With J: Image Processing 1: Smoothing Filters

Cliff Reiter
Mathematics Department
Lafayette College,
Easton PA 18042, USA
reiterc@lafayette.edu

Image processing includes techniques that are used to correct defects in images and to enhance the visibility of features of interest. In this note we will look at several different methods for smoothing images in order to remove specks of dust and artifacts from scanning. In future notes we plan to discuss other techniques for enhancing images in the spatial domain including the removal of orientation and lighting defects. Previous discussions [6,7] considered the removal of motion blur in frequency domain using Fast Fourier Transforms.

Figure 1 shows a grayscale image of a snowflake and a zoom into the image that shows individual pixels. The image was scanned from Bentley and Humphreys' classic book of snowflake images [1]. We have recently used images from that book as figures in papers on snowflake growth [2,3]. The zoom into the image makes dust and undesirable artifacts of scanning the half-toned image apparent. Notice the textured, almost periodic gray points appearing in what should be white regions. We aim in this note to remove or diminish the appearance of the dust and texture while maintaining a clear image of the snowflake.

When correcting defects in an image, there is a presumption that the features of interest in the image are somehow different from the defects to be diminished; otherwise, features can not be distinguished from defects. We often assume that the features of interest are larger than the defects; thus, the image can be corrected by using suitable averages. We will also look at rank based methods including an ad hoc technique we used for creating black and white images for [3]. For those filters the presumptions are somewhat different.

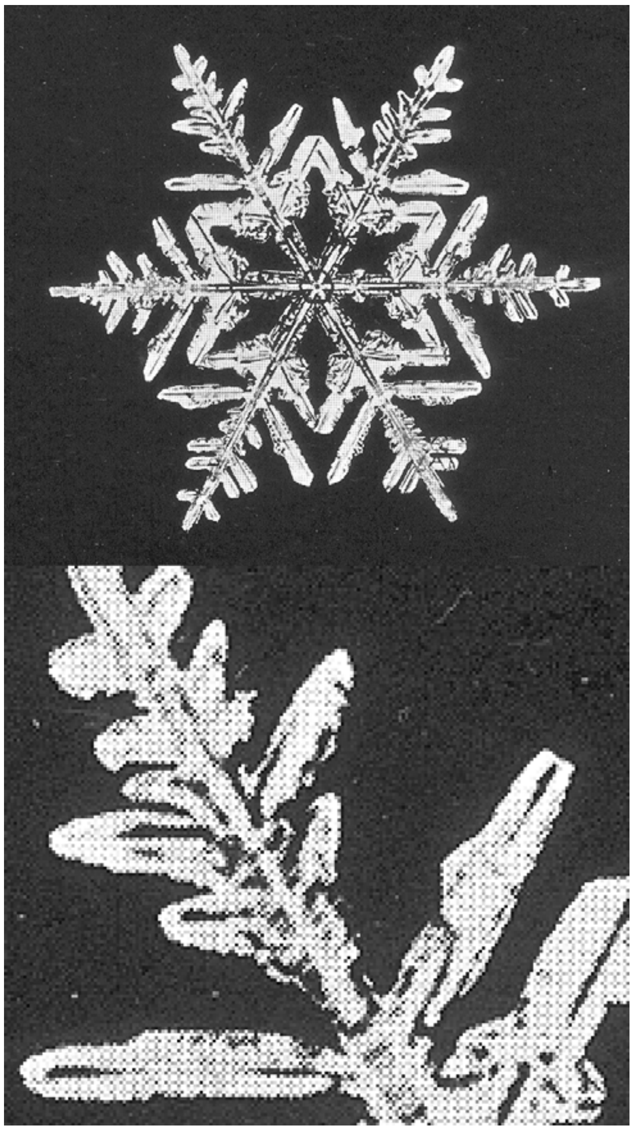
Figure 1. Scan and Zoom of a Snowflake Picture

While we will consider several methods in this note, there are many techniques for removing defects of this type, each with its own merits. Excellent references to digital image processing include Russ [9] and Gonzalez and Woods [5]. Packages implementing images processing techniques include commercial packages, such as Photoshop^{tm} and gnu software, such as GIMP[4]. A script giving the J definitions used in this note, and the sample image, is available from [8].

