Parallel Implementation of Morphological Operations on Binary Images Using CUDA

Chapter *in* Lecture Notes in Electrical Engineering · January 2016

DOI: 10.1007/978-3-319-32213-1_15

4 authors, including:

Jun Ming Koay
Malaysian Institute of Microelectronic Systems
1 PUBLICATION **5** CITATIONS
SEE PROFILE

Shahirina mohd tahir
Malaysian Institute of Microelectronic Systems
6 PUBLICATIONS **45** CITATIONS
SEE PROFILE

Dr. Sankaraiah Sreeramula
12 PUBLICATIONS **77** CITATIONS
SEE PROFILE

Parallel Implementation of Morphological Operations on Binary Images Using CUDA

Jun Ming Koay, Yoong Choon Chang, Shahirina Mohd Tahir and Sankaraiah Sreeramula

Abstract Morphology is a common technique used in image processing because it is a powerful tool with relatively low complexity. Albeit simple, morphological operations are typically time consuming due to the fact that the same operations are repeated on every pixel of an image. Since the processing of the pixels of an image is an embarrassingly-parallel process, the morphological operations can be carried out in parallel on Nvidia graphic cards using Compute Unified Device Architecture (CUDA). However, most of the existing CUDA work focuses on the morphological operations on grayscale images. For binary image, it can be represented in the form of a bitmap so that a 32-bit processor will be able to process 32 binary pixels concurrently. With the combination of the bitmap representation and van Herk/Gil-Werman (vHGW) algorithm, the performance of the proposed implementation in term of computation time improves significantly compared to the existing implementations.

Keywords Morphology · Binary images · Bitmap · Parallel · CUDA · GPU · vHGW

J.M. Koay (✉)
Faculty of Engineering, Multimedia University, 63100 Cyberjaya, Selangor, Malaysia
e-mail: kjm_coffee@hotmail.com

Y.C. Chang
Faculty of Engineering and Science, Universiti Tunku Abdul Rahman, 43000 Kajang, Selangor, Malaysia
e-mail: ycchang@utar.edu.my

S.M. Tahir · S. Sreeramula
MIMOS Bhd, Technology Park Malaysia, 57000 Kuala Lumpur, Malaysia
e-mail: shahirina.mtahir@mimos.my

S. Sreeramula
e-mail: sankaraiah.ramula@mimos.my

P.J. Soh et al. (eds.), *Advances in Machine Learning and Signal Processing*,
Lecture Notes in Electrical Engineering 387, DOI 10.1007/978-3-319-32213-1_15

1 Introduction

Morphology is a common technique used in image processing, since it is a simple and powerful tool. In general, morphology can be applied on both grayscale and binary images. The two main operations in morphology are erosion and dilation. In the context of grayscale images, erosion is the process of scanning through the neighbouring pixels under consideration and assigning the smallest value to the central pixel. On the other hand, dilation is the complete opposite process of erosion, where instead of finding the smallest pixel value, the largest pixel value is assigned to the central pixel. By combining the erosion and dilation operations, more complicated morphological operations such as opening and closing can be performed.

In order to process an input image using morphological operator, an extra input which is known as structuring element is also required. The structuring element selects the neighbouring pixels that need to be considered during the morphological operations. In fact, the structuring element can have any type of shapes and sizes. If an erosion operation is performed using a 3×3 structuring element, the pixel value of all the nine pixels inside the window are compared with each other and the smallest value is assigned to the central pixel.

Although morphological operations are straightforward, its process is time consuming because the same operations need to be repeated on every pixel of an image. Hence, there is a need to speed up the operations for real-time applications. Since the operations on the pixels are independent of each other, the pixels can be processed in parallel in order to speed up the morphological operations.

There are a few researchers that have implemented the morphological operations on parallel programming platforms like Nvidia's Compute Unified Device Architecture (CUDA). However, most of the implemented work is focused only on grayscale images [2], [5]. Although the grayscale implementation of morphological operations can also be used to process binary images, it is not an optimised way to handle binary images because an 8-bit grayscale pixel has 256 possible values while a binary pixel can only have two possible values. Hence, special optimisation techniques that utilise the bi-level nature of the binary pixels are proposed in this paper in order to further speed up the computation time of the morphological operations.

