G-neighbors

Terrance Boult
Department of Computer Science,
Columbia University, NYC, NY 10027
boult@cs.columbia.edu

Robert A. Melter
Long Island University- Southampton,
Southampton NY

Frank Skorina
Key Technology, Inc. Walla Walla, Washington.

Ivan Stojmenovic
University of Ottawa, Ottawa, Canada.

Abstract

Image processing often involves operations using pixel "neighborhoods". This paper combines the usual definition of 4 or 8 connected neighbors with image information to produce local neighbor definitions that are signal dependent. These generalized neighbors, G-neighbors, can be used in a variety of image processing tasks. The paper examines their use in detail preserving smoothing and morphology. The simple/local nature of G-neighbor definitions make them ideal for implementation on low-level pixel parallel hardware. A near real-time parallel implementation of the G-neighbor computation, including G-neighbor-based detail preserving smoothing and G-neighbor morphology, is discussed. The paper includes a qualitative comparison of G-neighbor-based algorithms to previous work.

1 Introduction and basic definitions

A digital image is often represented by a grid of pixels. The neighbors of a pixel are those pixels that surround it. The grids are generally rectilinear with with 4-connected and 8-connected neighbors. A pixel's 8-connected neighbors are the 8 pixels surrounding it (row and/or column index differing by at most one) and the 4-connected neighborhood are the 4 non-diagonals surrounding that pixel (row XOR column index differing by at most one.) While a few researchers have investigated other neighbor definitions, e.g. 6 and 16 connected, 4 and 8 connected are the dominate ones.

We say that a pixel P_1, with value(P_1)=A, is the *G-neighbor* of another pixel P_2, with value(P_2)= B, if and only if it is a neighbor in the usual sense (4 or 8 or 16 connected) and if A and B satisfies a simple symmetric predicate $Neigh(A, B)$. The simplest neighborhood is one which is spatially constant, i.e. the predicated $Neigh_g(A, B) = True$. This is what is currently used in image processing. This paper explores only a slightly more complex neighbor model – simple yet complex enough to allow spatially varying neighborhoods. In particular the examples in this paper will consider $Neigh_G(A, B) = |A - B| < G$, i.e. the absolute value of the difference between the pixels is less than the *a priori* given threshold G. Figure 1 shows some G-neighbor relationships between pixels.

Another useful neighbor predicate we have briefly explored is $Neigh_M(A, B) = |A - B|/(A + B) < M$. This is useful when one anticipates a noise model which is multiplicative rather than additive. Such predicates may be useful for SAR processing. It is also interesting to allow 8 connected neighbors to include the distance (which is longer on the diagonals) in the predicate $Neigh(A, B)$. This paper concerns itself only with gray-scale images. One can extend the ideas to both multi-spectral images and range images, with appropriate $Neigh(A, B)$ definitions, and most of the following carries over. One can also use an auxiliary "image" for defining the neighbors of a pixel, but we have yet to explore this aspect of G-neighbors.

There are many reasons to use neighborhoods in image processing. One of the biggest reasons is for noise removal. Before we get into the details of G-neighbors, their properties and their parallel implementation, we will provide some examples of their applications to such noise removal problem. We will briefly look at may be considered two different of types of noise: image noise, and "feature noise".

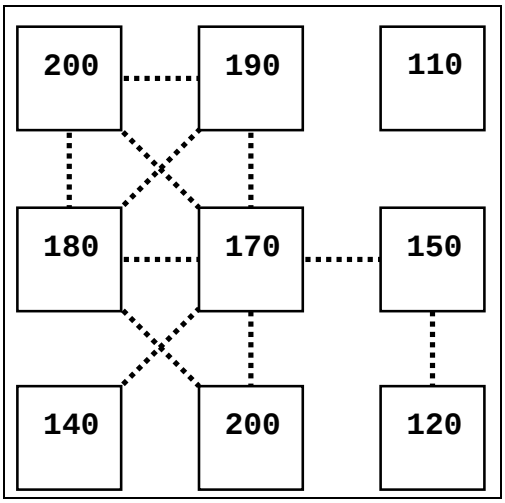

Figure 1: The number inside each pixel represents the intensity value for that pixel. Pixels that are G-neighbors when G=30 are shown (with $Neigh_G$) with dashed lines.

Image noise is a combination of electronic/thermal noise in the camera/digitizers, atmospheric affects, lighting variations, etc. It is common for people to have at least a weak model for type of noise. Many algorithms have been developed for the smoothing of image noise, while attempting to preserve image details. This paper considers, in section 2.1, the application of G-neighbors to detail preserving smoothing.

"Feature noise" refers to the occurrence of "features" in the image which are undesirable. For example, in a binary image being used for an inspection task, "feature noise" might be small regions whose gray-level differ only slightly from the background. We call it "feature noise" because it is a type of noise in the image after a feature detector has been applied defining foreground/background. This noise is generally caused by inexactness in the feature detector combined with image noise. One approach for dealing with feature noise is morphology. Morphology can be applied to either the binary output (foreground/background) or to the original gray image. The former is cheaper to implement and more commonly used. In section 2.2, this paper will touch on the use of G-neighbors for feature-noise reduction. As we shall see, they provide a flexibility / cost tradeoff somewhere between binary and gray-scale morphology.

As we explore these applications we will also comment on relation to previous work, including a qualitative comparison. At one level, the idea of image dependent neighbors is not new. Many researchers have built algorithms that use 4 or 8 connected neighbors where the algorithm uses image dependent "weights" computed from these neighborhoods. One might say these weights effectively determine local neighbors. Some important differences are that image-based "weights" are rarely symmetric, thus they do not induce a neighborhood, and the weights mix the "neighborhood" definition with the desired computation. The use of "real" weights often increases the complexity of implementation and require more tuning.

After describing these applications, we come back to the properties of G-neighbors and then discuss their parallel implementation. Basic code for a serial implementation of G-neighbors, which runs under KBVision, is available from T. Boult. A parallel version using a PIPE is also available.

2 Applications of G-neighbors

A G-neighborhood is a restriction to the definition of a neighborhood. It is natural to perform the same operations on G-neighborhoods as on regular neighborhoods. Whatever operations are to be performed, it is important to note that the operation will only be performed over the connected component; processing is localized within G-components, which are isolated from each other. A common neighborhood operation is noise removal, either image noise, or feature noise and in this section we consider two applications of G-neighbors: detail preserving smoothing and morphology.

2.1 Detail preserving smoothing

The goal of detail preserving smoothing, is to remove image noise while retaining important details in the image. There have been many algorithms which address this problem, e.g. see the surveys/bibliography in [Scher *et al.*-1980], [Chin and Yeh-1983]. G-neighbors were not invented to provide a new and better detail preserving filtering mechanism. The goal of this section is to show that they provide a reasonable one. As we shall see later they do so very cheaply.

Preserving detail while removing noise has an intuitive definition and can be evaluated with a subjective error criterion. Quantitative definitions, such as those in [Chin and Yeh-1983], which presents a quantitative comparison of 7 different techniques, are important. However, their measure does not, in our opinion, capture the effectiveness/quality at preserving small details, exactly the things in which we are interested. It is difficult for a single measure to capture size and shape and amplitude variations which define the "signals" which are to be preserved. In addition, we need to deal with noise models with "fatter tails" than a Gaussian.

While [Chin and Yeh-1983] is a start, it really considers detail to be "edges". For some applications we need measures that are more sensitive to small features and other noise models. In applications using FLIR, SAR, and medical imaging technologies (CAT, MRI, PET) small details (a few pixels), if sufficiently high in contrast, should not be blurred. This paper presents only a qualitative comparison.

2.1.1 Previous work on smoothing

There is a large literature on smoothing, and we can not hope to survey it here, but present a sampling. We classify existing techniques into three main groups:

1. pure statistical based techniques, e.g. mean filtering, median filtering, K-mean filtering,
2. weighted statistical methods, e.g. weights averaging with dependent on local variance,
3. "edge-based" or template-based techniques.