Multiplicative Filters

The simplest filters replace each pixel by an average of the nearby pixels. A three by three matrix of ones is often used to represent the average obtained from averaging a pixel value with the values of its nearest neighbors.

```
                        1 1 1
                        1 1 1
                        1 1 1
```

Often one wants the pixels nearest the center to be weighed more heavily than the distant pixels. These non-uniform averages can be represented by a matrix of the weights. It is understood that the pixel being updated is replaced by a sum of the nearby pixel values times the weights shown in the matrix and divided by the sum of the coefficients in the matrix. The pixel itself corresponds to the central entry in the matrix, such as the 4 in the weights below.

```
                        1 2 1
                        2 4 2
                        1 2 1
```

A systematic way to choose the weights is to use a normal distribution (also called a Gaussian distribution) with a specified standard deviation and width of the sample. By making a multiplication table and rescaling so the entries sum to 1, we get an approximation to the binormal distribution function. This can be implemented as follows where the left argument is the (1-dimensional) standard deviation and the right argument is the number of sample points (pixels) to be used in each direction.

```
   gauss=:1 : 0
[:(%+/)^@-@*:@-:@(%&m.)@:(i.--:@<:)
)

   gauss2d=:1 : '[: */~ m. gauss'

   1 gauss 5
0.11170 0.23648 0.30364 0.23648 0.11170

   1 gauss2d 5
0.01248 0.02642 0.03392 0.02642 0.01248
0.02642 0.05592 0.07180 0.05592 0.02642
0.03392 0.07180 0.09220 0.07180 0.03392
0.02642 0.05592 0.07180 0.05592 0.02642
0.01248 0.02642 0.03392 0.02642 0.01248
```

We can apply the binormal weights on all 5 by 5 neighborhoods (using the adverb `MFilt2d`) or apply a horizontal and vertical pass of the 1 dimensional filter in blocks of size 5 (using the adverb `MFilt1d2`), obtaining the same result in both cases.

```
   round=:<.@(0.5&+)

   MFilt2d=: 1 : 0
[: round ($ m.)"_ +/@,@:(m.&*);._3 ]
)

   MFilt1d2=: 1 : 0
[: round (# m.)"_ +/@:(m.&*)\"1 (# m.)"_
+/@:(m.&*)\ ]  NB. one line
)

   require 'addons\image3\image3.ijs'

   $b=: {."1 read_image 'snowflake.png'
700 780

   time=:6!:2

   time 'm=:(2 gauss 25) MFilt1d2 b'
3.36238

   time 'n=:(2 gauss2d 25) MFilt2d b'
11.3965

   m-:n
1
```

In the above experiment, the array `b` is a matrix giving the gray levels of the image as integers between 0 and 255. We see that using the two passes of width 25 is significantly faster than single filter of size 25 by 25. There is little difference between these two implementations for small neighborhood size, but it is clearly advantageous for large neighborhood sizes to separate the filter into two 1-dimensional passes.

Figure 2 shows the result of applying a Gaussian filter with standard deviation 2 on 9 by 9 neighborhoods which can be created with the following.

```
   f2=:(2 gauss 9) MFilt1d2 b
```

Notice the dust and texture are almost gone, but there is considerable blurring. Using smaller neighborhoods gives less blurring, but is less effective at removing the defects. One can adjust both the number of pixels used and the standard deviation in order to attempt to remove the defects with a minimum amount of blurring, but the technique has a fundamental blurring effect. Interestingly, in other