The rest of the paper is organised as follows. Section 2 discusses related work on the implementation of morphological operations using CUDA. In Sect. 3, the proposed CUDA implementation is explained in detail. Section 4 compares the performance of the proposed CUDA implementation with the existing implementations. Finally, Sect. 5 concludes the proposed work.

2 Related Work

2.1 Decomposition of Structuring Element

For a morphological operation using a rectangular structuring element with a size of $x \times y$, the basic operations need to be repeated for xy times to obtain the output of one single pixel. Hence, for an image with n number of pixels, the total number of operations that need to be executed is nxy. Nevertheless, the computational complexity can easily be reduced by using a technique known as structuring element decomposition [4]. By using this technique, a rectangular structuring element can be decomposed into one-dimensional horizontal and vertical structuring elements. This means that instead of using a rectangular structuring element to perform the morphological operations, the same output can be obtained by using a horizontal structuring element followed by a vertical structuring element. This optimises the computational complexity from nxy to $n(x + y)$. By using this technique, the speedup obtained for a 5×5 structuring element is approximately 2.5 times.

2.2 vHGW Algorithm

One of the drawbacks of the conventional way of implementing morphology is the computation time increases as the size of the structuring element becomes larger. In order to solve this problem, van Herk [7] and Gil and Werman [3] proposed a method known as van Herk/Gil-Werman (vHGW). By using the vHGW algorithm, the computation time remains relatively constant regardless of the structuring element size. For an erosion operation on a grayscale image using a horizontal structuring element of size p, the steps in the vHGW algorithm are explained as follow.

1. The image rows are split into multiple horizontal segments of size p. In addition, aprons with a size of $(p-1)/2$ are appended to both the left and right ends of the segments to form segments with a total size of $2p-1$, which is shown in Fig. 1.
2. Starting from the midpoint of the segments, a cumulative comparison is carried out for the elements on the left half of the segment and the minimum values are

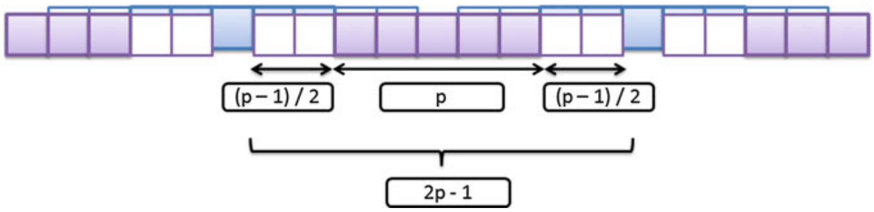

Fig. 1 First step of vHGW algorithm: dividing the image rows into multiple segments

stored inside a new array known as suffix array. The same operation is repeated for the elements on the right half of the segment and the results are stored inside another array known as prefix array.

3. The value of the elements in the suffix array are compared with the value of the elements in the prefix array to obtain the final minimum values.

2.3 CUDA Implementation of the vHGW Algorithm

In general, for most of the morphological operations, the pixels of an image can be processed independently. Hence, it offers an opportunity to speed up the processing by parallelising the implementation on a many-core platform. In this case, graphics processing units (GPUs) are well suited to perform the morphological operations in a parallel manner due to its highly parallel structure. There are various computing platforms that are available for the implementation of general purpose computing on a GPU, such as openCL, openGL and CUDA. For the proposed research work, CUDA is chosen as the computing platform because it is a matured platform in the field of computer vision, with support from computer vision libraries like OpenCV.

Various work has been done on the implementation of the morphological operations using CUDA. Domanski, Vallotton and Wang [2] implemented the vHGW algorithm using CUDA and observed that the CUDA version of the algorithm achieves a speedup of 13 to 33 times compared to the serial implementation. Domanski et al. also showed that the computation time of the vHGW CUDA implementation remains relatively constant as the size of the structuring element varies. However, the vHGW CUDA implementation by Thurley and Danell [5] showed that only the horizontal structuring element exhibits the constant-time behaviour. For the vertical structuring element, the computation time increases as the size of the structuring element becomes larger. In order to overcome this problem, Thurley et al. replaced the vertical morphological operation with another algorithm known as transpose-horizontal-transpose (THT). For the THT method, the input image is transposed first before a horizontal structuring element is applied on the image. The processed image is then transposed again to obtain the final output.