Note that the weighted techniques may use edge response filters (e.g. gradient magnitude). To reduce cost while still providing significant smoothing, the algorithms are often used in an iterative fashion. Of course, since these are generally non-linear methods, multiple applications of a small window not the same as a single application of a larger window.

In the statistical techniques, statistical quantities are computed over the neighborhood and used to update the pixel value. For some statistical techniques, neighborhoods considered are often larger and 4 or 8 connected, e.g. 5x5, 9x9 or larger square regions. One statistical technique is mean-filtering where each pixel is replaced by the mean of a neighborhood. This does remove noise, but also blurs details. Another important and well studied algorithm, is median filtering. In median filtering the current pixel value is replaced with the median value of a local neighborhood, (see [Tukey-1976, Narendra-1981, Huang *et al.*-1979, Gallagher and Wise-1981]). This is noticeably better with respect to preserving detail, when the detail is larger than the median window. It still, however, blurs fine detail, e.g. corners, thin lines, rapidly varying texture.

To make these statistical algorithms more robust and still preserve details, use of only some of the neighbors may be used, or some points may be given more "votes". A useful variation on the mean-algorithm is the K-means algorithm, [Davis and Rosenfeld-1978], where only those K neighbors closest in gray-level are averaged. Another approach is average with the neighbors that is from the same histogram population, [Narayananand and Rosenfeld-1981]. Both of these give image dependent neighborhoods, but they are not symmetric and require more computation. We will compare our G-neighbor methods to one statistical method, median filtering where the central pixel is given multiple votes (3 or 5). We the use the implementation of median filtering from KBVision [AAI-1985]. One could further extend the statistical techniques to use G-neighbors as well as the above extensions. This would increase the power of G-neighbor techniques but also increase their costs.

The second class of detail preserving algorithms use local information to determine weights which are then used in a statistical combination. The combination rule is usually averaging. Again, there are a large number of filters in this category. The weights can be determined from local gradient information (e.g. [Wang *et al.*-1981]), local

second derivative information (e.g. [Graham-1966]), or local mean and variance information (e.g. [Lee-1980] [Lee-1981], [Overton and Weymouth-1979].) The general idea is to weight points that are likely to be on the same "intensity" surface more. Methods differ in how they make that decision. We will also compare our algorithms to one weighted method, that proposed by Weymouth, [Overton and Weymouth-1979], as implemented in KBVision [AAI-1985]. This method has 2 parameters, K and S, and the weights in the neighborhood are given by the formula:

$$\frac{1.0}{dist * (1.0 + K * d^{\frac{S * d^2}{V}})}$$

where *dist* is the distance to the pixel's location (either 1.0 or $\sqrt{2}$), d is the difference between the pixel's value and the central pixel's value, and V is the second variance of all the pixels in the neighborhood. As mentioned before, these weights, if they vary sufficiently, are almost the same as defining G-neighbors. In the qualitative comparison, this approach will be quite competitive with the G-neighbor based technique, but its implementation computation is more expensive and its lack of symmetry can make it less intuitive to understand.

The final class of detail preserving algorithms use local edge estimators (or some template) to determine the location/direction of an edge. Smoothing is then performed only along the edge direction. Difference algorithms in this category, [Nagao and Matsuyama-1979, Tomita and Tsuji-1977, Haralick and Watson-1981] differ mostly in how to determine the edge location/orientation and how to smooth the data. The algorithms within this class can vary significantly in performance. These methods are generally better in high noise situations, where the signal is large homogeneous regions. The facet-based method of [Haralick and Watson-1981] is better when the "signal" contains regions that are well approximated by low order polynomials, but not by a constant gray level. We will compare our G-neighbor smoothing to the Nagao algorithm, [Nagao and Matsuyama-1979], as implemented in KBVision [AAI-1985].

2.1.2 G-smoothing

The G-neighbor smoothing approach is quite simple: apply your favorite smoothing operator, but only using a pixel's G-neighbors. In the examples that follow, we will use the mean (i.e. simple averaging), in a recursive manner. More sophisticated operations, e.g. G-medians, could be used. However, we have found the average sufficient and the effectiveness of the approach with this simple operator suggest the value of the view. Better operators would only make things more effective but occupy other points in the cost/performance tradeoffs.

Because of their definition, the G-neighbor based smoothing is a combination of statistical and edge-based techniques. As described in section 3, G-neighbors can be viewed as defining a super-detailed edge detector. Because the "edges" in the edge-based smoothing are super small, it is possible to retain even single pixel features, if they are sufficiently different from their neighbors. Also, the smoothing is not just along an edge, but rather within all the area on one side of the edge. The signal dependent nature of the neighborhood gives them the ability to apply a simple statistical technique without worrying about blurring points. One can view the G-neighbor definition as providing a combined "noise/edge" model: any two pixels which are separated by less than the prescribed amount are consistent with noise on a single surface, any that differ by more are consistent with a local edge. Still, pixels which are separated from their adjacent pixels may still be affected by that pixel's value if more that one iteration of smoothing is applied. The G-neighborhood operation results in filtering being performed only over connected components. Infinite smoothing would set each pixel in the connected component to the average value in that connected component.

Figures 2–5 shows examples of G-neighbor smoothing, as well as the same examples for the Nagao, and Weymouth smoothing operations. The goal of these examples is not to pronounce one algorithm "superior", but rather to show that G-neighbor smoothing is a reasonable alternative. The examples show the expected results.

In figure 2, the upper right shows the original image of a line-drawing imaged with a Sony XC77 camera under good lighting conditions. To reduce camera noise, 16 images were taken and averaged. The total dynamic range of the window is 38 gray values. The mean value of the lighter area is around 115. The mean value of the dark regions is about 90-95. The upper left is the same image with uniformly distributed noise, from the range [-5,5], added. The remaining panels show the results of different detail preserving algorithms. Except for the G-smoothing algorithm, where G was chosen based on the noise level, the free parameters of the algorithms were adjusted by the first author to achieve a subjectively "best" result. Decide for yourself which results are "good".

The lower left is the results of 1 iteration of Weymouth's algorithm with parameters .3 .3. The lower right is the

results of using a Nagao operation with a 3x3 window. The middle right is the results of using 3×3 median filtering, with the central pixel given 5 votes. (If we use median with uniform votes, the detail is almost totally lost!) The middle left shows the results of 1 iteration G-neighbor smoothing with a G of 10. (Other G values are on this example are shown in figure 3).

In Figure 4, we the algorithms have applied to the same original scene (upper left), which has been corrupted with N(0,5) white noise. The upper left is the noisy input image to be smoothed. The remaining panels show the results of different detail preserving algorithms in the same order as before: middle left is G-smoothing (G=10), lower left is Weymouth (parameters .3 .2), lower right Nagao (3x3), and middle right is median filtering (with the central pixel given 5 votes.)

A more complex example applies the same algorithms to a real scene, shown in the upper left of figure 5. In this case no noise has been added. A "mode" filtering (with central pixel given 3 votes) appears in the upper right. The remaining quadrants are labeled the same as in figure 4: middle left is G-smoothing (G=10), lower left is Weymouth (parameters .3 .3), lower left Nagao (3x3), and middle right is median filtering (with the central pixel given 3 votes.).

The final smoothing example applies the same algorithms to a small part of a thermal image. The original data is shown in the upper left of figure 6. In this case no noise has been added. The "mean" filtering (i.e. unweighted average) appears in the upper right. The remaining quadrants are labeled the same as in figure 4: middle left is G-smoothing, lower left is Weymouth, lower left Nagao (3x3), and middle right is median filtering (with the central pixel given 3 votes.) The thermal image was supplied By Texas Instruments Research Center and is from their second generation FLIR, a 8-12 micron uncooled sensor with NTSC output.

Overall we see that the G-neighbor smoothing approach is quite good in preserving small details. Its computational cost may make it a good choice for certain applications. Because of the robustness properties of the median, it might also be interesting to experiment with G-neighbor median smoothing.

