

2. Image Filtering

2.1. Introduction

Convolution provides the basis for many fundamental functions for both image and signal processing. Now that the basics of convolution have been presented, it is time to put this knowledge to use. Most specifically to image processing, convolution plays a crucial role in image filtering. From the basic mean filter, to the more complex Gaussian filter, convolution allows key operations to function. This is thanks to the unique mathematical nature of the convolution operation.

Image filtering makes possible several useful tasks in image processing. A filter can be applied to reduce the amount of unwanted noise in a particular image. Another type of filter can be used to reverse the effects of blurring on a particular picture. Yet another filter can be applied to soften or darken the color and brightness levels of an image. Although each of these functions produces a different result, they all rely on convolution to carry out their specified tasks.

Before the idea of image filtering can be fully understood, it is necessary to first grasp the basics and comprehend the underlying operations being performed during each procedure. Afterwards, this knowledge can then be used to understand the more complex utilizations of image processing. Therefore, due to the nature of the discussion in Part 1 concerning convolution, this discussion will only cover basic image filtering using 1-dimensional images, known as skyline images.

2.2. The Basic Image Filter

2.2.1 Goal and Function

All image filters are based on one basic design, and have three main objectives: noise suppression, smoothing, and filtering. When an image filter is applied to an image I corrupted by noise n , the goal of that filter is to eliminate n altogether without altering I significantly. Keep in mind that this goal is two-fold. The first portion of the goal is to “eliminate n altogether.” Depending on the approach to this problem, this can be a trivial or tricky procedure. Logic would dictate that in order to remove *all* the noise in I , it would be simplest to just lighten, darken, or blend I to the point that all the noise is removed. This effectively removes the noise in the image, and the problem is solved; a trivial solution, to a seemingly trivial problem. However, this does not fully complete the task at hand. Recall that the second portion of the goal of an image filter is to perform the aforementioned operation “without altering I significantly.” The trivial solution presented fails this goal. Now the seemingly trivial solution is clearly insufficient for the problem statement. And thus, the problem that seemed trivial has now become quite difficult. Now comes the fundamental question concerning image filtering: is it possible to “eliminate n altogether without altering I significantly?” The simple answer is “no.” Removing n from an image I is not as simple as cleaning bugs off a windshield, or buffing out fingerprints off a glass window. Due to the nature of convolution, any noise that permeates I in most cases will never be fully removed. It is possible to remove most of n , but not all of n . Even if n is removed to such a degree that n becomes imperceptible, it will still be present in the resulting image. Speaking mathematically,

even if the inverse of the convolution is applied to the resulting function, the inverted function will not be *exactly* the same as the original function.

Because of this limitation in image filtering, it is necessary to change the original goal to better describe the problem situation: “when an image filter is applied to an image I corrupted by noise n , the goal of that filter is to attenuate n as much as possible (or remove it altogether) without altering I significantly.” This goal is much more attainable than the previous. So now the question becomes: is it possible to “attenuate n as much as possible without altering I significantly?” The simple answer is “yes.” However, the resulting filtered image may not closely resemble the original image I . It may have the same general appearance as I , but differences may be very apparent between the two. This does still fulfill the second portion of the goal, since the basic properties of I will still be maintained. But do keep in mind that filtering is not perfect, and the results will therefore not be as perfect as is sometimes desired.

2.2.2 Definition

The basic (and most common) technique used to filter an image is known as *linear filtering*. A linear filter convolves the image with a constant matrix, called a *mask* or *kernel*. The term *linear* in this case is due to the type of convolution that the filter is performing. Since the image is being convolved with a constant kernel, this models space- and time-invariant linear systems. In this aspect (consistent also with signal processing), the kernel can be viewed as the *impulse response* of the filter. Linear filters can be applied to both single- and multi-dimensional images. The basic idea of each filter is the same, with the multi-dimensional filter performing the filter operation in each dimension specified.