3 Implementation of Morphological Operations Using CUDA

Instead of using the data structure of a grayscale image to store a binary image, Van Den Boomgaard and R. van Balen represented the binary image in the form of a bitmap, so that only one bit of memory is required to store the value of a particular binary pixel [6]. As a result of this, a 1-byte memory space is able to store the value of eight binary pixels and the required memory space is reduced by eight times. As most of the modern processors are 32-bit or 64-bit processors, if the binary image is represented in the form of a bitmap, all 32 or 64 binary pixels can be processed in parallel using one single processor. By using the bitmap optimisation technique, the speedup attained can be more than 30 times. Thus, the bitmap representation is used in the proposed implementation.

The morphological operations on bitmapped binary images are implemented by modifying the grayscale implementation. For the implementation of the erosion operation, the 'min' operator in the grayscale implementation is replaced with the logical 'and' operator. In a similar way, the 'max' operator is changed to the logical 'or' operator for the implementation of the dilation operation.

The serial implementation of the morphological operations is time consuming, even by using the vHGW algorithm and bitmap representation. Hence, there is a necessity to speed up the implementation of the morphological operations by parallelising it on CUDA. A CUDA-enabled GPU operates based on the parallel execution model of single instruction, multiple threads (SIMT). In this architecture, multiple CUDA threads run concurrently in the GPU. For the implementation of the morphological operations on grayscale images, each CUDA thread is assigned the task of calculating the value of an output pixel. Hence, the number of CUDA threads is set according to the resolution of the image. With the introduction of bitmap representation, one single CUDA thread is able to process 32 binary pixels. Hence, the number of CUDA threads needed is reduced by 32 times. The parallelisation strategy for processing a bitmapped binary image is shown in Fig. 2, where N is the number of CUDA threads per block while M is the number of blocks needed to process the whole image.

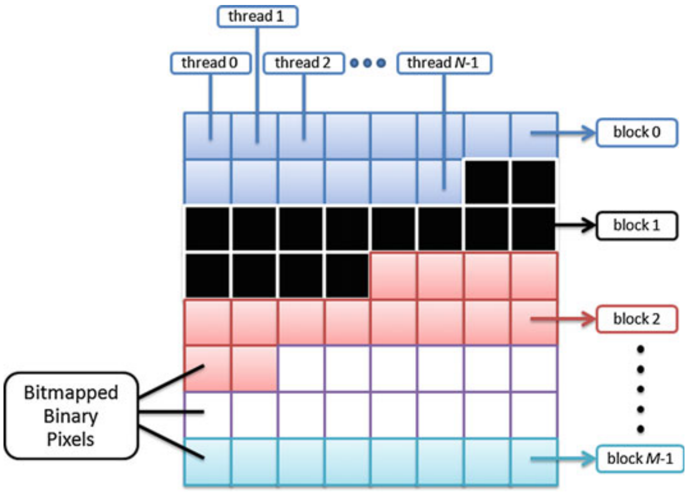

Fig. 2 Parallelisation strategy for processing a bitmapped binary image in the GPU using CUDA

During the morphological operations, the same pixel location is read multiple times. Since the data access to the shared memory is faster than the global memory of the GPU, the pixel data that are accessed repeatedly should be stored in the shared memory. Nevertheless, the size of the shared memory is only limited to 48 kB for GPUs with compute capability 3.5 [1]. Thus, for large binary images with Full High Definition (FHD) resolution, the image is not able to fit into the shared memory. Hence, it is very important to schedule the usage of the shared memory in such a way that at any point of time, only a small segment of the binary image is stored in the shared memory.

Although the bitmap representation reduces the computational complexity of the CUDA implementation, the computation time increases when the size of the structuring element grows larger. In order to resolve this problem, the existing vHGW CUDA implementation for grayscale images is modified and applied to the case of bitmapped binary images. However, for bitmapped binary images, the vHGW algorithm is not suitable for implementing the horizontal morphological operation. Hence, the vHGW algorithm is only implemented for the vertical morphological operation.

For the implementation of the vertical vHGW algorithm, the image columns are divided into multiple vertical segments and each CUDA thread is responsible for the processing of one single segment. Since the limited memory space of the shared memory poses a restriction on the size of the structuring element, both the suffix and prefix arrays are stored in the global memory of the GPU. Figure 3 illustrates the implementation of the vertical vHGW algorithm using CUDA.