2.2 Morphology

Morphology is a non-linear filtering, generally performed to remove "feature noise". In the case of binary morphology, operations are performed on a binary image, which is obtained by applying a foreground/background operator to an image. This operator determines "features". Binary morphology allows one to perform operations that basically depend on the shape/size of the features to remove (or fill in) portions of the foreground. There have been many papers one the mathematical properties of morphology as well as its practical applications, e.g. [Haralick *et al.*-1987, Vogt-1989, Giardina and Dougherty-1988].

One of the difficulties of binary-morphology is that it depends only on the shape/size of features, not on the actual signal values used to obtain them. A generalization, gray-scale morphology, is signal dependent, but requires significantly more computational effort.

We propose that G-neighbors provide a means for simple signal-dependent morphology, while still providing efficient binary processing. In G-neighbor morphology, the standard operations of dilate and erode are restricted to apply only to G-neighbors. Thus the morphology is being done over arbitrary graphs (as opposed to a regular lattice). Because these graphs have more "boundary" points, and connectivity patterns not all of the properties of traditional morphology carry over. The symmetric nature of G-neighbor definition insure that many properties do carry over. For example, the idempotent property (probably the most important property) carries over for G-open and G-close. The extensive and anti-extensive properties also carry over.

G-neighbor binary morphology operator dilate "turns on" a pixel if any of its G-neighbors are on. Erode "turns off" a pixel if any of its G-neighbors are off. As in traditional binary morphology, one can use both 4 and 8 connected neighborhoods (and often mix the two). In Figure 7 we give some examples of binary morphology and G-neighbor binary morphology.

To describe the examples, we use the following notation. Let 3d8 mean 3 iterations of 8-connected dilation. Similarly 5e4 is 5 iterations of 4-connected erosion. Opening with a 7x7 square would thus be 3e8 3d8.

The lower right is the input image, obtained from the noisy example used in figure 2. The image was thresholded to yield a binary image, which is in the middle bottom. Black represents foreground. The first column presents some

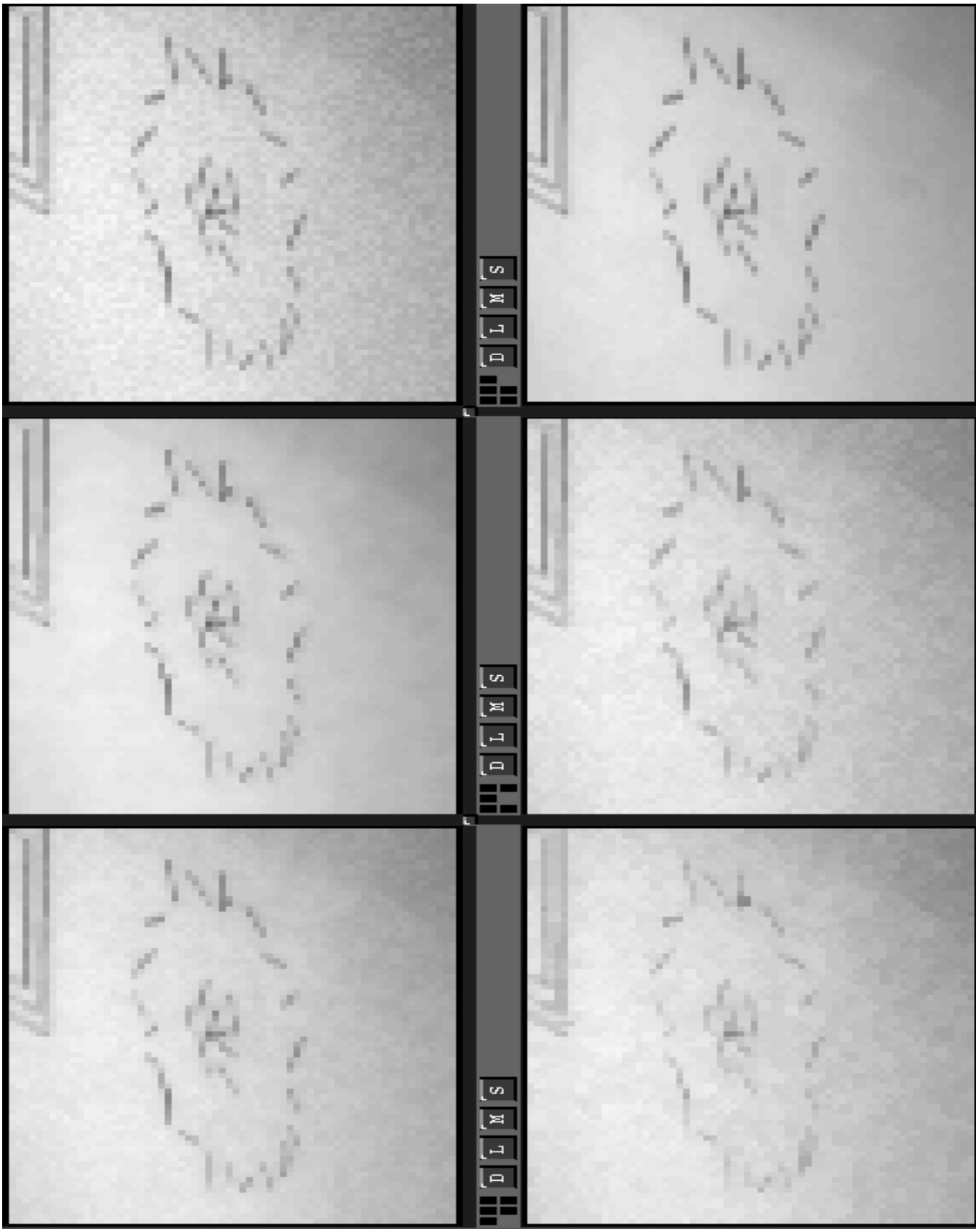

Figure 2: Detail preserving algorithms applied so a simple image with uniform noise. Top row shows noisy image (left) and original image (right). Middle row shows G-smoothing and median smoothing. Bottom row is Weymouth smoothing and Nagao smoothing. See text for more details.

