Non-maximum Suppression Using fewer than Two Comparisons per Pixel

Tuan Q. Pham

Canon Information Systems Research Australia (CiSRA)
1 Thomas Holt drive, North Ryde, NSW 2113, Australia
tuan.pham@cisra.canon.com.au
http://www.cisra.com.au

Abstract. Non-Maximum Suppression (NMS) is the task of finding all local maxima in an image. This is often solved using gray-scale image dilation, which requires at least 6 comparisons per pixel in 2-D. We present two solutions that use fewer than 2 comparisons per pixel with little memory overhead. The first algorithm locates 1-D peaks along the image's scan-line and compares each of these peaks against its 2-D neighborhood in a spiral scan order. The second algorithm selects local maximum candidates from the maxima of non-overlapping blocks of one-fourth the neighborhood size. Both algorithms run considerably faster than current best methods in the literature when applied to feature point detection. Matlab code of the proposed algorithms is provided for evaluation purposes.

1 Introduction

The term 'non-maximum suppression' first appeared in an edge detection context as a method to reduce thick edge responses to thin lines [1]. This type of directional NMS operates one-dimensionally (1-D) perpendicular to the edge. Kitchen and Rosenfeld [2] extended the concept to isotropic NMS to locate two-dimensional (2-D) feature points from an image. The feature points are selected as local maxima of a 'cornerness' image over some neighborhood. This NMS approach to corner detection was subsequently adopted by many interest point detectors [3–5]. This paper deals with the latter, isotropic, definition of NMS. We target efficient algorithms that require little extra memory use.

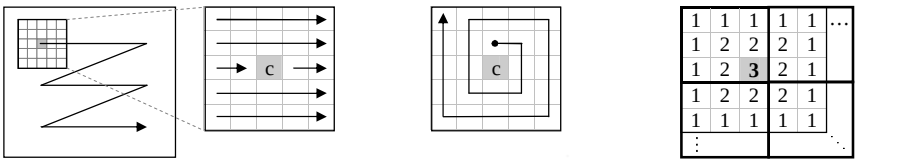
There are known NMS algorithms that require a fixed number of comparisons per pixel regardless of the suppression neighborhood size. One-dimensional max filter, for example, requires three comparisons per pixel [6, 7, 13]. By separable implementation, a two-dimensional max filter can be realized from only six comparisons per pixel. Recently, Neubeck and Van Gool (2006) [8] proposed a block-partitioning algorithm that reduces the number of comparisons down to 2.39 per pixel. However, they were unaware of a simple algorithm dated 20 years back by Förstner and Gülch (1987) [9] that has a similar level of computational complexity. Yet, all of these prior art methods require more than two comparisons per pixel. In this paper, we extend the algorithms in [8] and [9] to reduce the number of comparisons to fewer than two comparisons per pixel.

The remainder of the paper is organized as follows. Section 2 describes three previous NMS methods in the literature. Section 3 presents our scan-line algorithm for 3×3-neighborhood NMS. Section 4 extends the 3×3 algorithm to handle $(2n+1)\times(2n+1)$ neighborhood. It also presents a second solution for general 2-D NMS using block partitioning. Section 5 compares our algorithms with the prior art methods in terms of computational complexity and runtime under Matlab. A coarse-to-fine extension of our algorithms is presented in Section 6. Finally, Matlab code for the proposed algorithms is provided in the Appendix.

2 Previous Solutions

A straightforward approach to NMS over a rectangular neighborhood is described in Figure 1a. The input image pixels are visited in a raster scan order (from left to right, then from top to bottom). Each visited pixel is compared to other pixels in its neighborhood (5×5 in Figure 1) also in a raster scan order. The central pixel c is a non-maximum if a larger or equal neighbor is found. The algorithm then skips to the next pixel in the scan line.

The straightforward method is simple to implement but it can take a long time to process an image. For example, under a worst-case scenario of an increasing intensity trend that runs along the raster order, the straightforward method requires $\lceil (2n+1)^2/2 \rceil$[†] comparisons per pixel for a $(2n+1) \times (2n+1)$ neighborhood. A best-case scenario happens when the intensity trend is reversed. The straightforward method then requires only one comparison per pixel. On average, however, the straightforward method requires $O(n)$ comparisons per pixel (according to the experimental result in Figure 7a).



(a) Raster scan order (b) Spiral order [9] (c) Block partitioning [8]

Fig. 1. Previous solutions for 2-D non-maximum suppression (5×5 neighborhood).