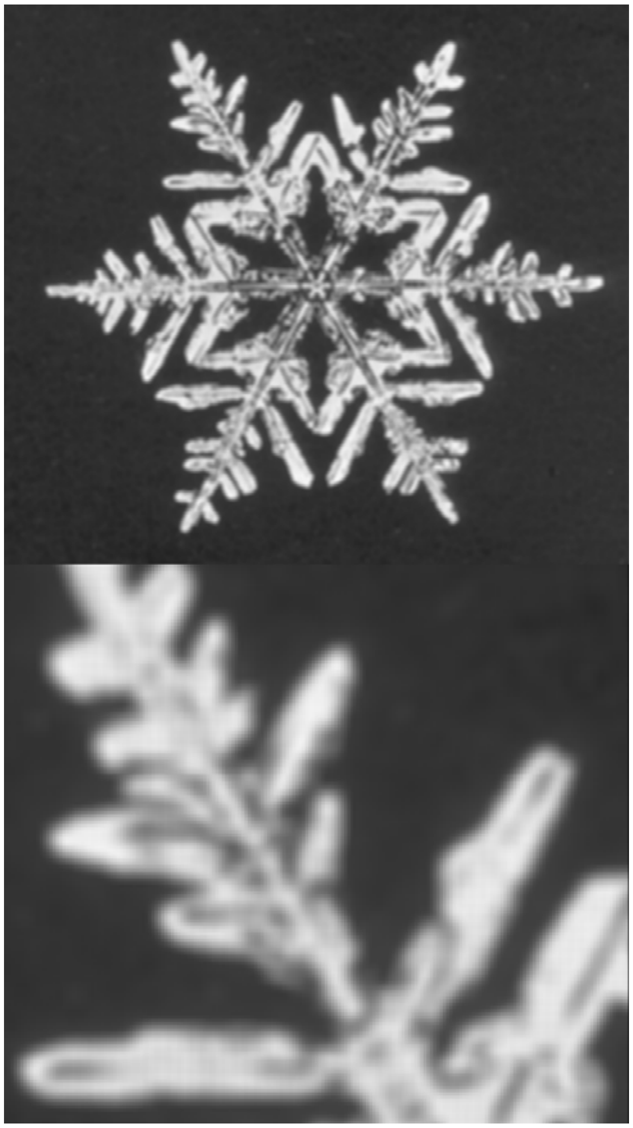
Figure 2. Application of a Gaussian Filter

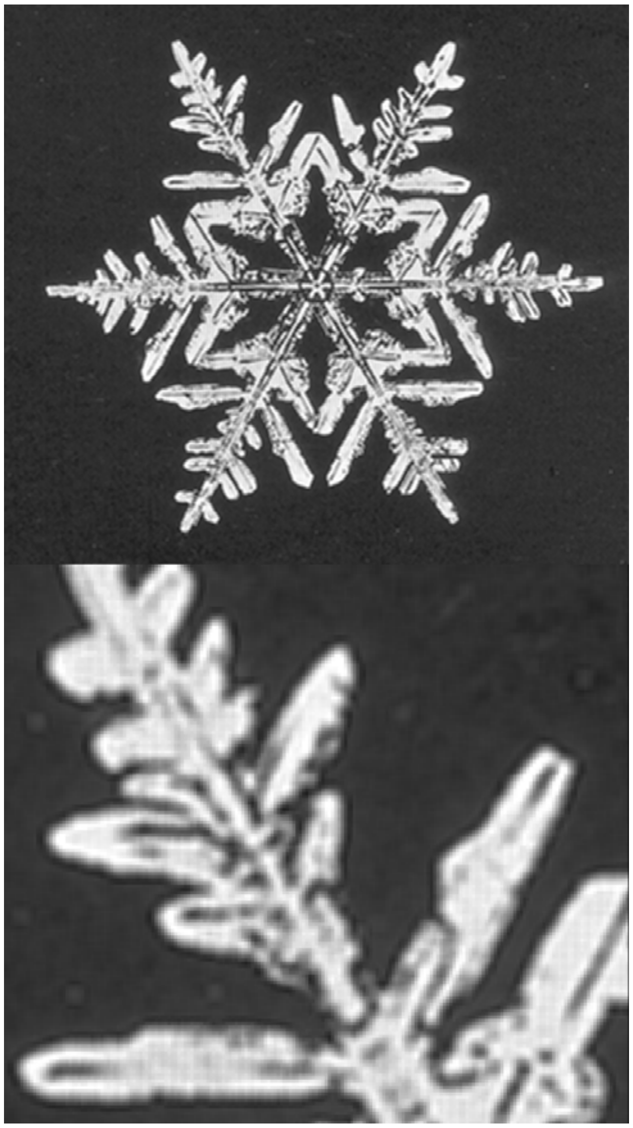
Figure 3. Application of Savitsky-Golay Filter

applications the blurring produced by this filter is a valuable feature. For example, when portions of an image have different lighting, averages over very wide neighborhoods can establish average lighting levels for that portion of the image which then can be used to balance the perceived illumination throughout the image. Nonetheless, in our application, blurring, especially of the snowflake edges, is not desired.