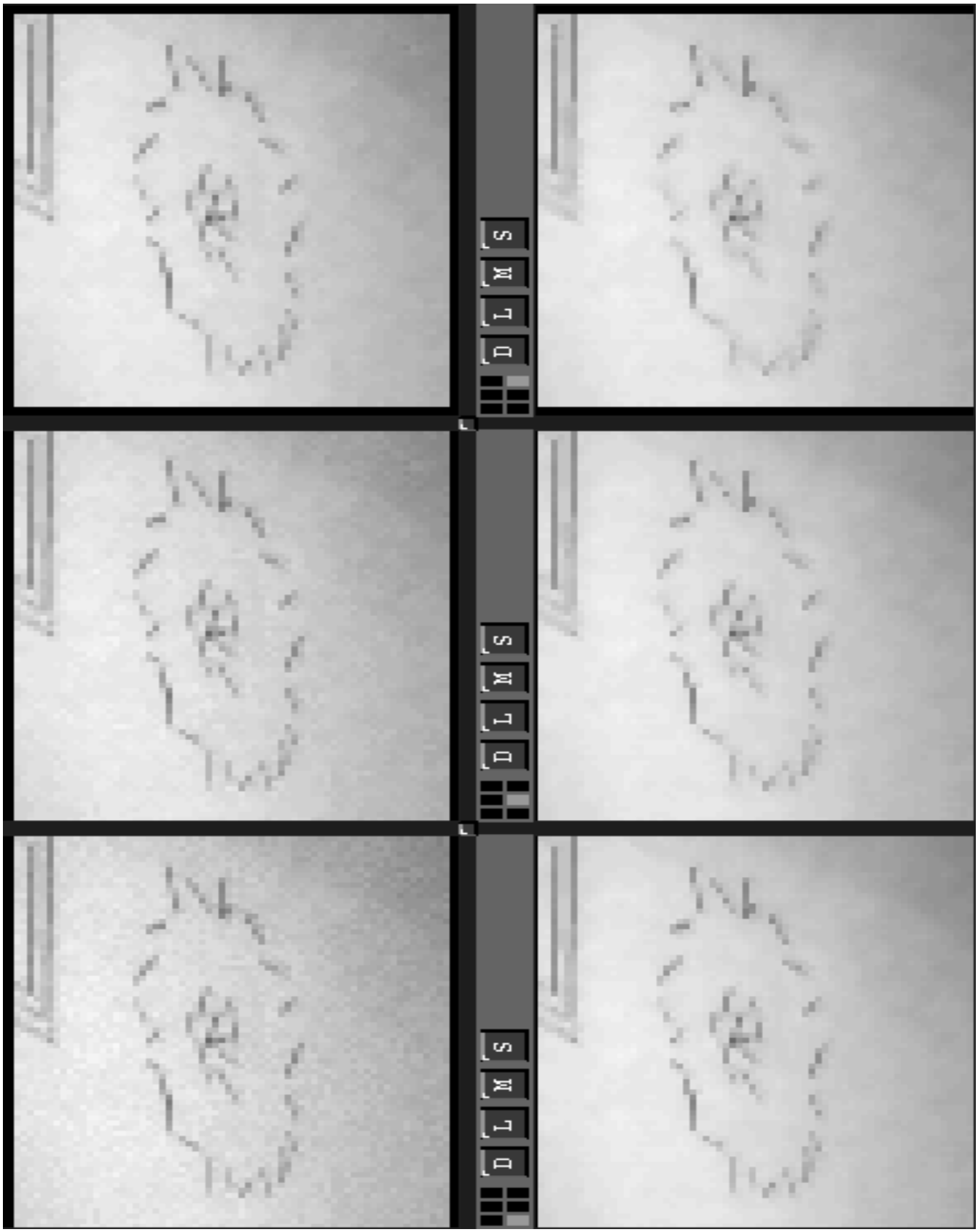

Figure 3: G-smoothing for various values of G applied to the example in figure 2. The left column shows, top to bottom, G=7,5,2. The right column shows G=20,15,10.

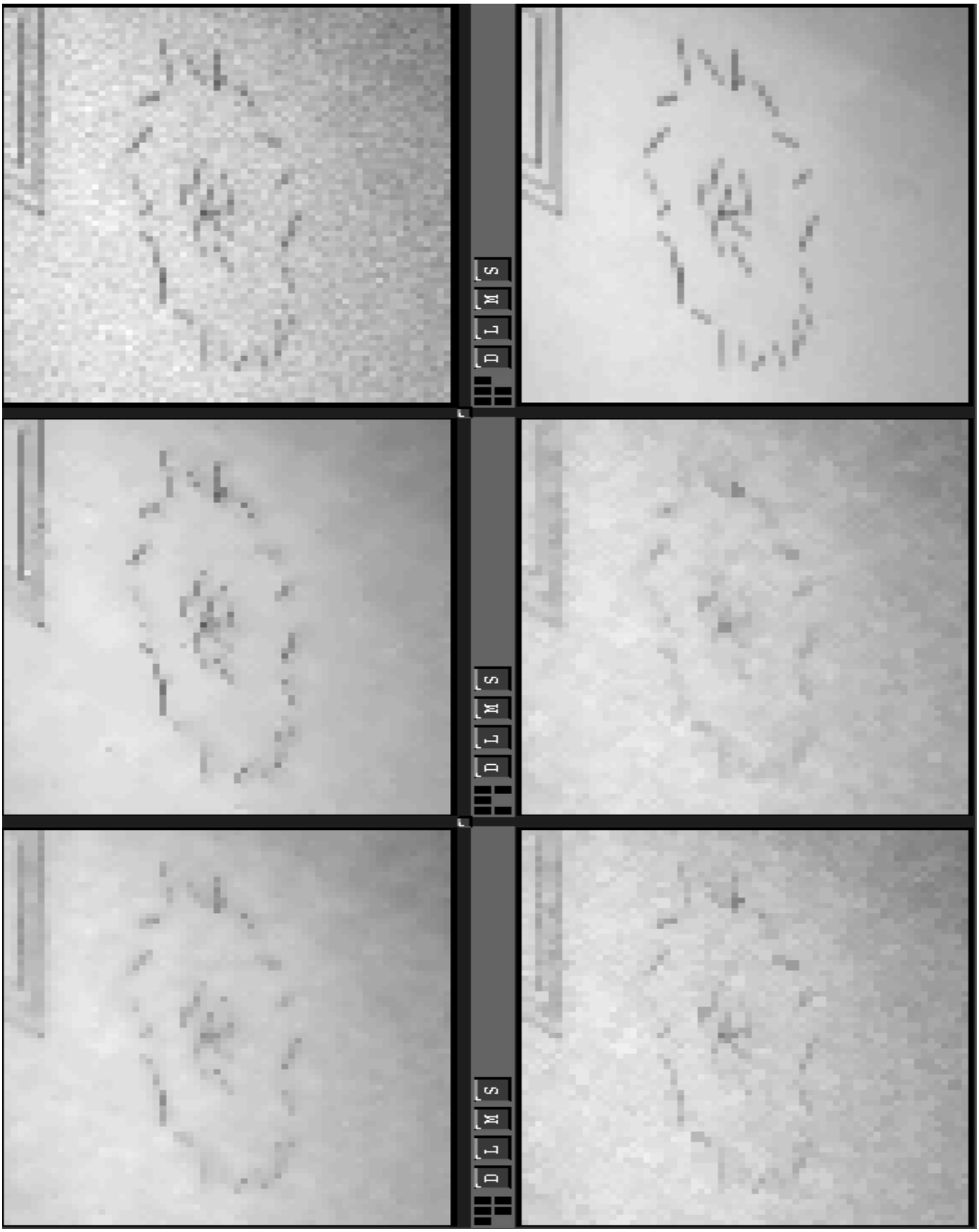

Figure 4: Detail preserving algorithms applied to scene with Gaussian noise. Top row shows noisy image (left) and original image (right). Middle row shows G-smoothing and median smoothing. Bottom row is Weymouth smoothing and Nagao smoothing. See text for more details.