It turns out that the complexity of a raster scan algorithm can be significantly reduced by visiting the neighboring pixels in a different order. Förstner and Gülch [9] presented such an algorithm with a local spiral order as shown in Figure 1b. By comparing with closer neighbors first, the central pixel is guaranteed to be a 3×3-neighborhood local maximum before it is tested against a larger neighborhood. Because the number of 3×3 local maxima in an image is usually small ($\leq 25\%$ of the total number of pixels), the spiral order algorithm

[†] $\lceil x \rceil$ and $\lfloor x \rfloor$ rounds x to the nearest integer towards $+\infty$ and $-\infty$, respectively.

quickly finds any non-maximum and skips to the next pixel. The number of $(2n+1) \times (2n+1)$ local maxima also decreases rapidly as the neighborhood size increases. As a result, the computational complexity of this algorithm is roughly constant (≤ 5 comparisons per pixel to detect a 3×3 non-maximum) irrespective of the neighborhood size.

Recently, Neubeck and Van Gool [8] presented an efficient NMS algorithm that requires 2.39 comparisons per pixel on average and 4 comparisons per pixel in the worst-case. They observed that the maximum pixel of a $(2n+1) \times (2n+1)$ neighborhood is also the maximum of any $(n+1) \times (n+1)$ window that encloses the pixel. The input image is partitioned into non-overlapping blocks of size $(n+1) \times (n+1)$, and the local maximum of each block is detected (Figure 1c illustrates this for $n = 2$). The block maximum is then tested against its $(2n+1) \times (2n+1)$ neighborhood minus the enclosing $(n+1) \times (n+1)$ block. Using only one comparison per pixel, the block partitioning step reduces the number of local maximum candidates by a factor of $(n+1)^2$. As a result, the Neubeck method is quite efficient for large neighborhood sizes.

For small to medium neighborhood sizes, the Neubeck method still has to process a large number of candidates: $MN/(n+1)^2$ for an $M \times N$ image. Each of these candidates requires a maximum of $(2n+1)^2-(n+1)^2-1$ comparisons with its neighbors. Although Neubeck and Van Gool have a solution to reduce the number of extra comparisons per candidate down to $2 + O(1/n)$, their solution increases the algorithm complexity and memory use significantly. We observe that when a block's candidate is not a $(2n+1)\times(2n+1)$-neighborhood maximum, most of the time it lies on the block boundary. This happens when the block's candidate lies on an intensity trend that extends into an adjacent block. A quick way to reject this candidate is therefore to check if it lies on a block boundary first, and if it does, compare it against three immediate neighbors on the adjacent block. This simple trick achieves a similar computational complexity as the improvement in Neubeck's method without using the extra memory.

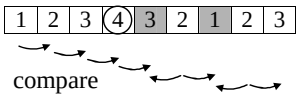
3 3×3-Neighborhood Non-maximum Suppression

NMS for a 3×3 neighborhood is often solved by mathematical morphology [10], whereby the input image is compared against its gray-scale dilated version. Pixels where the two images are equal correspond to the local maxima. However, mathematical morphology does not return strict local maxima, where the center pixel is strictly greater than all neighboring pixels. Morphology is also inefficient. Even a separable implementation of a 3×3-neighborhood gray-scale image dilation requires six comparisons per pixel [6, 7].

In this section, we propose a scan-line algorithm for 3×3 NMS that requires at most 2 comparisons per pixel. The algorithm first searches for 1-D local maxima along the scan line. Each scan-line maximum is then compared against its neighbors in adjacent rows. A rolling buffer of two binary masks is kept for a current and a next scan line. As a new central pixel is processed, its future

neighbors (i.e. unvisited neighbors) are masked out if they are smaller than the central pixel. Masked pixels will be skipped when it is their turns for processing.

Figure 2a illustrates part of our algorithm to find 1-D local maxima along the scan-line [8]. The arrow from one pixel to another denotes a comparison between the central pixel and its neighbor. At the beginning of the sequence [1, 2, 3, 4, 3, 2, 1, 2, 3], the intensities are on an increasing trend. As a result, only one forward comparison is needed before the current pixel is marked as a non-maximum. When the increasing trend stops at a local maximum (*circled* in Figure 2a), its right neighbor is masked out (shaded). The scan-line algorithm then continues with the next unmasked pixel which has a of value 2 in Figure 2a. This time, both a forward and a backward comparison are needed before the pixel is ruled out as a candidate. Again, the right neighbor of value 1 is masked out after being found smaller than the central pixel. In general, every second pixel on a decreasing trend is masked out. This 1-D non maximum suppression algorithm therefore requires one comparison per pixel.



(a) 1D non-maximum suppression

1	2	3
4		5
6	7	8

(b) raster scan order

6	7	8
2		1
3	4	5

(c) our scan order

Fig. 2. 3-neighborhood non-maximum suppression and 3×3-neighborhood scan order.