The basic two-dimensional linear filter algorithm is as follows:

Let I be a $N \times M$ image, m an odd number smaller than both N and M , and A the kernel of a linear filter, that is a $m \times m$ mask. The filtered version I_A of I at each pixel (i, j) is given by the discrete convolution:

$$2.1 \quad I_A(i, j) = I * A = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} A(h, k) I(i-h, j-k),$$

where $*$ indicates discrete convolution, and $m/2$ integer division (e.g., $3/2 = 1$).

Recall the formula that was used to describe convolution back in Part 1. Compare the appearance of Formula 1.4 to that of Formula 2.1 above:

$$1.4 \quad f_g(i) = f * g = \sum_{h=-m/2}^{m/2} g(h) f(i-h),$$

$$2.1 \quad I_A(i, j) = I * A = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} A(h, k) I(i-h, j-k)$$

A linear filter is simply the convolution of the image with the kernel. The structure of Formula 1.4 is better suited for this discussion, since the subject is single-dimensional images. Therefore, it is necessary to modify Formula 2.1 in order to operate with single-dimensional images:

Let I be an image of size N , m an odd number smaller than N , and A the kernel of a linear filter, that is a mask of size m . The filtered version I_A of I at each pixel (i) is given by the discrete convolution:

$$2.2 \quad I_A(i) = I * A = \sum_{h=-m/2}^{m/2} A(h) I(i-h)$$

Now the relationship between the linear filter and the basic convolution formula is more apparent. Formula 2.2 will provide the basis for further discussion on image filtering.

When this filter is applied to an image, each value $I(i)$ is replaced with a weighted sum of I values in a neighborhood of (i) . The weights used to compute each weighted sum in I_A comes from each entry in the kernel. The operations that occur here are precisely those that occur in a convolution between two functions (or two matrices). This should be very apparent from analyzing and comparing Formulae 2.2 and 1.4.

2.3. A Working Example: The Mean Filter

2.3.1 Description

The simplest filter to implement is known as the *mean filter*. The mean filter performs *average smoothing* on an image. The name perfectly describes the function of this filter. Each pixel in I is replaced with the mean of the pixels that surround it. Essentially, noise is blended into the rest of the picture. A filter that performs average smoothing must use a kernel with all entries being non-negative. For example, if a kernel A was used with $m = 3$:

$$A_{avg} = 1/3 [1 \quad 1 \quad 1]$$

Additionally, it is absolutely necessary for all the entries in the kernel to have a sum of one. If the sum is not equal to one, then the kernel must be divided by the sum of the entries (hence the multiplication of the $1/3$). If this requirement is not met, then the filtered image will become brighter than the original image, along with undergoing the specified filtering effect. This limitation on the mean filter fulfills the second portion of the image filtering goal. This filter is effective at attenuating noise because averaging removes small variations. The effect is identical to that of averaging a set of data to help reduce the effect of outliers. In a two-dimensional mean filter, the effect of averaging m^2 noisy values around pixel (i, j) divides the standard deviation of the noise by $\sqrt{m^2} = m$.

2.3.2 Implementation and Example

2.3.2.1 Skyline Image

The most basic image that can be used to demonstrate a one-dimensional mean filter is a skyline image. A skyline image is a one-dimensional image that resembles the

well a filter preserves these features in the final filtered image will demonstrate its mode of operation, as well as prove its effectiveness.

2.3.2.2 Mean Filter #1

To demonstrate the mean filter and its effect on the skyline image, a basic kernel that matches our previous description is chosen: $A = [1 \ 1 \ 1] / 3$. The filtered image (blue) that occurs is displayed below, and again on top of the original image (black):