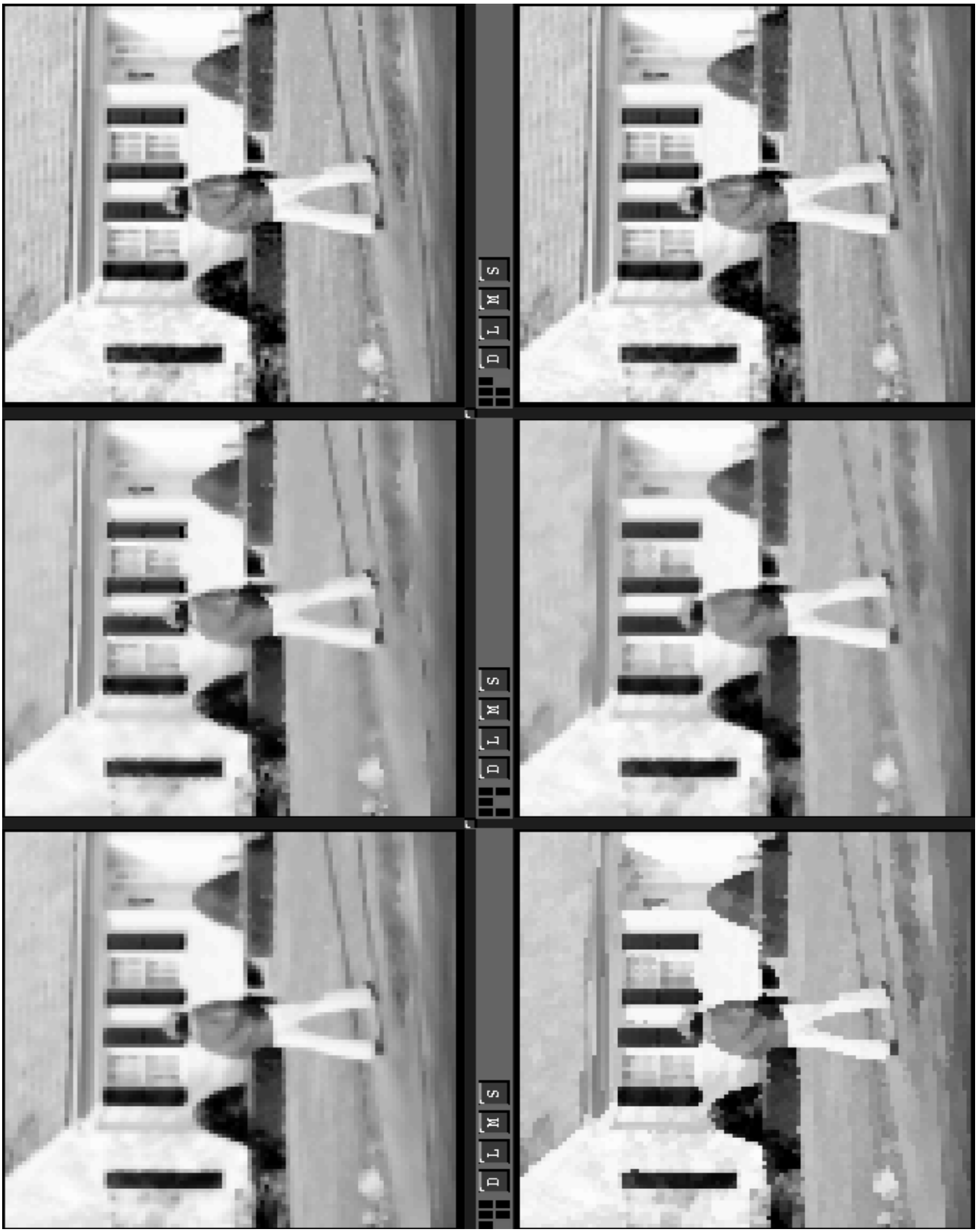

Figure 5: Detail preserving properties applied to a more complex image. Top row shows mode filtering (left) and original image (right). Middle row shows G-smoothing and median smoothing. Bottom row is Weymouth smoothing and Nagao smoothing. See text for more details.

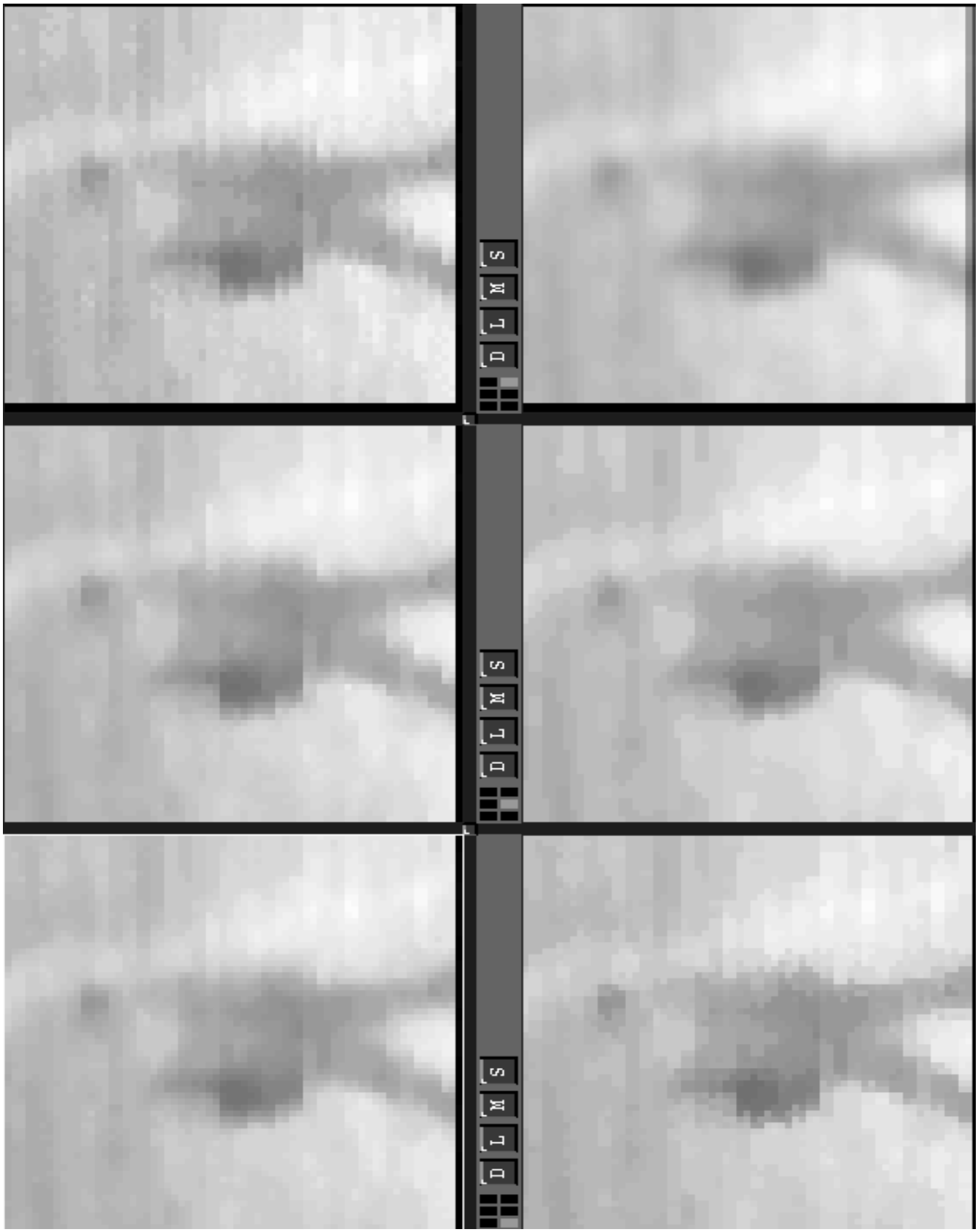

Figure 6: Detail preserving properties applied to a thermal (8-12 micron) image. Image courtesy of Texas Instruments. Top row shows mean image (left) and original image (right). Middle row shows G-smoothing and median smoothing. Bottom row is Weymouth smoothing and Nagao smoothing. See text for more details.

regular binary morphology examples. From bottom to top, they are: original gray-scale image, 1e8 1d8, 1d8 1e8, 2d8 2e8, and 1d8 1e8 1e4 1d4. Note how the opening totally removed the "figure". The closing leaves much of the "figure", but it is not overly connected and blocky. The last example drives home what is happening, any erosion which is not preceded by a dilation will almost totally remove the "figure" because it is so small.

The second column are basic G-morphology dilate and erode examples. They are, from bottom to top, original binary, 1d8 G=5, 3d8 G=5, 1e4 G=5, 1e4 G=10. Note that the dilations do not make the image too blocky, and the erosions leave much of the figure intact. Regular morphology erosions would totally remove the figure.

The left column show more complex G-morphology examples. From bottom to top they are: 1e8 G=5, 1e8 1d8 G=10 , 1d8 1e8 G=10, 1e4 1d4 G=10, and 1d8 1e8 1e4 1d4 G=10. We see that in general the operations preserve the image well. However, the 1e8 operations does behave a little non-intuitively. This is partially because we did not weight G as a function of distance, and so diagonals are more likely to be G-neighbors than are the 4-connected neighbors.

If the value of G is small (compared to the dynamic range of the data), the neighborhoods tend to be small. In these cases many structuring elements will not fit within the neighborhood. Remember, operations in G-neighborhood are actually within connected components of the graph. It is possible for a connected component to consist of a single pixel (or a small number of them). If the entire component is in the foreground (or background), it is not affected by G-morphology operations – the entire set is in (or out), and that does not change.

It has been our experience that the boundary affects can be significant. For example, if the G-connected components are small, dilation by a large structuring element may result in the entire connected component being considered foreground. Once this happens, further operations have no affect. These types of affects occur for regular morphology, just around the borders of the image and so they are often ignored. Because of the many borders, G-neighbor morphology can be a bit harder to use – one can not predict its behavior just by looking at the binary image. Given an understanding of the original gray image, they can be use effectively.