When extending the 1-D NMS algorithm in [8] to 2-D, we found that the order in which the six neighbors on different rows are compared to the central pixel significantly influences the efficiency of the algorithm. A conventional method traverses a 3×3 neighborhood along a raster scan order as shown in Figure 2b. This results in a maximum of five comparisons per output pixel in a worst-case scenario of an intensity trend that runs along the raster scan. In this scenario, the central pixel is always greater than its four preceding neighbors. Hence, a fifth comparison with the immediate neighbor on the right is needed to reject the central pixel.

In our scan-line algorithm, the future neighbors on the row below are compared first, followed by the past neighbors on the row above (visit order shown in Figure 2c). worst-case scenario happens when the three future neighbors are smaller than the central pixel. However, this does not happen to every pixel in the image because the central pixel must be a strict maximum along the current scan-line. Furthermore, future neighbors can be masked out if found to be smaller than the current pixel. Masked out pixels will be skipped in the future, which saves computations. Similar to the 1-D NMS in Figure 2a, this pixel skipping trick helps reduce the effective cost of this inter-row comparison to one comparison per pixel. As a result, our 3×3-neighborhood NMS requires at most two comparisons per pixel.

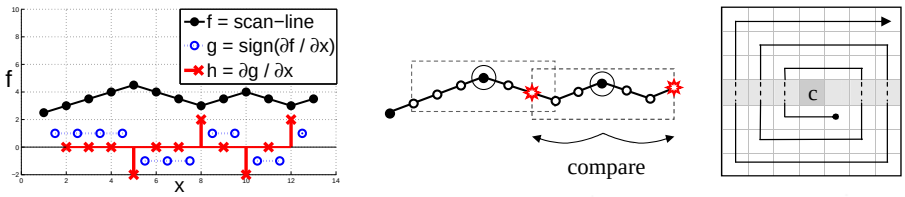
Another reason to visit the future neighbors before the past neighbors is: because the current pixel was not masked out, it is either higher than the past neighbors, or the past neighbors were not processed. In the first case, comparison with the past neighbors is superfluous without triggering an early non-maximum detection. In the second case, since the past neighbors were previously skipped, they are likely to be small, which again reduces the chance of an early non-maximum detection of the current pixel.

4 (2n+1)²-Neighborhood Non-maximum Suppression

Two algorithms for $(2n+1) \times (2n+1)$-neighborhood NMS are presented. The first algorithm extends the scan-line algorithm in Section 3, which use the spiral traverse order from Förstner and Gülch [9]. The second algorithm extends the block partitioning algorithm of Neubeck and Van Gool [8]. Both algorithms use fewer than two comparisons per pixel.

4.1 Scan-Line Algorithm

The scan-line algorithm for 3×3-neighborhoods in Section 3 can be extended to handle $(2n+1) \times (2n+1)$ neighborhoods for $n \geq 1$. First, $(2n+1)$-neighborhood maxima on the current scan-line are located (*circled pixels* in Figure 3b). These 1-D maxima serve as candidates for the 2-D maxima. Each candidate is then compared against its $(2n+1) \times (2n+1)$ neighborhood in a spiral order similar to that of Foerstner's method. Note that the neighbors on the same scan-line have already been compared and can therefore be skipped (*gray pixels* in Figure 3c). This results in a maximum of $2n(2n+1)$ neighbors to be compared per candidate. In practice, the average number of comparisons per candidate is much smaller thanks to the spiral traverse order (explained in Section 2).



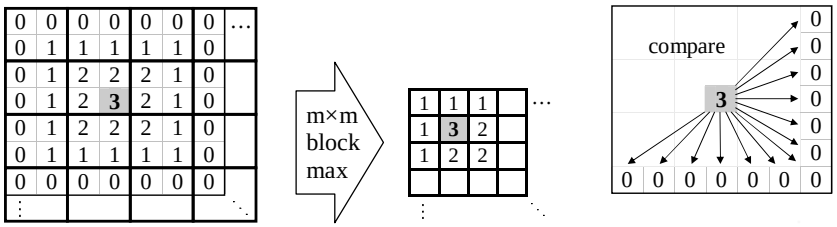
(a) 1D peak and trough detection (b) 1D non-max suppression (c) spiral traverse

Fig. 3. Scan-line algorithm for $(2n+1) \times (2n+1)$ non-maximum suppression ($n = 3$). In Figure 3b, *dashed windows* are the local neighborhoods of the *circled pixels*; *hollow pixels* are on a downward slope from the corresponding circled pixel, hence need not be re-compared; *star pixels*, on the other hand, are to be compared against the left circled pixel. In Figure 3c, *gray pixels* need not be re-compared against the central pixel c.