The Savitsky-Golay filter is another multiplicative filter. For this filter, the weights are chosen so that a least square fit of a polynomial to the data is used to interpolate the new pixel value. These weights may easily be computed in J as shown below. The left argument is the degree of the polynomial fit (default 4) and the right argument is the size of the neighborhood.

```
   sav_gol=: 4&$: : ([: {.@%. i:@-:@<:@]
^/ i.@>:@[)  NB. one line

   sav_gol 7
0.021645 _0.12987 0.324675 0.5671
0.324675 _0.12987 0.021645
```

Notice that the Savisky-Golay filter has negative coefficients for high enough degrees and hence could lead to out of range grayscales. Nonetheless, it is an averaging technique that may be desirable because it does not blur edges as the Gaussian filter does. In fact, the edges are somewhat enhanced. Figure 3 shows the result of applying a Savitsky-Golay filter using a least square quadratic fit to 9 pixel values nearest each point. Since this is a multiplicative filter, it can be applied either in a 2-dimensional pass or in two 1-dimensional passes. For example, we would obtain the grayscales in the figure, `f3`, using the following, where `clamp` forces the values to be legitimate grayscales values between 0 and 255.

```
   clamp=:0&>.@(255&<.)

   f3=:clamp (2 sav_gol 9)MFilt1d2 b
```

In Figure 3, notice that the dust and texture are diminished, but that the edges are much clearer than for the Gaussian filter—however, some blurring is apparent.

Rank Order Based Filters

Another type of local filter is based upon taking median (or some other rank order) values from a neighborhood of each pixel. The idea is that outlying values will be discarded; if most of the pixels near a certain pixel have a certain value, then that is the value that will be used. Multiplicative filters tend to introduce many more intermediate gray pixels as they blur the image, but rank based techniques use actual values appearing in the image. These rank based filters are generally computationally involved since each 2-dimensional neighborhood needs to be ordered. We implement this below with the adverb `medianf` whose left argument gives the neighborhood size. Note in the example below, each 3 by 3 neighborhood is replaced by its median element.

```
   medianf=: 1 : 0
(2#m.)"_ (<.-:*:m.)&{@:(/:~)@,;._3 ]
)

  ]b=.?.2+i.5 _5
 0  3  1  1  0
 0  6  6  7  2
 8 12  0  0  6
14  0  7  1  7
17 14 22 19 11

   3 medianf b
 3 3 1
 6 6 6
12 7 7

   f4=:3 medianf b
```

Figure 4 shows the result of the median filter on 3 by 3 neighborhoods. Notice the edges are relatively clear and the texture and dust are greatly diminished.

Many variations on the median filter can be considered. A slightly non-intuitive rank filter known as the hybrid median technique uses 5 by 5 neighborhoods and computes three quanities: the median of the 9 pixels forming the two diagonals in the neighborhood, the median of the 9 pixels in the same row and column as the center, and the central pixel value. Then the median of those three values is computed and utilized. This is implemented below and the result is shown in Figure 5. Here the edges are also good, but some of the defects remain visible. They can be essentially removed by a second application of the hybrid median method, but that introduces more blurring than the ordinary median filter.

```
   hmedianf=:3 : 0
med3=. 1&{@:(/:~)
med9=. 4&{@:(/:~)
x=.0 4 6 8 12 16 18 20 24&{
p=.2 7 10 11 12 13 14 17 22&{
5 5 med3@:(med9@:x,med9@:p,12&{)@:,;._3 y.
)

   f5=:hmedianf b
```

As we mentioned at the outset, our study of these filters was motivated by our desire to correct defects in scanned images. We actually corrected the images that we used by running a filter and then selecting a threshold to distinguish black from white. On each 2 by 2 neighborhood we took the item with position 2 in the ordered list of the 4 entries in the neighborhood and tested whether that value exceeded 180. If so, the pixel was white; otherwise, it was black. We designed this filter in an ad hoc way after looking at the values appearing in the textured defective portion of one image. Since then, we have learned that rank order filters in general, and median filters in particular, are valuable, known tools. Figure 6 shows the application of our ad hoc "snowflake cleaning" filter.