In practice, we generally use much larger G values for morphology than for smoothing. This allows G-morphology to behave mostly like regular morphology, but still be sensitive to extreme variations. With a decent camera, reasonable lighting and a G of about 30-50, no camera noise or lighting variations will cause G-discontinuities. If an object is protected from removal by G-morphology in such circumstances, it should be investigated.

3 Properties of G-neighbors

Hopefully the previous section has lead the reader to view G-neighbors as a potentially useful concept. In this section we briefly discuss some of their properties and then their implementation.

First we note that the computation of G-neighbors is quite inexpensive. Its cost depends mostly on the cost of the neighbor test $Neigh(a, b)$. We can represent the neighbor relation in a graph, which we can implement using one byte per pixel, one bit per neighbor. This representation actually uses 2 bits per graph arc ("neighbor" pair), while only one is needed. This does, however, make for more efficient traversals and is generally easier to compute. The encoding, which can be viewed as a set of 8 bit-planes, can be stored in a standard image. For many real-time image processing engines, such these types of combined/compressed bit-masks can be used to implement G-neighbor operations in near-real time (a few frame times). In section 4, we describe such an implementation.

When two neighbors, in the 4/8-connected sense, are not G-neighbors, then there is a G-discontinuity between them. In general, a discontinuity in the $Neigh$ measure, then defines a micro-scale edge detector. Note each pixel can have up to 4/8 micro-edges, somewhat like a line processes in Markov Random Field edge detectors. For $Neigh_G$, this micro-edge detector is a threshold edge detector, in the sense of [Abdou and Pratt-1979]. It is one where the strength of the edge is not known and the direction is rounded to one of 8 orientations (4 in 4-connectedness). Figure 8 shows a G-neighbors map of an area on an edge. In the more general definition of $Neigh$, unconnected neighbors are still micro-edges, with the interpretation depending on $Neigh$. For example, with $Neigh_M$, it is a thresholding edge detector based on relative contrast across the edge.

Before you say, "We don't need yet another edge detector", we point out that edge detection is not the goal of G-neighbors. As edge detectors go, this is about as simple as they come: there is no smoothing – so its may be noise sensitive – and the directional selectivity has limited resolution – only 4/8 choices. The interpretation as an edge

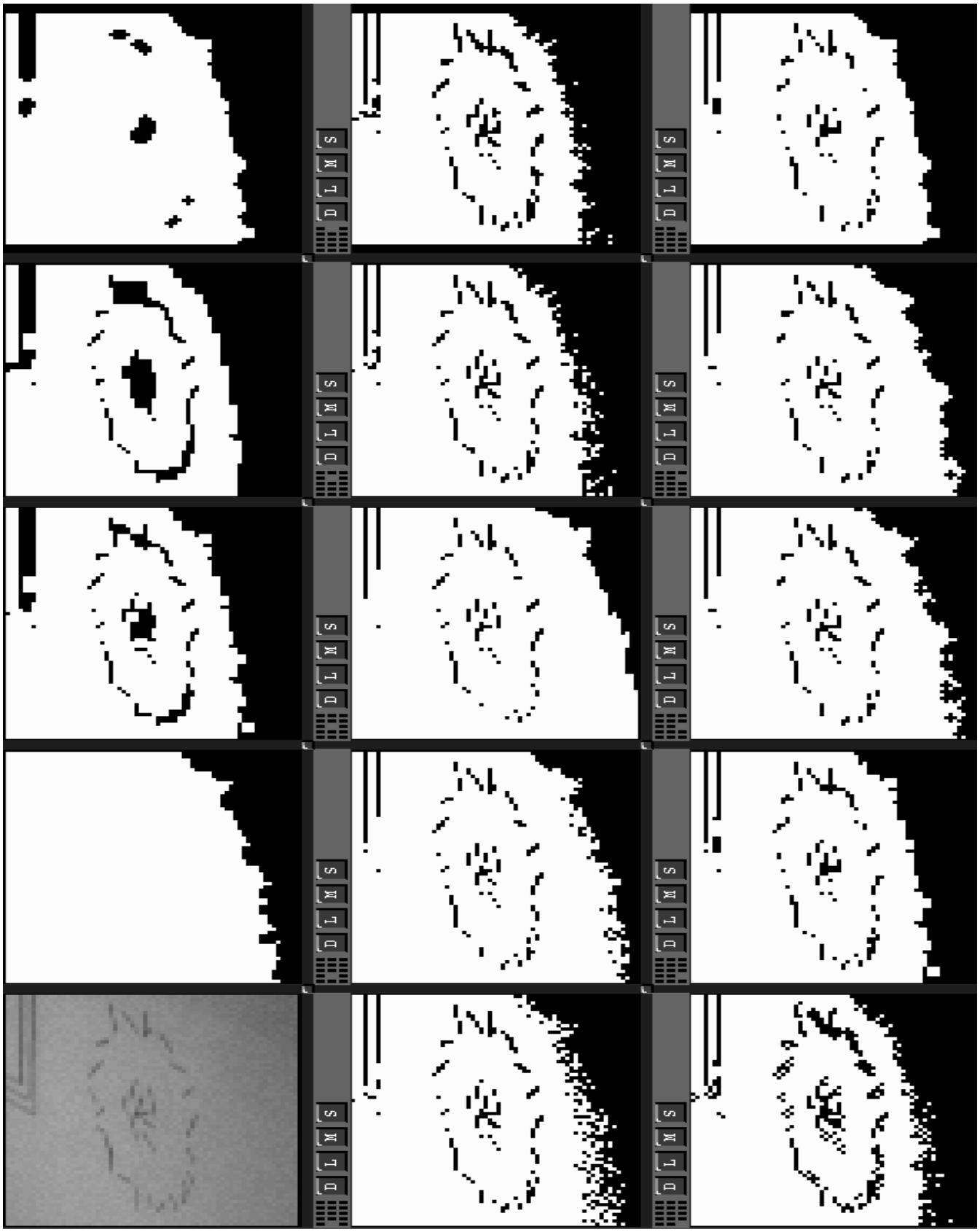

Figure 7: Morphology examples. See text for explanation.

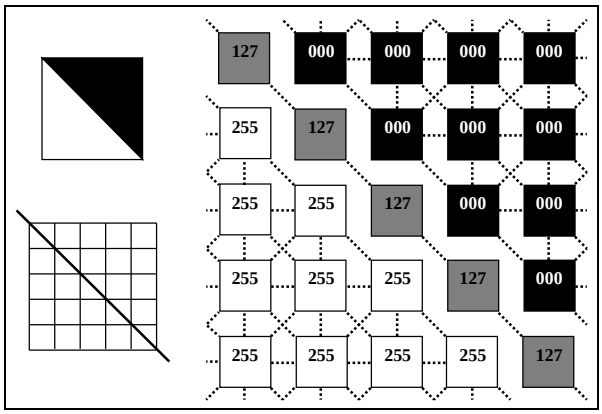

Figure 8: Top Left: Image with an edge. Bottom Left: Location of the edge across the pixel grid. Right: Magnification of the image at the edge with, say, G=100.

detector is just a way of explaining part of the behavior of the G-neighbors. The micro-scale of the edge-detection characterization is important to understanding their behavior and usefulness.

Given a G-neighbor definition, one can define a *G-neighborhood* to be the G-neighbors in the local 4/8 connected neighborhood of a pixel. One can also define a *G-component* as the set of all image points in a single distinct connected component. Because of the symmetry property, it is easy to see that if pixel A is in pixel B's G-neighborhood (G-component), then pixel B is in pixel A's G-neighborhood (respectively G-component). In figure 8 we can see three G-components, one on either side of the edge and one along the edge. An image with a greater concentration of low frequency components (i.e. a smoothly varying image) would tend to have less G-components, i.e. fewer G-connected components. In an image with many high frequencies, a pixel would tend to have fewer G-neighbors and so more G-components are expected. The value of G and the neighborhood metric both affect the size and number of G-components. For the neighborhood metrics mentioned above, a lower G would tighten the requirement to be a G-neighbor and therefore there will be fewer G-neighbors and more connected components. This can be seen in figures 1 and 10 which show the G-neighbors in a small image for 2 different values of G. By increasing the number of connected components, the size of each connected components is reduced. Operations that are restricted to a single connected component would tend to be more local.