The detection of the $(2n+1)$-neighborhood maxima on a 1-D scan-line f is shown in detail in Figure 3a. If g is the sign of the finite difference of f, g is either -1, 0 or 1 depending on the local slope of f. g's finite difference h therefore equals -2 at local peaks, +2 at local troughs and 0 elsewhere. 1-D peak and trough detection therefore requires only one comparison per pixel. Next, each 1-D peak is compared against its $(2n+1)$-neighborhood with the knowledge of the extremum detector h. Neighboring pixels which are on a consecutive downward slope from the local peak, i.e. $\{x|h(x)=0\}$, are by definition smaller than the current peak, hence need not be re-compared. Only pixels outside the enclosing troughs of the current peak need extra comparison. The number of extra comparisons to obtain $(2n+1)$-neighborhood maxima from an initial list of 3-neighborhood maxima is therefore very small for a smooth function f.

4.2 Quarter-Block Partitioning Algorithm

As described earlier in Section 2, Neubeck and Van Gool [8] partition an image into non-overlapping blocks of size $(n+1)\times(n+1)$ and use these blocks' maxima as candidates for the $(2n+1)\times(2n+1)$-neighborhood maximum detection. If this is loosely referred to as a half-block partitioning algorithm (because Neubeck's block size roughly equals half the neighborhood size), then a quarter-block partitioning algorithm can be described with reference to Figure 4 as follows. The $M\times N$ image in Figure 4a is partitioned into non-overlapping blocks of size $m\times m$ where $m=\lfloor(n+1)/2\rfloor$. The maximum of each block is extracted to form a $\lfloor M/m\rfloor\times\lfloor N/m\rfloor$ block maximum image in Figure 4b. 3×3 local maxima of this block maximum image are selected as candidates for the $(2n+1)\times(2n+1)$-neighborhood maxima detection (*gray pixel* in Figure 4b). Each candidate is finally compared to its $(2n+1)\times(2n+1)$ neighborhood minus the $3m\times3m$ neighboring pixels already compared during the block-partitioning step (Figure 4c).



(a) input image with overlaid $m\times m$ blocks (b) block maximum image (c) comparison with the rest of the neighborhood

Fig. 4. Quarter-block partitioning algorithm for $(2n+1)^2$ NMS ($n=3$, $m=2$).

It can be shown that a $(2n+1)\times(2n+1)$ window always fully encloses 3×3 partitioned blocks of size $m\times m$ each. A $(2n+1)\times(2n+1)$-neighborhood

maximum therefore is also a 3×3 local peak in the block maximum image. The quarter-block partitioning algorithm avoids Neubeck's inefficiency problem of having many false candidates lying on the block boundaries. Each of our candidates, on the other hand, lies well within the middle of a $3m \times 3m$ neighborhood. The number of candidates of our method is therefore significantly smaller than that of Neubeck. This is a key reason why our quarter-block method is considerably faster Neubeck's. Our algorithm also requires fewer comparisons to evaluate the full neighborhood of a candidate: $(2n+1)^2 - 9\lfloor (n+1)/2 \rfloor^2$ compared to $(2n+1)^2 - (n+1)^2$.

5 Results

5.1 3×3-Neighborhood Non-maximum Suppression

We compared our 3×3 NMS algorithm in section 3 with two other algorithms: the straightforward and the gray-scale morphology algorithm [10]. Both ours and the straightforward method were implemented in *non-vectorized* Matlab code (see the Appendix). Matlab ®Image Processing Toolbox (R2008a) function `imregionalmax` was used for the morphology implementation.

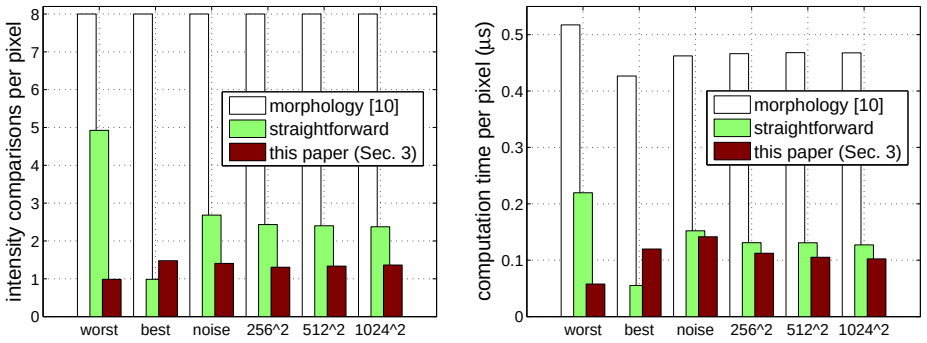
Six gray-scale images were used in this experiment: `worst` and `best` are 256×256 images of worst and best-case scenarios for the straightforward algorithm (see section 2 for description). `noise` is a 256×256 image of uniformly distributed noise. `256^2`, `512^2` and `1024^2` refer respectively to Harris cornerness [3] images of the `Cameraman`, `Lena` and `Pentagon` square image of the corresponding size. We used the following variant of the Harris cornerness function that does not require a tuning parameter:

$$C = \frac{\sum\limits_S I_x^2 \sum\limits_S I_y^2 - \left(\sum\limits_S I_x I_y\right)^2}{\sum\limits_S I_x^2 + \sum\limits_S I_y^2 + \varepsilon} \tag{1}$$

where I_x and I_y are Gaussian gradients ($\sigma = 1$) of the input image I along the $x-$ and $y-$direction, respectively; $\sum\limits_S$ is an average operator over a neighborhood S, which is a 11×11-pixel window in our experiments; and $\varepsilon = 2^{-52}$ is a small offset added to the denominator to avoid division by zero in flat image areas.

The number of intensity comparisons and the average runtime over 100 executions were recorded for each test image on an Intel 3.2 GHz system with 3.25 GB of RAM, the result of which is shown in Figure 5.

Figure 5a shows that our 3×3-neighborhood NMS method requires between 1 to 1.5 comparisons per input pixel. The complexity of the straightforward method varies more widely, ranging from one comparison in its best-case to five comparisons in its worst-case with an average of 2.5 comparisons per pixel. The morphology algorithm requires the most comparisons at eight per pixel. Note that the worst-case scenario of the straightforward method is actually the best-case scenario for ours and vice versa.



(a) Number of comparisons per pixel (b) Computation time per pixel

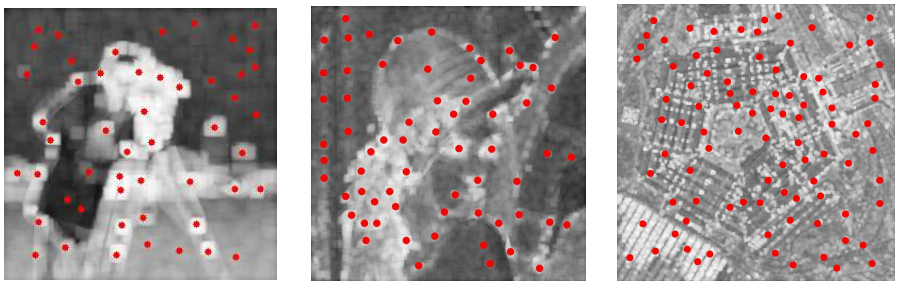
Fig. 5. Comparison of different algorithms for 3×3 non-maximum suppression.

The execution time shown in Figure 5b is generally in agreement with the algorithm complexity in Figure 5a. On average, our method runs at a speed of 0.1μs per pixel. This is 25% faster than the straightforward method. The straightforward method is only faster than ours in its best-case scenario. When compared to morphology, our algorithm implemented in pure Matlab code is almost five-time faster than Matlab's built-in function `imregionalmax` written in C++. This demonstrates that simple *non-vectorised* Matlab code can run as fast as C code with the help of Matlab Just-In-Time (JIT) compiler. The execution time of our algorithm is quite stable over a wide range of image contents and sizes. It is also independent of the number of detected local maxima.

5.2 (2n+1)×(2n+1)-Neighborhood Non-maximum Suppression

We compared two of our 2-D NMS algorithms in Section 4 with three prior art methods: straightforward, Förstner [9] and Neubeck [8]. The algorithms were used to detect Harris corners [3] from three images: `Cameraman` (256×256), `Lena` (512×512) and `Pentagon` (1024×1024). Examples of corners found from these images using different neighborhood size n are displayed in Figure 6. Larger neighborhood sizes were used for larger images to improve the corner separation. We have also tested the robustness of our algorithms by shifting the input images by several pixels in each direction and obtained the same set of corners. This confirms that our implementations of all NMS algorithms work correctly as expected.