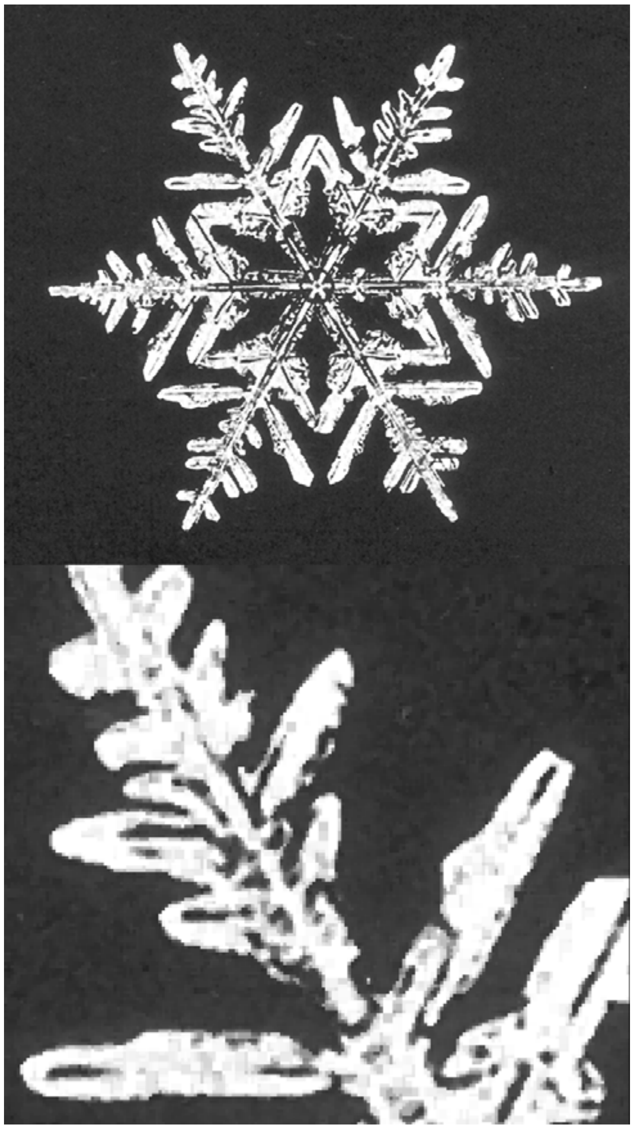
Figure 4. A Median Filter

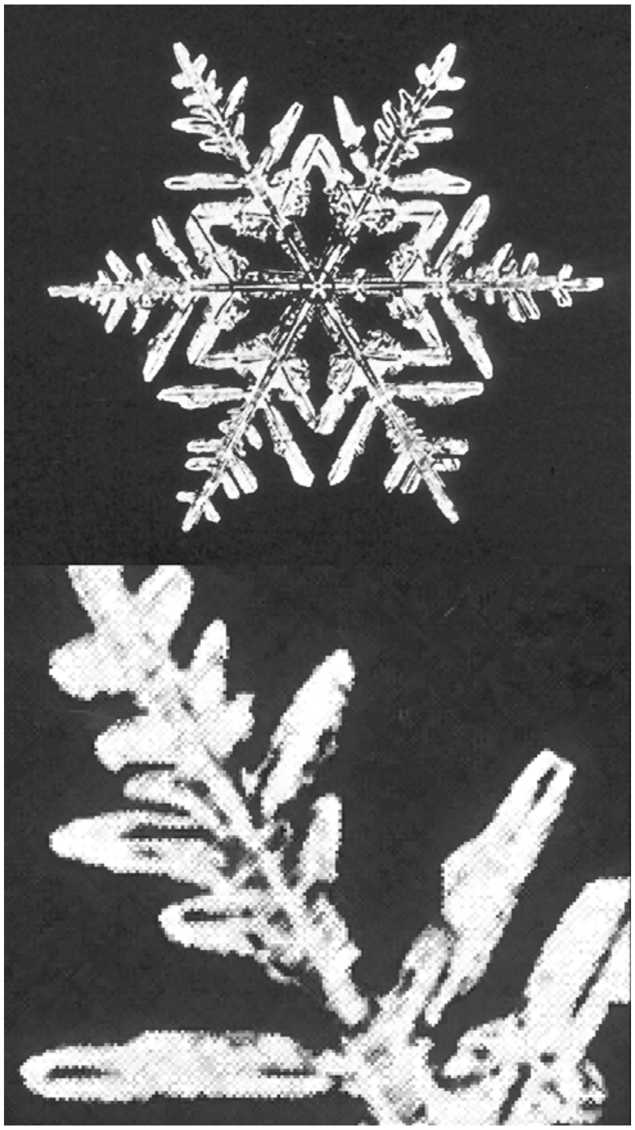
Figure 5. The Hybrid-Median Filter

```
   sf_filt=:3 : 0
255*180<2 2((2&{)@:(/:~)@:,);._3 y.
)

   f6=:sf_filt b
```

Notice the image in Figure 6 is different from the others due to the threshold used to create the black and white image. The edges are crisp, but some of the dust remains visible and some of the interior detail may have been lost.

