IMPLEMENTATION USING THE VAN HERK/GIL-WERMAN ALGORITHM

The van Herk/Gil-Werman (vHGW) algorithm is similar to our fast method for convolution with a flat kernel, where we first computed an accumulation matrix and then used it to quickly generate the convolution for any rectangular kernel. vHGW differs in that we compute a localized running Max (or Min) array that is specific to the size of the linear SE, and we then use the array to compute the result pixels locally (over a length equal to the SE size).

The vHGW algorithm was published by van Herk in "A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels", *Patt. Recog. Letters*, **13**, pp. 517-521, 1992 and by Gil and Werman in "Computing 2-D min, median and max filters", *IEEE Trans PAMI* **15**(5), pp. 504-507, 1993. (There appears to be some priority dispute, so I take a neutral position and give credit to both sets of authors.) This was the first grayscale morphology algorithm to compute dilation and erosion with complexity independent of the size of the SE. It is simple and elegant, and the surprise is that it was only discovered as recently as 1992. It works for SEs composed of horizontal and/or vertical linear elements, and requires not more than 3 pixel value comparisons for each output pixel.

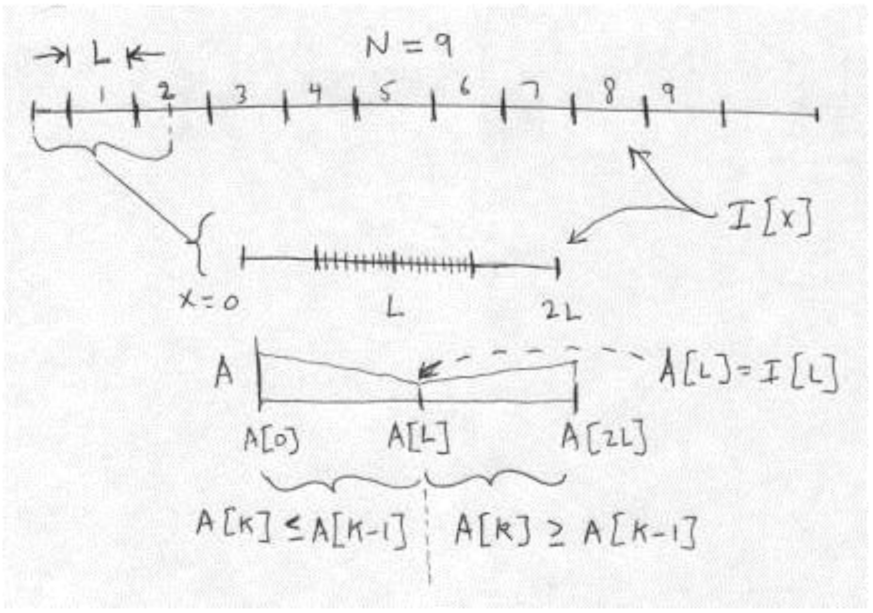
The algorithm has been recently refined by Gil and Kimmel, in "Efficient Dilation, Erosion, Opening and Closing Algorithms," given at ISMM (International Symposium on Mathematical Morphology) 2000, Palo Alto, CA, June 2000, and was published in *Mathematical Morphology and its Applications to Image and Signal Processing*, Kluwer Acad. Pub, pp. 301-310. They bring the number down below 1.5 comparisons per output pixel, at a cost of significantly increased complexity. Consequently, we'll describe and implement the original method.

We describe dilation; for erosion, substitute *Min* for *Max*. In brief, the image is divided into pixel groups of length L, where L, an odd number, is the size of the SE. We describe the case for a horizontal SE with its origin at the center position (L/2). Vertical SEs are handled similarly, except the pixels are selected in vertical groups. On each raster line of length w, we compute output pixels starting at x = L/2 (to avoid boundary effects) and operate on N = (w - 2 * (L/2))/L segments of length L. N takes this particular form to avoid boundary effects on the right side as well, so that the last segment of length L is guaranteed to have the SE entirely within the image at all L points. As a result, we do not evaluate the first L/2 pixels and, in the worst case, the last 3L/2 pixels. To get reasonable results on all pixels in the image, we embed the actual image in a larger image with these augmented dimensions, where the added border pixels are appropriately initialized (to 0 for dilation and to 255 for erosion).

The algorithm proceeds for each group of L pixels in 2 steps. In the first, we form the running Max array; in the second, we evaluate the output pixel values.

For each group of L pixels, we consider a window of 2L+1 pixels in the source image that extends L/2 pixels to either side. The pixel at the center of the L pixels is also at the center of this larger window. Form an array A[] of length 2L+1, consisting of backward and forward running Max pixel values, which are computed starting at the source pixel at the center of the group of L pixels. This array will coincide with the larger window of 2L+1 pixels. Now, consider the very first group in the raster. The larger window starts at the left edge (x = 0) and proceeds to x = 2L. The center value of the Max array, A[L], is given by the pixel at x = L. The value to the left of that, A[L-1], is given by the Max of the values of the pixel at x = L and the pixel at x = L-1, and so on progressively to the value at the beginning of the array, A[0], which is the Max of all pixels from x = L back to x = 0. The array values to the right of A[L] are likewise found by taking the Max of all pixels from x = L up to that point, progressing finally to the value at the end of the array, A[2L], which is the Max of all pixels from x = L to x = 2L. In all, this step requires computing 2(L-1) Max functions.