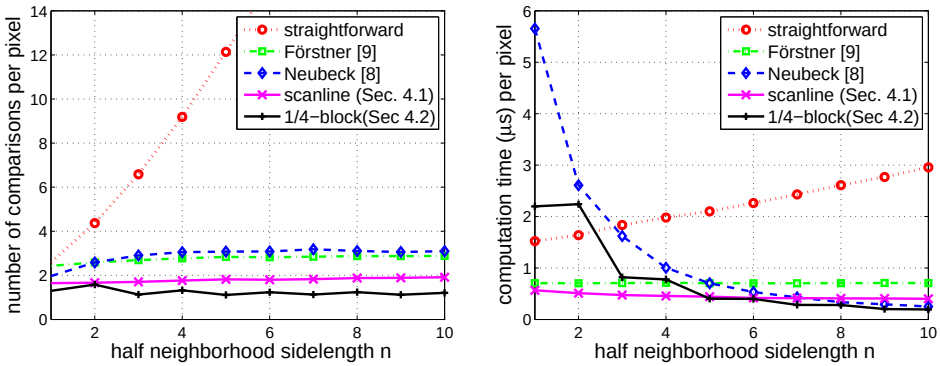
To compare the complexity of different NMS algorithms, the number of pairwise intensity comparisons were counted for each test image. These numbers are then normalized by the number of pixels in each input image. Their average is finally plotted in Figure 7a for different values of neighborhood sidelength n. Figure 7a shows that the complexity of the straightforward method is linearly proportional to the neighborhood sidelength n. All other methods, however,



(a) Cameraman (n=10) (b) Lena (n=20) (c) Pentagon (n=40)

Fig. 6. Results of non-maximum suppression on several Harris cornerness images.

require a similar number of comparisons per pixel irrespective of the neighborhood size. Our scan-line method in Section 4.1, being an improvement of the Förstner method, requires 40% fewer comparisons than the Förstner method. Our quarter-block partitioning method in Section 4.2, being an improvement of the Neubeck method, requires 60% fewer comparisons than its predecessor. Both our methods use fewer than two comparisons per pixel. The quarter-block method is slightly cheaper than the scan-line method.



(a) Number of comparisons per pixel (b) Computation time per pixel

Fig. 7. Performance of $(2n+1)\times(2n+1)$-neighborhood non-maximum suppression.

Average runtime of these five different algorithms on three test images are plotted in Figure 7b. The runtime of the straightforward method increases linearly with the neighborhood sidelength n as expected. For other methods, however, the run time does not always correspond to the complexity in Figure 7a. For small neighborhood sidelength ($n \leq 2$), the block partitioning methods (Neubeck and ours in Section 4.2) are even slower than the straightforward methods. This

is because the block size is too small for any significant candidate pruning. The scan-line methods (Förstner and ours in Section 4.1), on the other hand, show stable runtime over a large range of neighborhood sizes. Both scan-line methods need less than 1 microsecond to process one pixel. Our scan-line algorithm is on average 60% faster than Förstner algorithm. As the neighborhood size gets larger ($n \geq 5$), the block partitioning methods start showing advantage over the scan-line methods. Our quarter-block partitioning algorithm is on average 40% faster than Neubeck algorithm.

6 Extensions and Applications

NMS algorithms usually require at least one comparison per pixel to process an image. This is because each pixel needs be compared to at least one larger neighbor before it is ruled out as a non-maximum. Even with our fastest NMS algorithms in Section 4, it still takes an order of one second to process a mega-pixel image in Matlab. Further speedup is therefore desirable.

Fortunately, NMS can be applied to a down-sampled image without substantial loss of detection accuracy. This is especially true when the neighborhood size is large and the input image is smooth. Harris corner detection, for example, often starts with a cornerness image that was previously low-pass filtered. NMS on a low-resolution image (e.g. by pixel binning) produces a small number of candidates. These candidates can then be compared against their full neighborhood in the original image.

Both algorithms described in this paper are extendable to N-D. The scan-line algorithm in Section 4.1 can be applied recursively: local maxima of an i-D slice of the input image over the first i dimensions are candidates to the $(i+1)$-D local maxima over the first $(i+1)$ dimensions. The block partitioning algorithm in Section 4.2 is dimension-dependent because an N-D image can be divided into a grid of N-D blocks.

NMS can be used to find distinctive feature points in an image. To improve the repeatability of a detected corner across multiple images, the corner is often selected as a local maximum whose cornerness is significantly higher than the close-by second highest peak [11]. NMS with a small neighborhood size can produce an oversupply of initial peaks. These peaks are then compared with other peaks in a larger neighborhood to retain strong ones only. The initial peaks can be sorted during NMS to facilitates the later pruning step.

For some applications such as multi-view image matching, an evenly distributed set of interest points for matching is desirable. An oversupplied set of NMS point features can be given to an adaptive non-maximal suppression process [12], which reduces cluttered corners to improve their spatial distribution.