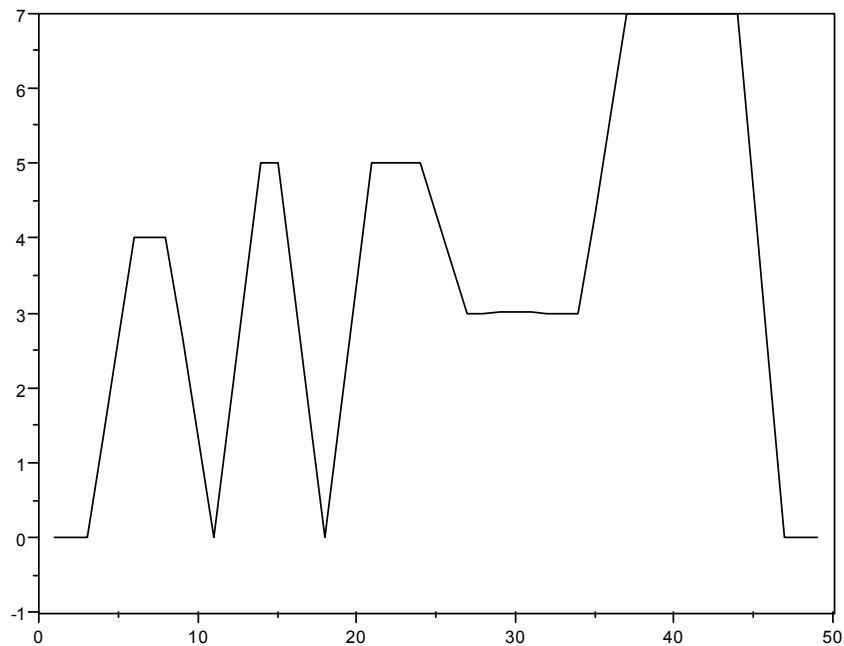


Image 2: Filtered Skyline Image Using $A = [1 \ 1 \ 1] / 3$

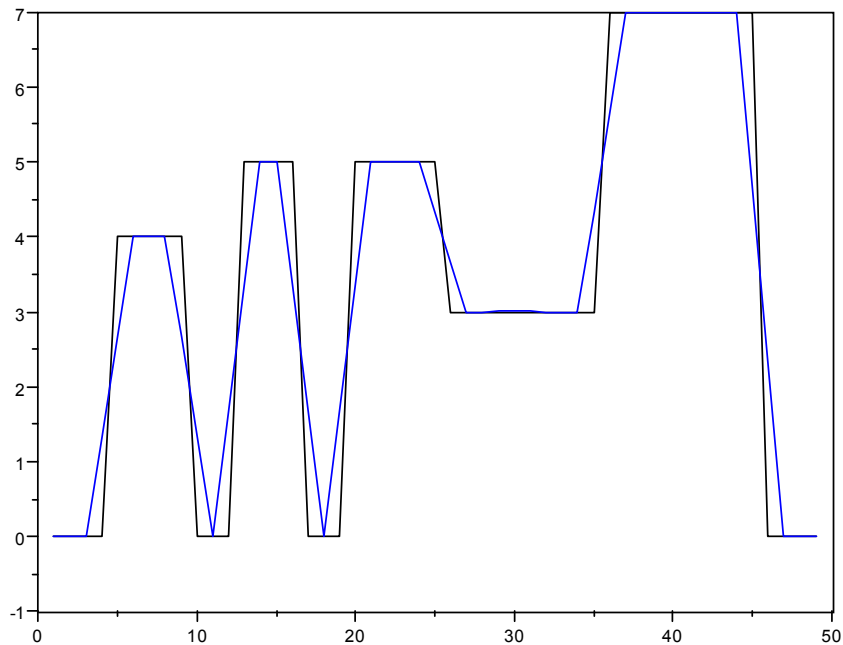


Image 3: Original Skyline Image (Black), Filtered Skyline Image (Blue)