Pixels within one G-component have, by definition, at least one path between each pair of pixels. These paths can be defined as *G-path*. We can also define the *G-distance* as the distance along the shortest G-path connecting two pixels, with the understanding that the distance to a point in a different G-component is $+\infty$. Although physically adjacent pixels that are in the same connected component usually have a G-distance of one, it is not too uncommon that the G-distance is greater, sometimes much greater. For example, in figure 9 the two pixels in the bottom center are adjacent but the G-distance is much greater than one. The properties of G-distances, as compared to other digital distances, is a subject for future study.

It is important to note that inside one connected component there can exist all possible intensity levels. If G is high or if the image is smoothly varying, connected components again would be large and would tend to contain a large range of intensity values. A ramp image or sub-image would hence contain a wide range of intensity levels inside a single connected component. Figure 9 shows a wide range of pixel intensities within one connected component.

While it might seem desirable that as a scene edge is is translated, the G-components of the pixels of the image near the projection of that edge should also approximately translate. This is often not the case. An edge in an image falls over the pixel grid. As the scene/camera is translated, the edge falls over the pixel grid differently producing pixels of different intensities for the same edge. This changes the G-components. This can be seen by comparing Figure 8 and Figure 11.

Finally, many cameras do not use a unit aspect ratio. This effects the G-component depending on the orientation of the image. A different value of G for the horizontal and vertical directions can be used to minimize such distortions.

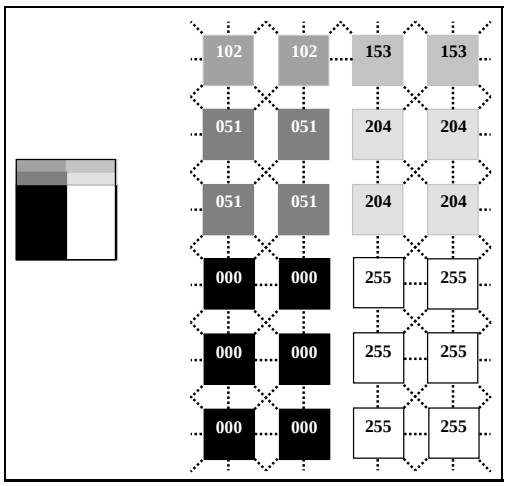

Figure 9: Left: Image. Right: Magnification of image with G=60.

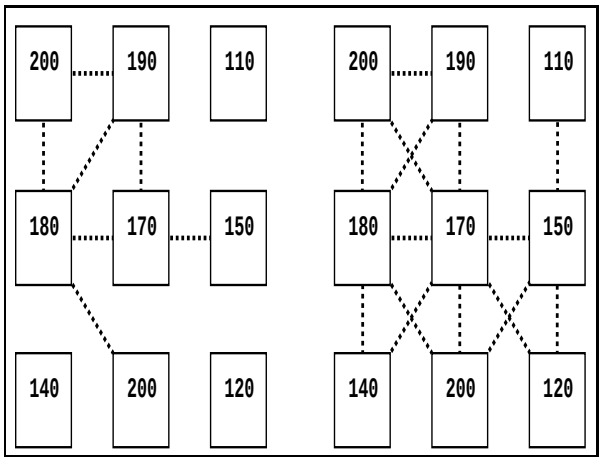

Figure 10: G=20 and G=50

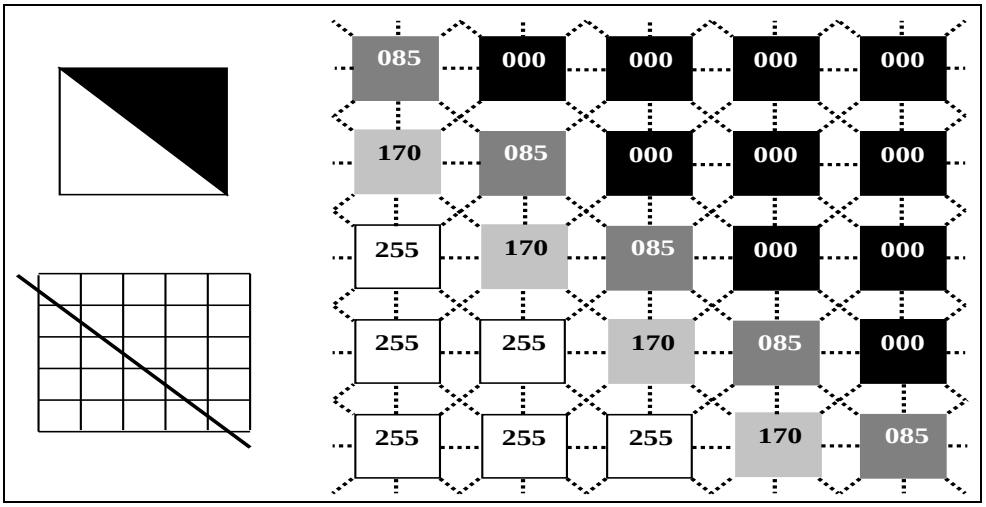

Figure 11: Top Left: Image with an edge. Bottom Left: Location of the edge across the pixel grid. Right: Magnification of the image at the edge with, say, G=100. For G< 85, multiple connected components would result. Compare with figure 1

4 A parallel implementation of G-neighbors

Compared to other techniques, G-neighbors are quite simple. This is both their failing and their saving grace. Because of their simplicity, they can be implemented efficiently. On pixel parallel machines, their local nature makes them natural candidates for real-time applications. In this section, we present a brief overview of our real-time parallel implementation of the computation of G-neighbors, G-neighbor based detail-preserving smoothing, and G-neighbor-based morphology. The parallel implementation was carried out using a PIPE processor.

4.1 Overview of the PIPE machine

The parallel machine used to implement the G-neighbor based algorithm is a PIPE[1], a parallel Pipelined Image Processing Engine. This section briefly overviews the machine programming model and hardware capabilities.

The pipe has a video stage (input digitizers/+ output DtoA) and 3-8 modular processing stages (MPS). Each stage is programmed as though it had a 2D array of processors (256x256 in our case, larger models are available). The logical clock time is 1 frame time (1/60 sec). Each of these logical processors can obtain data (almost any weighted combination) from their 10 neighbors (8 spatial neighbors in this MPS, plus the pixels (at the same location) in the previous MPS stage and the next MPS state. The data from links to other stages called forward and backward links can also be passed back to the same location. (These forward/backward/recursive link pass the data into the next frame time). MPS's can also communicate using one of 6 global busses.

The input data can be combined in a variety of ways using ALU's and lookup tables. Addition, subtraction, and logical operations use ALU's. More complex operations (e.g. multiplication, sqrt, ...) are accomplished using lookup tables. Each MPS can store a small number of images (2 in our machine). The machine can actually do many operations in a single cycle (if the program is structured right one can do 2 neighborhood operations, a two value-lookup, 3 different ALU operations, and a few 1 value lookup tables.).

4.2 Parallel implementation of G-neighbor computation

The parallel implementation has 3 parts. The computation of the Neighbor mask, G-neighbor smoothing and G-neighbor morphology. We discuss each in turn.

The computation of the G-neighbor mask could be accomplished by using 8 frame times and comparing in each direction. Instead, we use 4 stages, having each compute differences in 2 directions (making use of 2 neighborhood operators), and combine the results in each stage using a two valued lookup table. (Computing, in parallel, 4 G-neighbor bit-mask, each with at most 2 bits turned on). Two pairs of these are then combined at the end of the first cycle (using ALU's). If one wants to use 8–connected based G-neighbor, then the two partial results are combined in a second cycle. The pipelined algorithm computes G-neighbor masks in near-real time; 4 connected masks are computed with 1/60 of a second delay, 8 connected G-neighbor masks with a 1/30 of a second delay.