7 Conclusion

We have described two new algorithms for non-maximum suppression of two-dimensional images. The scan-line algorithm improves upon a previous method

by Förstner and Gülch [9] and it is 60% faster than [9]. The quarter-block partitioning algorithm improves upon the current best method in the prior art by Neubeck and Van Gool [8] and it is 40% faster than [8]. The scan-line method is the preferred method for small neighborhood sizes ($n < 5$), while the quarter-block algorithm runs faster for larger neighborhood sizes ($n \geq 5$). Both of our algorithms require fewer than two comparisons per input pixel while other algorithms in the literature require more than two. Runtime under Matlab on a 3.2 GHz Intel system consistently took less than 0.5 microsecond per pixel. These fast NMS algorithms require little memory overhead, which is desirable for many computer visions tasks such as object and corner detection. Our algorithms can also be extended to handle images of any dimension.

References

1. Rosenfeld, A., Kak, A.: Digital Picture Processing. 2 edn. Academic Press (1976)
2. Kitchen, L., Rosenfeld, A.: Gray-level corner detection. Pattern Recognition Letters 1, 92–102 (1982)
3. Harris, C., Stephens, M.: A combined corner and edge detector. In: Proc. of the Fourth Alvey Vision Conference, pp. 147–151 (1988)
4. Lowe, D.: Distinctive image features from scale-invariant keypoints. IJCV 60, 91–110 (2004)
5. Mikolajczyk, K., Schmid, C.: Scale and affine invariant interest point detectors. IJCV 60, 63–86 (2004)
6. van Herk, M.: A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. Pattern Recognition Letters 13, 517–521 (1992)
7. Gil, J., Werman, M.: Computing 2-d min, median, and max filters. IEEE Trans. on PAMI 15, 504–507 (1993)
8. Neubeck, A., Van Gool, L.: Efficient non-maximum suppression. In: Proc. of ICPR, Volume 3, pp. 850–855 (2006)
9. Förstner, W., Gülch, E.: A fast operator for detection and precise locations of distinct points, corners, and centres of circular features. In: Proc. of Intercommission Conf. on Fast Processing of Photogrammetric Data, pp. 281–305 (1987)
10. Soille, P.: Morphological Image Analysis: Principles and Applications. Springer (1999)
11. Schmid, C., Mohr, R., Bauckhage, C.: Evaluation of interest point detectors. IJCV 37, 151–172 (2000)
12. Brown, M., Szeliski, R., Winder, S.: Multi-image matching using multi-scale oriented patches. In: Proc. of CVPR, Volume 1, pp. 510–517 (2005)
13. Coltuc, D., Bolon, P.: Fast computation of rank order statistics. In: Proc. of EUSIPCO, pp. 2425–2428 (2000)

A Matlab code

A.1 3×3 Non-maxima Suppression

Below is a non-vectorized Matlab implementation of the 3×3-neighborhood non-maxima suppression algorithm described in Section 3. This function produces the same output as `imregionalmax` from Matlab Image Processing Toolbox, only faster.

```
function mask = nonmaxsupp3x3(im)

[h w] = size(im);  mask = false([h w]);  % binary output image
skip = false(h,2);  cur = 1;  next = 2;  % scanline masks

for c=2:w-1
   r = 2;
   while r<h
      if skip(r,cur), r=r+1; continue; end  % skip current pixel

      if im(r,c)<=im(r+1,c)    % compare to pixel on the left
         r=r+1;
         while r<h && im(r,c)<=im(r+1,c),  r=r+1; end  % rising
         if r==h, break; end   % reach scanline's local maximum
      else                     % compare to pixel on the right
         if im(r,c)<=im(r-1,c),  r=r+1; continue; end
      end
      skip(r+1,cur) = 1;       % skip next pixel in the scanline

      % compare to 3 future then 3 past neighbors
      if im(r,c)<=im(r-1,c+1),  r=r+1; continue; end
      skip(r-1,next) = 1;      % skip future neighbors only
      if im(r,c)<=im(r  ,c+1),  r=r+1; continue; end
      skip(r  ,next) = 1;
      if im(r,c)<=im(r+1,c+1),  r=r+1; continue; end
      skip(r+1,next) = 1;
      if im(r,c)<=im(r-1,c-1),  r=r+1; continue; end
      if im(r,c)<=im(r  ,c-1),  r=r+1; continue; end
      if im(r,c)<=im(r+1,c-1),  r=r+1; continue; end

      mask(r,c) = 1;  r=r+1;   % a new local maximum is found
   end
   tmp = cur;  cur = next;  next = tmp;   % swap mask indices
   skip(:,next) = 0;           % reset next scanline mask
end
```