Maximum Likelihood Filter

The maximum likelihood filter attempts to take advantage of the idea that an image has a certain "correct" value in each neighborhood and the image may be viewed as a corrupted version of the true image. By identifying values which deviate little from neighbors, we have found estimates for the correct values, and avoided outlying values arising from the corruption.

In particular, consider the 5 by 5 neighborhoods surrounding each pixel. Each 5 by 5 neighborhood contains nine 3 by 3 sub-neighborhoods. Compute the sum of the square of the deviations of the entries in the 3 by 3 neighborhoods from their central value, and

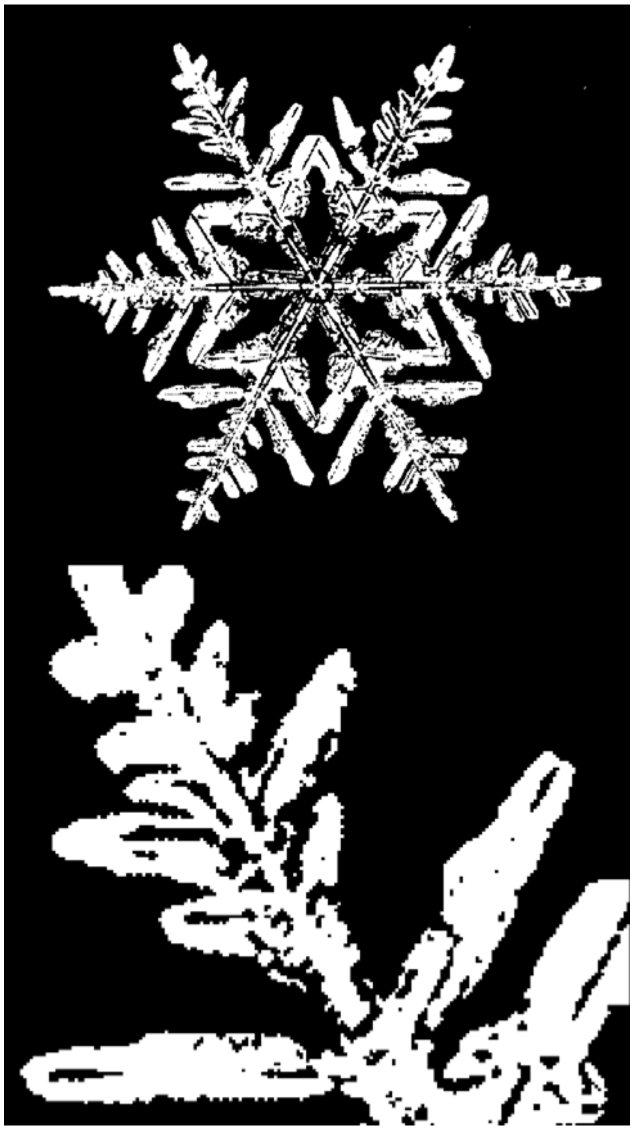
Figure 6. A Rank Order Filter and Threshold