There are clear differences between the two images. The filtered image does not preserve the same shape as the original image, however the overall outline is still maintained. Therefore the original goal of not “altering I significantly” has been maintained, but at the loss of some image quality. Upon close observation, it is clear that the filter softened up some of the sharper angles of the original image. These are most apparent at the floors and roofs of the “buildings” in the image. Additionally, the sides of the buildings are now more angled, and take on a more trapezoidal appearance. In terms of brightness and color in a multi-dimensional image, the filter has effectively softened the brightness of the entire image. Some of the points that were darker in the original image (i.e., towards the bottom of the image) are now brighter (i.e., moved higher in the image); likewise, some of the points that were brighter in the original image (i.e., towards the top of the image) are now darker (i.e., moved lower in the image). The filter did not

affect all the elements in the original image. Those points that made up the longer rooftops and floors of the buildings were maintained. It was the points that made up the walls and corners of the buildings that were modified most. The effect of this basic filter is best described as a *blur*. The overall softening and smoothing of the colors creates a soft but blurry image.

2.3.2.3 Mean Filter #2

This blurring effect is still apparent when a stronger mean filter is used. Even though the sum of the elements in the kernel will still remain one, the size of the elements in the kernel can be of any value desired. This is where the overall strength of the filter is observed. For the second blurring of the skyline, a new kernel is chosen: $A = [1 \ 4 \ 1]/6$. The sum of the elements in the kernel is six ($1 + 4 + 1 = 6$). Therefore, the kernel must be divided by six in order to prevent the filtered image becoming dramatically brighter than the original image. The result of the new kernel is demonstrated below, both alone and alongside the original image:

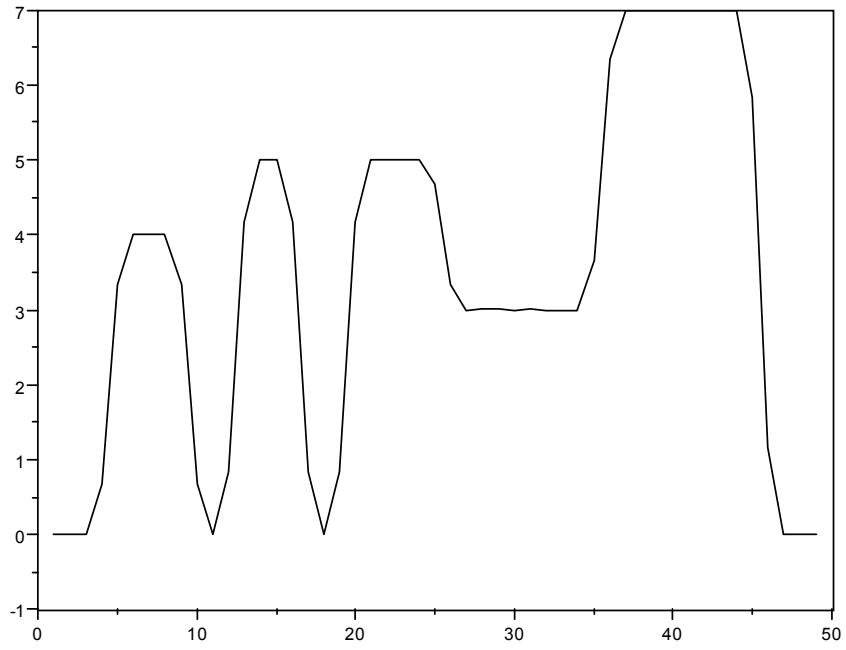


Image 4: Filtered Skyline Image Using $A = [1 \ 4 \ 1] / 6$

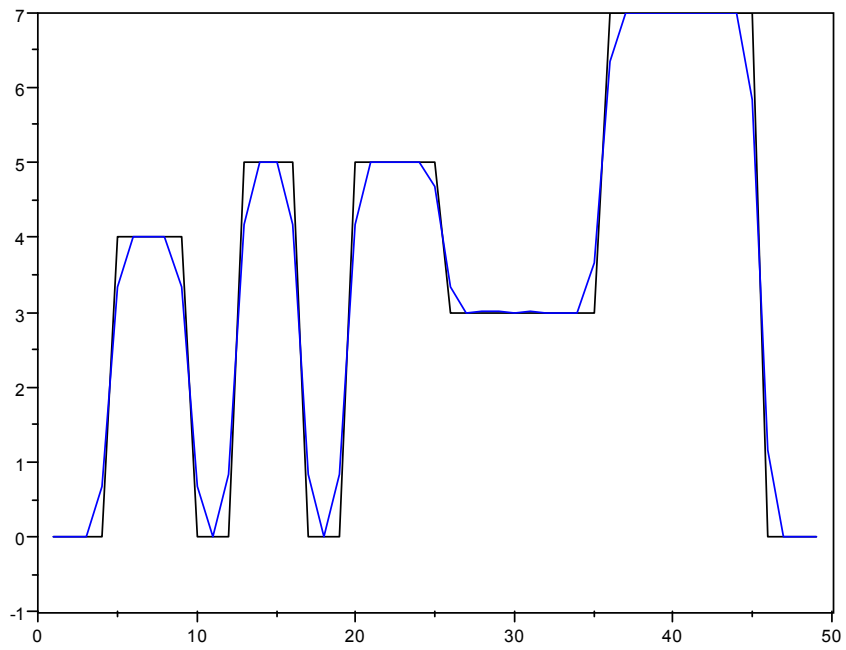


Image 5: Filtered Skyline Image (Blue), Original Skyline Image (Black)

Once again, there are clear differences in the two images. But the differences are not as dramatic as those present in the first trial. The following image compares the two filtered results side by side:

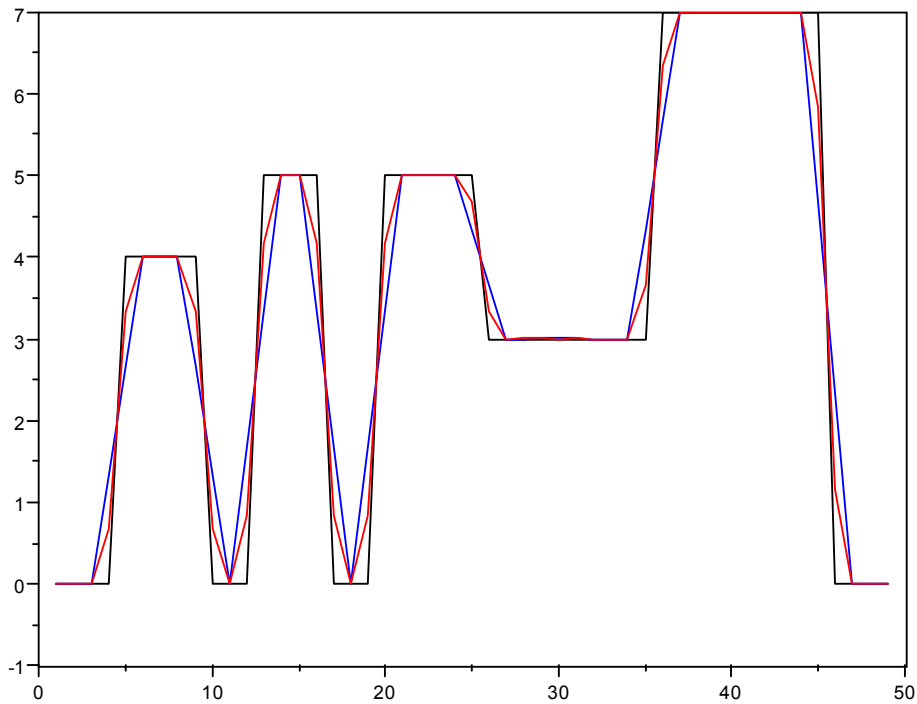


Image 6: Original Skyline Image (Black), Filter #1 (Blue), Filter #2 (Red)