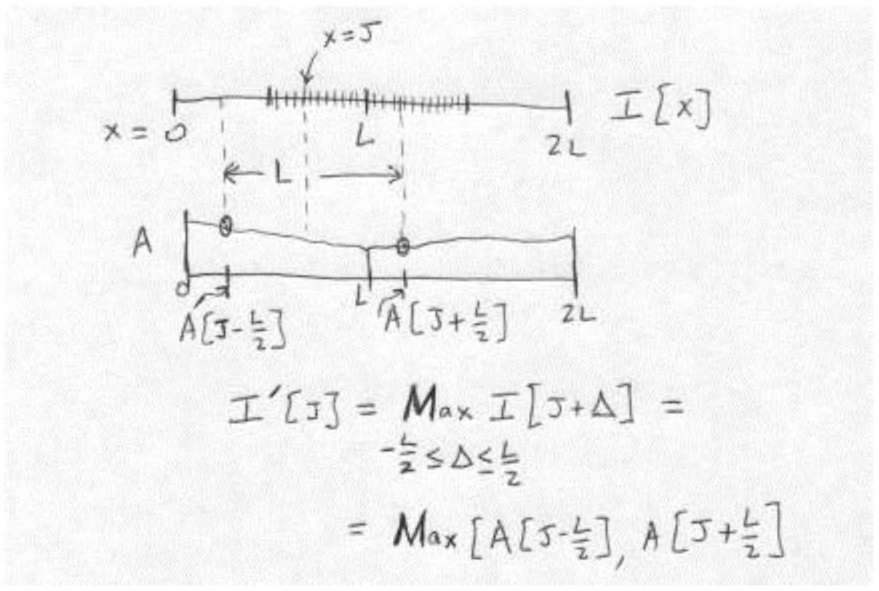
The generation of the running Max array A[] from the source image I is shown below. We display the domain over which a raster line of the image I is defined, but we do not display I[x] itself. As shown, there are N=9 pixel groups of length L for which output pixels can be computed. We magnify the first interval of length 2L+1 in I. This covers the relevant pixels in I that are used for generating the first pixel group of L output pixels in I':



Once this array is formed, the values of the dilated pixels, that are written to the destination image, can easily be read off. For this first interval of length L, the L output image pixels from x = L/2 to x = 3L/2 are computed by imagining that the SE is placed on the array A with its center at one of these L locations; namely, at each pixel for which the SE fits entirely within the array A.

We start at x = L/2, where the end points of the SE are at x = 0 and x = L, and take the Max(A[0], A[L]) = A[0]. (We know here that A[0] >= A[L], by the construction of the array A.) Then at x = L/2 + 1, we take Max(A[1], A[L+1]), and so on, up to the last value at x = 3L/2, which is Max(A[L] + A[2L]) = A[2L], again by construction. In all, this step requires computing L-2 Max functions.

The method for computing the destination (dilated) pixel values I'[J] from the source image I, for the first pixel group given by L/2 <= x <= 3L/2, is shown below. The computation of the output pixel at x = J is shown explicitly.



$$I'[J] = \underset{-\frac{L}{2} \le \Delta \le \frac{L}{2}}{\mathrm{Max}} I[J+\Delta] = \mathrm{Max}\left[A[J-\tfrac{L}{2}], A[J+\tfrac{L}{2}]\right]$$

The total number of comparisons per output pixel is 3 - 4/L, which is never greater than 3. The calculation described above is repeated at each of the N segments of length L on the raster line.

The implementation is straightforward. We set up two buffers, one to hold an entire raster line (or column, for vertical SEs) of the image, and the second to hold the running Max(Min) array of size 2L+1. The raster line buffer is used to hold the pixels in byte order. For little-endian machines, this is exactly opposite to the standard pixel order for 32-bit words, which puts the pixels in MSB-to-LSB order; i.e., 3, 2, 1, 0. The pixel order in each 32-bit word is reversed as pixels are copied to the buffer. (For big-endian machines, no pixel reshuffling is necessary.) For vertical SEs, we just take the pixels in a column in order from top to bottom. See the source code in graymorphlow.c for other details.

As with binary morphology, opening and closing are defined as a sequence of erosion/dilation and v.v. These operations are idempotent, so one simple test for correctness is to open an image with a SE and then *open the result*. The second opening should give the same image as the original one. Our implementation permits morphological operations with SEs that are linear (horizontal or vertical) and square (implemented as a sequence of horizontal and vertical operations). The speed is independent of the size of the SE and proceeds at about 120 P3 machine cycles per output pixel, for large images.

Source: http://www.leptonica.com/grayscale-morphology.html