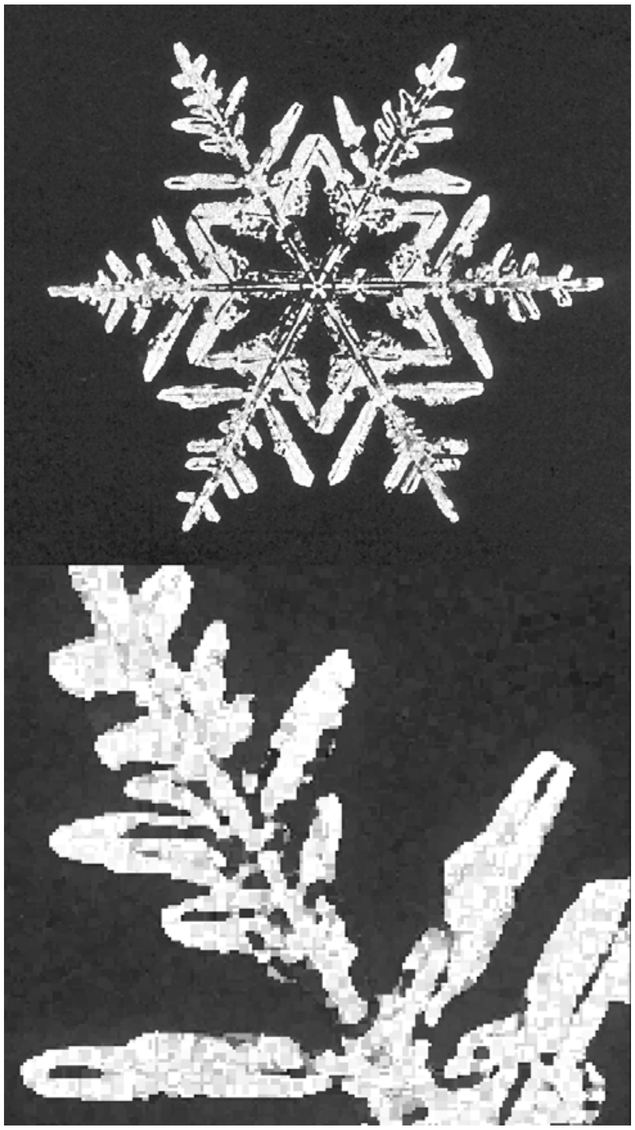
Figure 7. The Maximum Likelihood Filter

select the central value that corresponds to the minimal sum. This means that each pixel is replaced either by its own value, or by the value of an immediate neighbor. When a 3 by 3 neighborhood has small deviation, its central value is likely to be chosen for each of the 5 by 5 neighborhoods for which it is a sub-neighborhood. Thus, there should be a tendency for values to clump.

```
   maxlike=:3 : 0
ic=.,0 5 10+/6 7 8
i9=.,/3 3 ,;._3 i.5 5
lfilt=.({~ [: {&ic@:{.@:/: [:(+/)@:*:"1
i9&{ - ic&{)@:,  NB. one line
5 5 lfilt;._3 y.
)

   f7=:maxlike b
```

Figure 7 shows the result of the maximal likelihood filter. Notice the edges are relatively clear and the texture and dust are greatly diminished, but there is noticeable posterization; that is, clumps of similar values appear.

Conclusion

We have seen that image filters are easy to implement on grayscale images in J. Multiplicative filters may be implemented efficiently using two 1-dimensional passes. The Savitsky-Golay filter is a multiplicative filter that does not blur as much as the more classical versions. Median and other rank based techniques are more computationally involved, but can often be used to remove defects with little blurring. While none of the techniques is suitable for all purposes, these filters are valuable tools for removing defects from images.

Color Images

Correcting defects in a color image is significantly more difficult than correcting a grayscale image. That is because color images have three color dimensions and most of the filters that we have discussed apply only on one color dimension, which risks "desynchronizing" the changes. That is, if the image is a typical "true color" computer image, its colors are given as RGB triples. We can apply filters to each of the red, green and blue color planes. However, different behavior would usually occur in different planes so that color shifts, and other distortions, are a serious problem.

It is often better to change to an alternate color space and apply the filters only on the components that require correction. For example, one could change to HSI (hue, saturation and intensity) color space and apply a filter on the intensity plane, leaving color and saturation unchanged. We plan to discuss color spaces in a future column. At that point, we can discuss correcting defects in color images more specifically.

References

[1] W. A. Bentley and W. J. Humphreys, *Snow Crystals*, McGraw-Hill Book Company, 1931. Also, Dover Publications, New York, 1962.

[2] A. Coxe and C. Reiter, Boolean Hexagonal Automata, submitted.

[3] A. Coxe and C. Reiter, Fuzzy Hexagonal Automata and Snowflakes, submitted.

[4] GIMP, *http://www.gimp.org*

[5] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Addison-Wesley Publishing Co., Reading, 1992.

[6] C. A. Reiter, *Fractals, Visualization and J, 2nd Edition*, Jsoftware, Inc., Toronto, 2000.

[7] C. Reiter, With J: Fast Fourier Transforms and Removing Motion Blur, *APL Quote Quad*, 31 1 (2000) 16-18.

[8] C. A. Reiter, J Quote Quad materials, *http://www.lafayette.edu/~reiterc/j/withj/index.html*.

[9] J. C. Russ, *The Image Processing Handbook*, 4th edition, Boca Raton, CRC Press LCC, 2002.