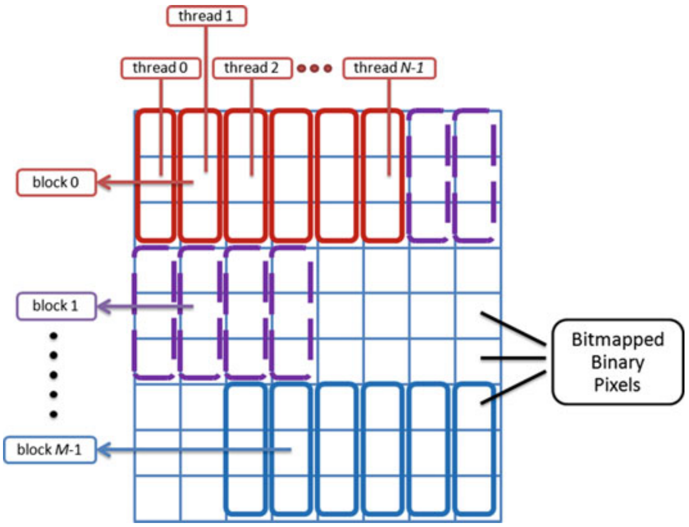

Fig. 3 Parallel implementation of the vertical vHGW algorithm in the GPU using CUDA

4 Results and Discussions

Since the erosion and dilation operations have similar execution steps, the computation time of the erosion operation is similar to the dilation operation. Hence, only the CUDA implementations of the erosion operation are benchmarked. All of the CUDA implementations are benchmarked on a server, which has two Intel Xeon E5-2650 v5 processors and one single Nvidia Tesla K20m graphic card.

4.1 Horizontal Erosion Operation

In this research work, the bitmap representation is incorporated into the implementation of the horizontal erosion operation on binary images. By using the bitmap representation, a 32-bit GPU core is able to process 32 binary pixels in parallel. Furthermore, the number of memory transfer operations required is also reduced substantially. For a horizontal structuring element with a size less than 65 pixels, only 0.125 memory access is needed for each output pixel in the image (four memory accesses for each 32-bit chunk), which is significantly lower than the $7-(4/p)$ memory accesses per pixel for the vHGW algorithm. Since memory access is a very expensive process in the context of the GPU [1], reducing the number of memory transfer operations will definitely improve the performance of the implementation.

The performance of the proposed implementation is compared with the CUDA 6.5 NPP library, OpenCV 3.0 CUDA library and vHGW CUDA implementation [5]. Although the CUDA NPP library, OpenCV library and vHGW implementation are designed to handle grayscale images, it is also possible for these implementations to process binary images by using only two grayscale values (0 and 255) to represent a binary pixel. The performance comparison results for processing a FHD binary image are shown in Fig. 4. For a better representation, the results obtained are illustrated in a log graph.

From Fig. 4, it is observed that the computation time of the proposed CUDA implementation is significantly shorter than the other implementations. Hence, it can process binary images at a faster rate.

For the horizontal erosion operation on bitmapped binary images, each pixel location is read multiple times. Hence, it makes sense to cache the data in the shared memory of the GPU to reduce the memory accesses to the global memory. The results in Fig. 5 compare the performance of the CUDA implementation that uses the shared memory of the GPU to the one that does not use the shared memory. From the graph, it can be observed that both implementations offer similar performance. This means that the usage of shared memory does not offer any further improvement to the proposed implementation. This behaviour is due to the fact that the memory access latency to the global memory is hidden by the L1 and L2 caches of the GPU. Hence, the bottleneck of the algorithm is not on the memory transfer