A.2 Spiral Indexing of a Local $(2n+1) \times (2n+1)$ Neighborhood

The following function returns the row and column offsets of pixels in a local $(2n+1)\times(2n+1)$ neighborhood when traversed in a spiral order. Due to Matlab's column-major order, the traverse order is slightly different from the row-major spiral order in Figure 1b.

```
function [r,c] = spiralindex(n)
r = 0;  c = 0;  run = 0;
for ii=1:n
   run = run+1;
   dr=-1;  dc= 0;
```

```
   for jj=1:run,  r = [r; r(end)+dr]; c = [c; c(end)+dc]; end
   dr= 0;  dc= 1;
   for jj=1:run,  r = [r; r(end)+dr]; c = [c; c(end)+dc]; end
   run = run+1;
   dr= 1;  dc= 0;
   for jj=1:run,  r = [r; r(end)+dr]; c = [c; c(end)+dc]; end
   dr= 0;  dc=-1;
   for jj=1:run,  r = [r; r(end)+dr]; c = [c; c(end)+dc]; end
end
dr=-1; dc= 0;
for jj=1:run, r = [r; r(end)+dr]; c = [c; c(end)+dc]; end
```

A.3 Scan-Line Algorithm for $(2n+1) \times (2n+1)$ NMS

A non-vectorized Matlab implementation of the scan-line algorithm for $(2n+1)\times(2n+1)$-neighborhood non-maximum suppression (Section 4.1) is given below.

```
function mask = nonmaxsupp_scanline(im,n)

[h w] = size(im);  mask = false([h w]);  % binary output image
scanline = zeros(h,1);  skip = false(h,1);    % scanline mask

[dr,dc] = spiralindex(n);  % index neighborhood in a spiral path
I = find(dc~=0);  dr = dr(I);  dc = dc(I);   % skip current line

for c=n+1:w-n
   scanline = im(:,c);  skip(:) = false;      % current scanline
   resp = [0; diff(sign(diff(scanline))); 0]; % discrete Hessian
   peaks = find(resp(n+1:end-n)==-2) + n;     % peak indices

   for ii=1:length(peaks)
      r = peaks(ii);
      if skip(r), continue; end         % skip current pixel

      curPix = scanline(r);  whoops = false;

      for jj=r+1:r+n, if resp(jj)~=0, break; end; end; %downhill
      for jj=jj:r+n
         if curPix <= scanline(jj), whoops=true; break; end
         skip(jj) = 1;     % skip future pixels if < current one
      end
      if whoops, continue; end      % skip to next scanline peak

      for jj=r-1:-1:r-n, if resp(jj)~=0, break; end; end;
      for jj=jj:-1:r-n
         if curPix <= scanline(jj), whoops=true; break; end
      end
      if whoops, continue; end      % skip to next scanline peak
```

```
      % if reach here, current pixel is a (2n+1)-maximum
      for jj=1:length(I)    % visit neighborhood in spiral order
         if im(r,c)<=im(r+dr(jj),c+dc(jj)),  break; end
      end
      if jj>=length(I) && im(r,c)>im(r+dr(jj),c+dc(jj))
         mask(r,c) = 1;   % a new (2n+1)x(2n+1)-maximum is found
      end
   end
end
```

A.4 $(2n+1) \times (2n+1)$ Quarter-Block Partitioning Algorithm

Below is a vectorized Matlab implementation of the quarter-block partitioning algorithm described in Section 4.2. Note that we actually compare a local maximum candidate with its full $(2n+1) \times (2n+1)$ neighborhood instead of with individual unvisited neighbors because vectorized code runs faster in this case.

```
function mask = nonmaxsupp_quarterblock(im,n)

[h w] = size(im); mask = false([h w]); m = floor((n+1)/2);
hh = floor(h/m);        % hh x ww = number of m x m blocks,
ww = floor(w/m);        % starting from (m,m) offset

%% vectorised code to compute local maxima of m-by-m blocks
val = im(1:hh*m,1:ww*m);
[val,R] = max(reshape(val,[m hh*ww*m]),[],1);
val = reshape(reshape(val,[hh ww*m])',[m ww hh]);
R = reshape(reshape(R  ,[hh ww*m])',[m ww*hh]);  % row indices &
[val,C] = max(val,[],1);        % column indices of local maxima
R = reshape(R(sub2ind([m ww*hh],C(:)',1:ww*hh)),[ww hh])';
val = squeeze(val)';  C = squeeze(C)';

%% compare each candidate to its (2n+1)x(2n+1) neighborhood
mask0 = nonmaxsupp3x3(val);    % local maxima of block max image
for I = find(mask0)'
   [ii,jj] = ind2sub([hh ww],I);
   r = (ii-1)*m + R(ii,jj);
   c = (jj-1)*m + C(ii,jj);
   if r<=n||c<=n||r>h-n||c>w-n, continue; end    % out of bound

   % compare to full (2n+1)x(2n+1) block for code simplicity
   if sum2(im(r+(-n:n),c+(-n:n))>=val(ii,jj))==1,
      mask(r,c) = 1;    % a new (2n+1)x(2n+1)-maximum is found
   end
end
```