Filter #2 preserves the original image with superior quality in some areas, and falls short in other areas. Filter #1 created angled walls for the sides of the building, which deviate from the inherent shape present in the original image. Filter #2, on the other hand, creates walls that are nearly identical to those found in the original image. It is here that Filter #2 excels at achieving the original image filter goal. However, this filter begins to lose image data around the corners and floors of the original image. In these areas, Filter #1 is the superior performer. The unique property that Filter #2 inserts into the resulting

image is an extra line segment between floors and walls. The walls now have one extra slope that connects them to the floor and the roof. These features were not part of the original image at all. Filter #1 does not insert this extra feature into the resulting image. Unfortunately, the result that Filter #1 does provide is not all that close to the original image. Despite the downfalls of each filter, the image Filter #2 produces matches more closely to the original image in more areas than Filter #1.

2.3.2.4 Mean Filter #3

The effects of a mean filter can be further investigated with multiple passes of a weak filter. Recall the nature of convolution, and its effect on image filtering. When convolving two matrices, the result is dependent on the magnitude of the matrices being convolved. As profound as this may or may not sound, the result of a convolution is solely based on the matrices interacting in the operation. Unlike scalar multiplication, a matrix cannot be convolved multiple times by a small matrix will not achieve the same result as convolving that same matrix with a larger matrix. For example:

$$16 * 16 = 256$$

$$16 * 4 * 4 = 256$$

This same property does not hold for convolution. An image filtered using a weaker filter repeatedly will result in too much filtering effect on the original image, so much that a great deal of the information in the original image is lost. Doing so fails the second portion of the goal of an image filter. The following images demonstrate Mean Filter #1 on the original image, where $A = [1 \ 1 \ 1] / 3$, for one, two, and three passes with the filter:

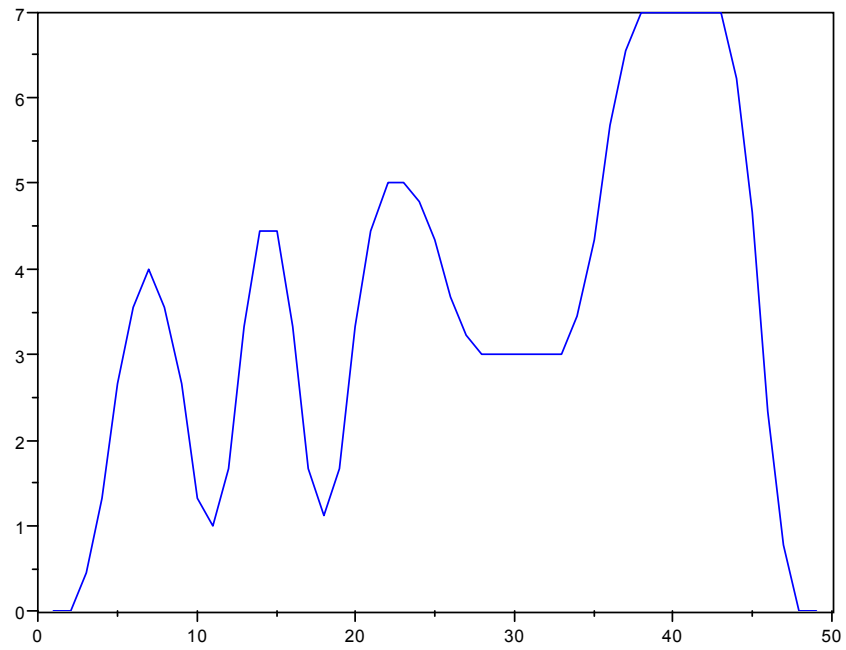


Image 7: Filtered Skyline Image Using $A = [1 \ 1 \ 1] / 3$, 2 Passes

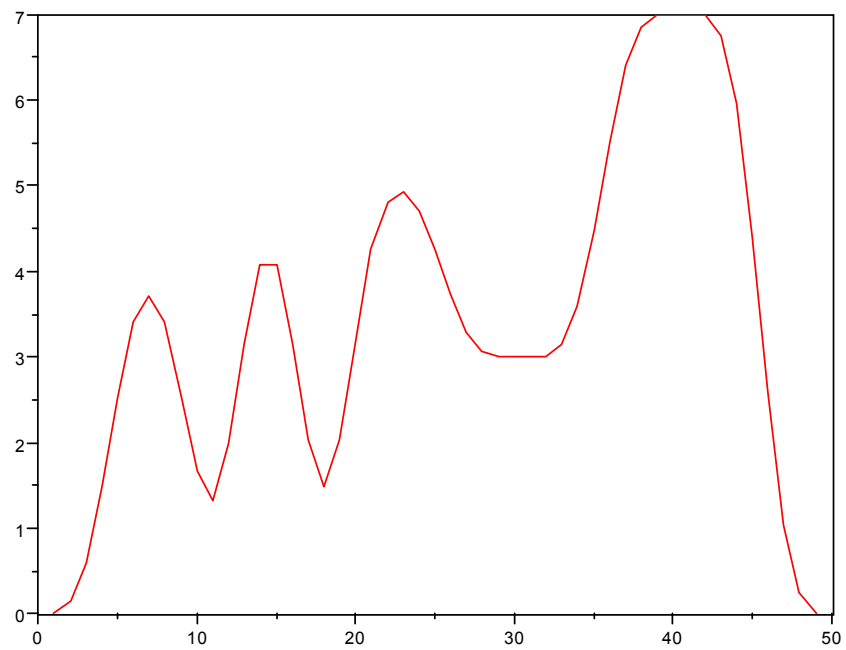
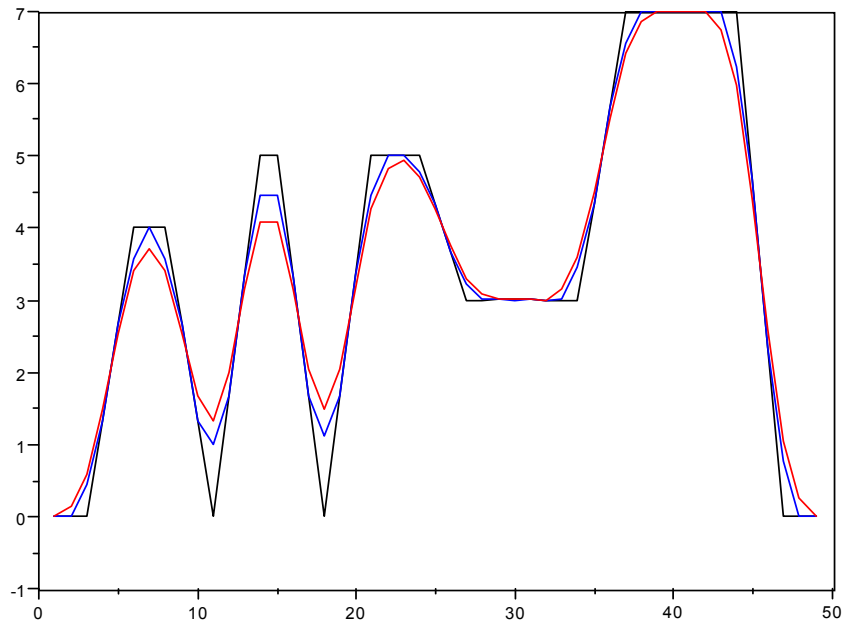
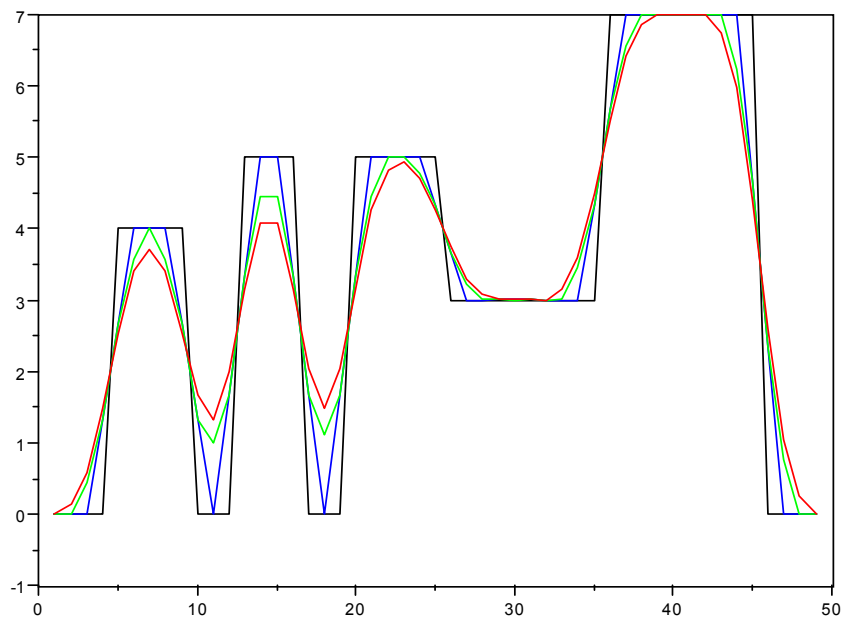


