, and our method is approximately 2.5 times faster than that of Urbach et al. [UW08] at a radius of 100.

Furthermore, because our method decomposes SEs and performs several simple morphological operations, it can be efficiently parallelized using many threads on a GPU. In particular, a comparison using CUDA shows that our method is 64 times faster than OpenCV and 1.8 times faster than Urbach et al. at a radius of 50, and more than 2.5 times faster at a radius of 150.

In this comparison, the method of Vanrell et al. [VV97] produced results that approximated a circle as hexadecagon, and our method produced almost accurate circular morphology operations. On the other hand, the other methods calculated perfect circular morphology operations. Our method could not accurately recover 13% of the circles with radii of up to $\sqrt{10000}$. Figure 9 graphically illustrates the reconstruction errors for different radii in the heat map. If the radius is an exact integer, the number of SEs in the top row is 1; thus, it cannot be accurately represented (see Section 4.2). The other radii at which errors occurred were unevenly distributed. Figure 10 shows a circular window of radius $\sqrt{1514}$, which is one of the windows with the highest error. The filtered results obtained using this window are shown in Figure 11. By close observation, the

circular bokeh is slightly jagged in the decomposed result; otherwise, the difference is invisible. If the error is critical, we can check in advance whether the decomposed windows or the software can prevent the user from selecting problematic radii.

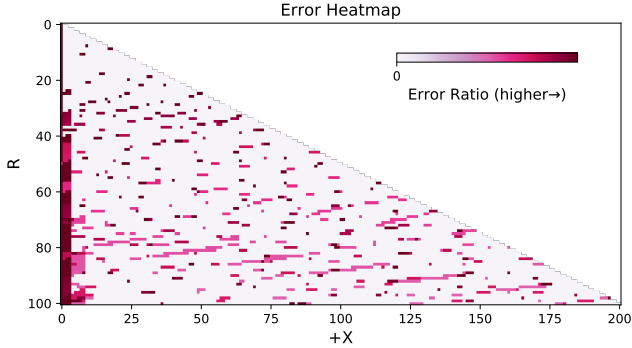


Figure 9: This heatmap shows how much error occurs when reconstructing a decomposed circular window using our method. The vertical axis is R and the horizontal axis is X , showing the amount of error in a circle with radius $\sqrt{R^2 + X}$. The area of the error is shown in pink, and the larger the error, the darker the red. The number of pixels with errors is counted and divided by the radius as the error.

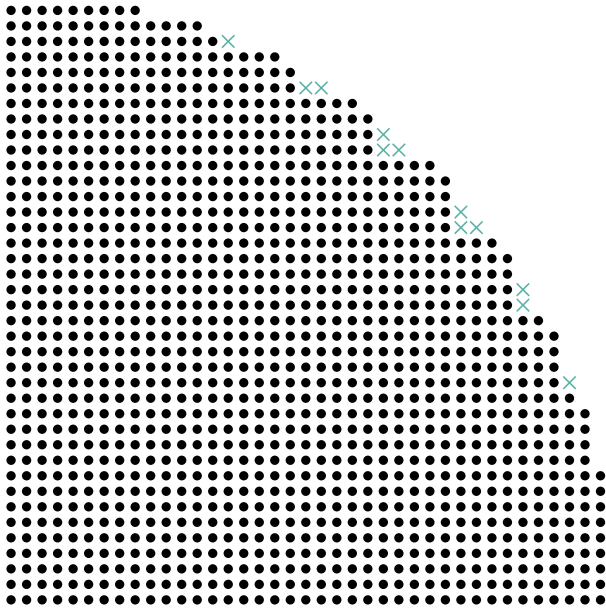


Figure 10: Circular window with radius $\sqrt{1514} \approx 38.91$ reconstructed with our method. This is one of the radii with the largest error. There are 1226 grid points in the circle, of which 12 points near the edges cannot be recovered (marked with \times).

6. Limitations and Future Work

The proposed method approximates a circular window with a convex hull. Approximately 87% of the radii, including the fractional



(a) Ground truth

(b) Ours



(c) Ground truth scale-up

(d) Ours scale-up

Figure 11: Filtering results with the window of radius $\sqrt{1514}$, one of the windows with the highest error. The edges are slightly jagged compared to the ground truth when extremely zoomed in, but we consider these small artifacts visually acceptable.

radii, could be filtered exactly without errors. However, the remaining 13% was not computed exactly, and a small artifact was observed around the edges. Although this is not a major visual problem, it is a fast and ideal filtering method for future studies.

In this study, we focused on circular windows. However, the method can be extended to ellipses and arbitrary convex hulls. Because circular windows are in great demand for morphological operation filters as well as for various other types of filters, we believe it is appropriate to target circular windows; however, we await applications to other window shapes as a technical challenge. In the case of circular windows, the directions of the edges of the convex hull changed smoothly; thus, the reconstruction error was rather small. However, if we extend this method to ellipses or arbitrary convex hulls, we may need to add further constraints to the decomposition.

The computational complexity of the proposed algorithm is $O(R)$ per pixel radius, R . Although this is relatively fast, exploring algorithms faster than $O(R)$ is a future challenge because morphological operations can be performed in $O(1)$ for rectangular windows and in $O(1)$ for circular windows in the case of binary images.

In this study, we propose a morphological operation algorithm for two-dimensional images. However, the direct application of this algorithm to three dimensions is not straightforward. Morphologi-

cal operations on 3D voxel grids may prove useful in cases dealing with density or occupancy fields. Thus, future research in this area is anticipated.