If the Neighborhood operator $Neigh$ is something other than gray-level difference, it may still be possible to use the above algorithms. For example, to get relative gray-level difference ($Neigh_M$) is just as easy. It simply requires different lookup tables for the neighborhood operations.

To do G-neighbor smoothing, we needed to make use of a feature of the PIPE called the ROI, region of interest. This feature allows the programmer to have up to 16 different "programs" operating at the same time, where each pixel can use a bit mask to choose which of the 16 "programs" to use. For 4 connected G-neighbor smoothing this is sufficient to do frame-rate computations: since there are only 16 different possible neighborhood combinations we can precompute a neighbor weighting matrix for each and then just choose the appropriate one. For 8-connected smoothing, we have 256 potential neighborhood, and so we need 2 MPS (so we can get 16*16 combinations). Each stage does a different 4 connected (standard, and the diagonals) and then an ALU is used to combine (average) the results. Thus we can do 1 iteration of G-neighbor smoothing in a single cycle (1/60 of a second).

The implementation of G-morphology is slightly more complex. Like smoothing we use the ROI to allow different

[1] A product of Aspex, Inc, NYC NY

neighborhood operations. In fact, the implementation of a single Morphological operation is almost identical to the 4 or 8 connected smoothing (except that the operators applied do logical operations rather than averaging). The more complex part is to take a sequence of morphological operations and then load these together. At the current time, this is done by providing each of the basic operations (dilate (4 and 8) erode (4/8) open (4/8) and close (4/8) and using them build a program. Each sequence of morphological operations requires a different program. The initial operation to obtain the binary foreground/background figure for morphology is a separate module (currently we just threshold). It is not important to the implementation that the G-neighbor mask is held constant throughout the morphological operations, but it is not clear what it would mean for it to change during morphology. Thus, the time for morphological operations (once setup) is 1/60 second per dilate/erode.

Reviewing the operations needed for the above parallel implementation it should be clear that a near-real time program could be written for almost any reasonable "pixel" processing machine (e.g. Datacubes, Androx, Maspar).

5 Pros and Cons of G-neighbors

We end with a summary of the good points (+) and bad points − of G-neighbors.

+ G-neighbor computations can be done in near-real time on specialized hardware (E.g. PIPE). Fast serial implementations are also available.

+ G-neighbor smoothing is reasonable at detail preserving smoothing for small features with a reasonable signal to noise ratio. It can preserve single pixel features, or one pixel width lines, while still providing significant smoothing around them.

+ G-morphology can provide signal dependent morphology at almost binary cost.

± Neighborhood definitions are simple. Thus they are cheap. However good neighborhood delineation may need larger region of information and more complex operators (e.g. textured neighborhoods.)

− Simple "edge" definitions and locality of operation make G-smoothing weak when looking at large objects in high noise situations.

− If G is not chosen appropriately, results can become too localized (G=0 implies every pixel is its own neighborhood. Then G-smoothing and G-morphology do nothing!), or loose their effectiveness G=255 and then G-neighbors are the same as regular neighbors).

− G-morphology is sometimes hard to use because of non-intuitive boundary affects. For example, a point with no G-neighbors will never be removed (or filled). This may only be an artifact of inexperience in the use of G-morphology.

− We currently do not have a good theory as to how to choose G as a function of noise levels and expected signal levels. We have not, however, had difficulty choosing reasonable values.

6 Conclusions

The paper examines the concept of G-neighbors. We showed how they can be useful for detail preserving smoothing and morphology. The simple/local nature of G-neighbor definitions make them ideal for implementation on low-level pixel parallel hardware. A near real-time parallel implementation of the G-neighbor computation, including G-neighbor-based detail preserving smoothing and G-neighbor morphology, is discussed.

Acknowledgements

T. Boult is supported in part by NSF PYI grant #IRI-90-57951, DARPA contract DACA-76-92-C-007, and Texas Instruments. The thermal images were supplied by TI.

R. Melter was supported in part by an NSF ROA supplement to T. Boult NSF grant #IRI-90-57951, and also by a Long Island University Faculty Research Grant.

R.Melter and I.Stojmenovic are supported in part by NATO Collaborative Research Grant CRG 900840.

I. Stojmenovic is supported in part by Natural Sciences and Engineering Research Council of Canada operating grant OGPIN007.

References

[AAI, 1985] AAI. Kbvision. Technical report, Amerinex Artifical Intelligence, Inc., 1985. Vision software and associated Manuals.

[Abdou and Pratt, 1979] I.E. Abdou and W.K. Pratt. Quantitative design and evaluation of enhancement/thresholding edge detectors. *Proceedings of the IEEE*, 67:753–763, 1979.

[Chin and Yeh, 1983] R.T. Chin and C.L. Yeh. Quantitative evaluation of some edge-preserving noise-smoothing techniques. *Computer Vision, Graphics, and Image Processing*, 23:67–91, 1983.

[Davis and Rosenfeld, 1978] L.S. Davis and A. Rosenfeld. Noise cleaning by iteratated local averaging. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-8:705–710, 1978.

[Gallagher and Wise, 1981] N.C Gallagher and G.L. Wise. A theoretical analysis of the properties of median filters. *IEEE Trans. on Acoustics, Speech and Signal Processing*, ASSP-29(6):1136–1141, December 1981.

[Giardina and Dougherty, 1988] C.R. Giardina and E.R. Dougherty. *Morphological methods in image and signal processing*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1988.

[Graham, 1966] R.E. Graham. Snow removal – a noise-stripping process for picture signals. *IRE Trans. Information Theory*, IT-8:129–144, 1966.

[Haralick and Watson, 1981] R.M. Haralick and L. Watson. A facet model for image data. *Computer Vision, Graphics, and Image Processing*, 15:113–129, 1981.

[Haralick *et al.*, 1987] R.M. Haralick, S.R. Sternberg and X. Zhuang. Image analysis using mathematical morphology. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-9(4):532–550, 1987.

[Huang *et al.*, 1979] T.S. Huang, G.J. Yang and G.Y. Tang. A fast two-dimensional median filtering algorithm. *IEEE Trans. on Acoustics, Speech and Signal Processing*, ASSP-27(1):13–18, February 1979.

[Lee, 1980] J.S. Lee. Digital image enhancement and noise filtering by use of local statistics. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-2(2):165–168, March 1980.

[Lee, 1981] J.S. Lee. Refined filterting of image noise using local stastics. *Computer Vision, Graphics, and Image Processing*, 15:380–389, 1981.

[Nagao and Matsuyama, 1979] M. Nagao and T. Matsuyama. Edge preserving smoothing. *Computer Vision, Graphics, and Image Processing*, 9:394–407, 1979.

[Narayananand and Rosenfeld, 1981] K.A. Narayananand and A. Rosenfeld. Image smoothing by local use of global information. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11:826–831, 1981.

[Narendra, 1981] P.M. Narendra. A separable median filter for image noise smoothing. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-3(1):20–29, January 1981.

[Overton and Weymouth, 1979] K.J. Overton and T.E. Weymouth. Image enhancement. In *Proceedings of the IEEE Computer Society Conference on Pattern Recognition and Image Processing*, pages 498–507, June 1979.

[Scher *et al.*, 1980] A. Scher, F.R. Dias Velasco and A. Rosenfeld. Some new imagge smoothing techniques. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10:253–258, 1980.

[Tomita and Tsuji, 1977] F. Tomita and S. Tsuji. Extractaction of multiple regions by smoothing iin selected neighborhoods. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-7:107–119, 1977.

[Tukey, 1976] J.W. Tukey. *Exploratory Data Analysis*. Addision-Wesley, Reading, MA, 1976. Chapter 7.

[Vogt, 1989] R.C. Vogt. *Automatic Generation Of Morphological Set Relation Algorithms*. Springer-Verlag, NYC, NY, 1989.

[Wang *et al.*, 1981] D.C.C. Wang, A.H Vagnucci and C.C. Li. A gradient inverse weighted smoothing scheme and the evaluation of its performance. *Computer Vision, Graphics, and Image Processing*, 15:167–181, 1981.