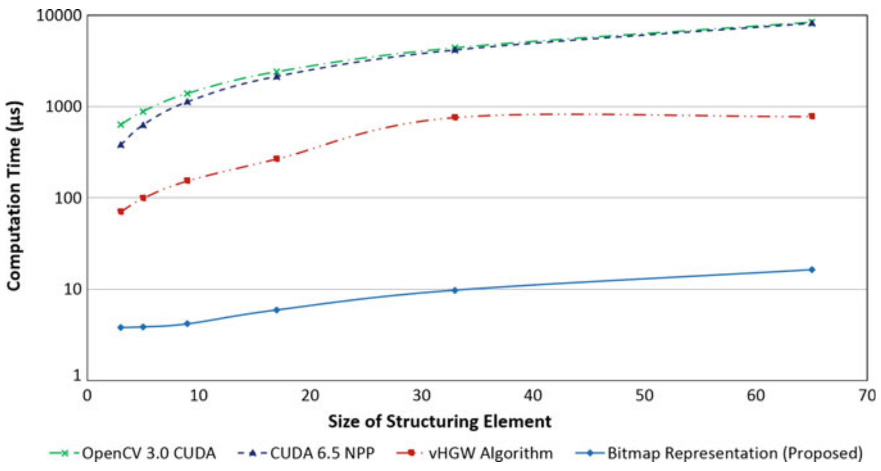

Fig. 4 Performance comparison of different algorithms for the horizontal erosion operation on a FHD binary image

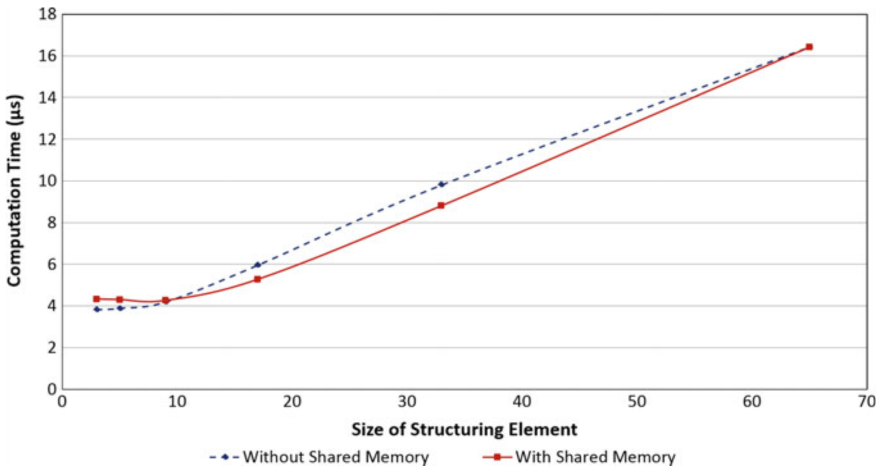

Fig. 5 Computation time of the CUDA implementation with shared memory versus without shared memory

part. In fact, the analysis results using Nvidia Visual Profiler show that the utilisation level of the processing cores is higher than the utilisation level of the memory. Thus, the performance of the implementation is limited by the workload of the processing cores in the GPU rather than the bandwidth of the memory.

4.2 Vertical Erosion Operation

Since most of the binary images are stored in the row-major format in the memory, different memory access patterns have different impacts on the performance of the algorithm. As a result, the vertical erosion operation exhibits different characteristics compared to the horizontal erosion operation. For the vertical erosion operation, the performance of the proposed CUDA implementation that incorporates the bitmap representation is compared with the existing techniques and the results are shown in Fig. 6. The results obtained are illustrated in a log graph to have a better visualisation of the data.

From Fig. 6, the computation time of the proposed technique is significantly faster than the external libraries and vHGW implementation, because the bitmap representation enables one single GPU core to process 32 binary pixels in one shot. At the same time, it is observed that as the size of the structuring element becomes larger, the computation time of the proposed implementation increases at a faster rate compared to the vHGW algorithm. Hence, it is reasonable to modify the vHGW algorithm for grayscale images and apply it to the case of bitmapped binary images. Figure 7 compares the performance of the implementation that incorporates the combination of the bitmap representation and vHGW algorithm to the implementation that utilises only the bitmap representation. From Fig. 7, it is concluded that the implementation that incorporates both the bitmap representation and vHGW algorithm produces a better result.