7. Conclusion

We focused on circular windows and proposed a morphological computation method for decomposing SEs that is several times faster than existing methods. Most importantly, existing methods impose a limit of 3×3 on the window size after decomposition when decomposing large windows; however, we used a decomposition method that does not limit the size of the SEs instead of limiting the number of elements to two. This reduces the total number of SEs; thus, the computational complexity. Our method is an approximate method, similar to the conventional SE decomposition method. However, the proposed method corresponds to approximately 87% of the decimal radii and provides an exact decomposition, whereas the conventional method itself has only a few decomposable circular windows. Even if exact decomposition is impossible, filtering can be performed without a significant visual impact. Our method also supports parallel operations and is significantly faster than the existing methods on GPUs.

Acknowledgements

We thank the anonymous reviewers for valuable feedback that improved our manuscript. This work has been partially funded by the Support for Pioneering Research Initiated by Next Generation (SPRING) program offered by the Japan Science and Technology Agency (JST).

References

- [Ada21] ADAMS A.: Fast median filters using separable sorting networks. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–11. 3
- [Dan80] DANIELSSON P.-E.: Euclidean distance mapping. *Computer Graphics and image processing* 14, 3 (1980), 227–248. 2
- [DD11] DOKLÁDAL P., DOKLÁDALOVÁ E.: Computationally efficient, one-pass algorithm for morphological filters. *Journal of Visual Communication and Image Representation* 22, 5 (2011), 411–420. URL: <https://www.sciencedirect.com/science/article/pii/S1047320311000393>, doi:<https://doi.org/10.1016/j.jvcir.2011.03.005>. 2
- [DT96] DROOGENBROECK M., TALBOT H.: Fast computation of morphological operations with arbitrary structuring elements. *Pattern Recognition Letters* 17 (12 1996), 1451–1460. doi:[10.1016/S0167-8655\(96\)00113-4](https://doi.org/10.1016/S0167-8655(96)00113-4). 2
- [GK02] GIL J. Y., KIMMEL R.: Efficient dilation, erosion, opening, and closing algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24, 12 (2002), 1606–1617. 2
- [Gre17] GREEN O.: Efficient scalable median filtering using histogram-based operations. *IEEE Transactions on Image Processing* 27, 5 (2017), 2217–2228. 3
- [GW93] GIL J., WERMAN M.: Computing 2-d min, median, and max filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15, 5 (1993), 504–507. 2, 3
- [HBF00] HASHIMOTO R. F., BARRERA J., FERREIRA C. E.: A combinatorial optimization technique for the sequential decomposition of erosions and dilations. *Journal of Mathematical Imaging and Vision* 13, 1 (2000), 17–33. 2, 3
- [HSZ87] HARALICK R. M., STERNBERG S. R., ZHUANG X.: Image analysis using mathematical morphology. *IEEE transactions on pattern analysis and machine intelligence*, 4 (1987), 532–550. 2
- [KCRU57] KIRSCH R. A., CAHN L., RAY C., URBAN G. H.: Experiments in processing pictorial information with a digital computer. In *Papers and Discussions Presented at the December 9-13, 1957, Eastern Joint Computer Conference: Computers with Deadlines to Meet* (New York, NY, USA, 1957), IRE-ACM-AIEE '57 (Eastern), Association for Computing Machinery, p. 221–229. 2
- [LSR18] LEIMKÜHLER T., SEIDEL H.-P., RITSCHEL T.: Laplacian kernel splatting for efficient depth-of-field and motion blur synthesis or reconstruction. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–11. 3
- [MHU21] MOROTO Y., HACHISUKA T., UMETANI N.: Fast polygonal splatting using directional kernel difference. In *EGSR (DL)* (2021), pp. 99–109. 3
- [Min01] MINKOWSKI H.: Ueber die begriffe länge, oberfläche und volumen. *Jahresbericht der Deutschen Mathematiker-Vereinigung* 9, 1 (1901), 115–121. 3
- [MU22] MOROTO Y., UMETANI N.: Constant time median filter using 2d wavelet matrix. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–10. 3
- [RP66] ROSENFELD A., PFALTZ J. L.: Sequential operations in digital picture processing. *Journal of the ACM (JACM)* 13, 4 (1966), 471–494. 2
- [Ser84] SERRA J.: *Image Analysis and Mathematical Morphology, Volume 1*, paperback ed. Academic Press, 2 1984. URL: <https://lead.to/amazon/com/?op=bt&la=en&cu=usd&key=012637242X>. 3
- [SPR97] SUSSNER P., PARDALOS P. M., RITTER G. X.: On integer programming approaches for morphological template decomposition problems in computer vision. *Journal of Combinatorial Optimization* 1 (1997), 165–178. 3
- [THS15] TANGWONGSAN K., HIRZEL M., SCHNEIDER S.: Constant-time sliding window aggregation. *IBM, IBM Research Report RC25574 (WAT1511-030)* (2015). 2
- [UW08] URBACH E. R., WILKINSON M. H. F.: Efficient 2-d grayscale morphological transformations with arbitrary flat structuring elements. *IEEE Transactions on Image Processing* 17, 1 (2008), 1–8. doi:[10.1109/TIP.2007.912582](https://doi.org/10.1109/TIP.2007.912582). 2, 6, 7
- [VH92] VAN HERK M.: A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recognition Letters* 13, 7 (1992), 517–521. 2, 3
- [VKM07] VAZ M. S., KIRALY A. P., MERSEREAU R. M.: Multi-level decomposition of Euclidean spheres. In *Proc. Int. Symp. Math. Morphology (ISMM) 2007* (2007), pp. 461–472. 2, 6
- [VV97] VANRELL M., VITRÌA J.: Optimal 3×3 decomposable disks for morphological transformations. *Image and Vision Computing* 15, 11 (1997), 845–854. 2, 3, 4, 7
- [Yam84] YAMADA H.: Complete euclidean distance transformation by parallel operation. In *ICPR Proceedings* (1984), pp. 69–71. 2