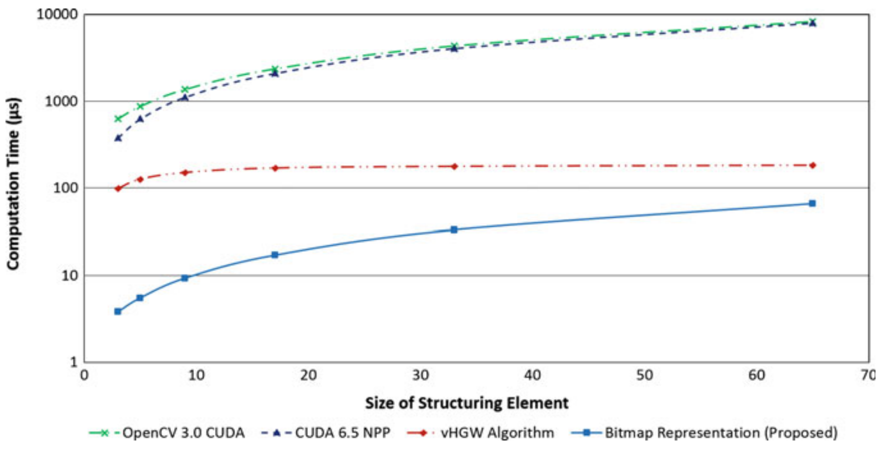

Fig. 6 Performance comparison of different algorithms for the vertical erosion operation on a FHD binary image

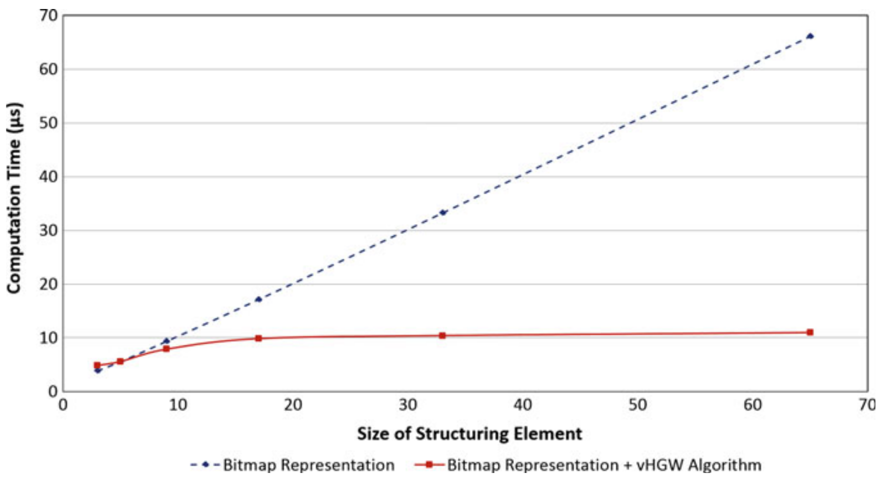

Fig. 7 Performance comparison of the CUDA implementation that incorporates the bitmap representation only with the implementation that incorporates both the bitmap representation and vHGW algorithm

5 Conclusion

Although there are quite a few CUDA implementations for morphology, most of the implementations were designed to process only grayscale images. Hence, we propose two optimisation techniques that make use of the bi-level nature of a binary pixel to further optimise the performance of the CUDA implementation for binary images. One of the optimisation techniques is achieved by representing the binary image in the form of a bitmap. With this optimisation technique, the one-dimensional horizontal erosion operation achieves a speedup up to 70 times, as compared to the existing vHGW CUDA implementation. In addition to the bitmapping of binary images, the existing vHGW algorithm for grayscale images is modified and applied to the case of binary images. With the combination of the bitmap representation and vHGW algorithm, the optimised version of the vertical erosion operation is approximately 20 times faster than the existing vHGW CUDA implementation.

Acknowledgements This project is supported by MOSTI, Malaysia under the e-science funding with a grant number of 010304SF0062.

References

1. Cook S (2012) CUDA programming: A developer's guide to parallel computing with GPUs. Newnes
2. Domanski L, Vallotton P, Wang D (2009) Parallel van herk/gil-werman image morphology on GPUs using cuda. In: GTC 2009 conference posters
3. Gil J, Werman M (1993) Computing 2-d min, median, and max filters. IEEE Trans Pattern Anal Mach Intell 15(5):504–507
4. Solomon C, Breckon T (2011) Fundamentals of digital image processing: A practical approach with examples in Matlab. John Wiley & Sons
5. Thurley MJ, Danell V (2012) Fast morphological image processing open-source extensions for GPU processing with cuda. IEEE J Sel Top Sign Proces 6(7):849–855
6. Van Den Boomgaard R, Van Balen R (1992) Methods for fast morphological image transforms using bitmapped binary images CVGIP. Graph Models Image Process 54(3):252–258
7. Van Herk M (1992) A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. Pattern Recogn Lett 13(7):517–521