Image 8: Filtered Skyline Image Using $A = [1 \ 1 \ 1] / 3$, 3 Passes



**Image 9: Filtered Skyline Image Using Passes With $A = [1 \ 1 \ 1] / 3$
Black = 1 Pass, Blue = 2 Passes, Red = 3 Passes**



**Image 10: Filtered Skyline Image Using Passes With $A = [1 \ 1 \ 1] / 3$
Black = Original, Blue = 1 Pass, Green = 2 Passes, Red = 3 Passes**

The more the original image is filtered, the more dramatic the blurring effect becomes. With each pass, the edges of the buildings become more rounded, the walls become more angled, and the roofs become more pointed. Additionally, the floors of the center two gaps become higher and higher. In terms of colors and brightness, the image becomes softer in brightness overall as the filter is applied repeatedly.

What is interesting to note is the loss in image data at the roofs and floors of the image. The first pass of the filter maintains the flat shape of the roofs. However, that flat shape is lost once the image is filtered the second and third times. It is here that the flat roofs start to become rounded (second pass), and then pointed (third pass). Of particular interest as well is the overall flattening down of the buildings as a whole. With each successive pass of the filter, the roof line drops down lower, the walls become more angled and contain more line segments, and the base of the walls become wider. Now, the final filtered image begins to resemble less and less the original image. The skyline image is beginning to more closely resemble a landscape rather than a cityscape. The buildings are now transforming into gradually rising mountains and peaks. Even though the original image has further morphed, the inherent result of the metamorphosis is the same: the resulting filtered image features an overall softening of the colors. Points that were darker in the original image are now brighter, and those points that were brighter in the original image are now darker. The blurring effect that has resulted from repeated filtering is now more pronounced, as the curved corners, angled sides, and shorter heights would indicate. If the original image were to be run through this filter (or any filter, for that matter) infinitely many times, the resulting image would most likely continue to flatten out to the point where the resulting image would become a flat line.

2.3.2.5 Mean Filter #4

The filters described thus far have implemented using 3-pixel kernels of varying strengths. Kernels do not have to necessarily be of this particular size. Theoretically any kernel size can be chosen to filter an image. Typically, 3-, 5-, and 7-pixel kernels are chosen (for two-dimensional filtering, 3×3 , 5×5 , and 7×7 kernels would be chosen). The size of the kernel—both in the number of elements used and the size of each element—will determine the overall affect that the filter has on the final image. Previous filters demonstrated the effects of using kernels of different magnitude. The next filters will demonstrate the effects of using kernels of different size, both in 5- and 7-pixel varieties. The fourth filter used utilizes a 5-pixel kernel: $A = [1 \ 1 \ 1 \ 1 \ 1] / 5$. What follows is the filtered skyline image, and a comparison of this image to the original skyline and the first filtered skyline:

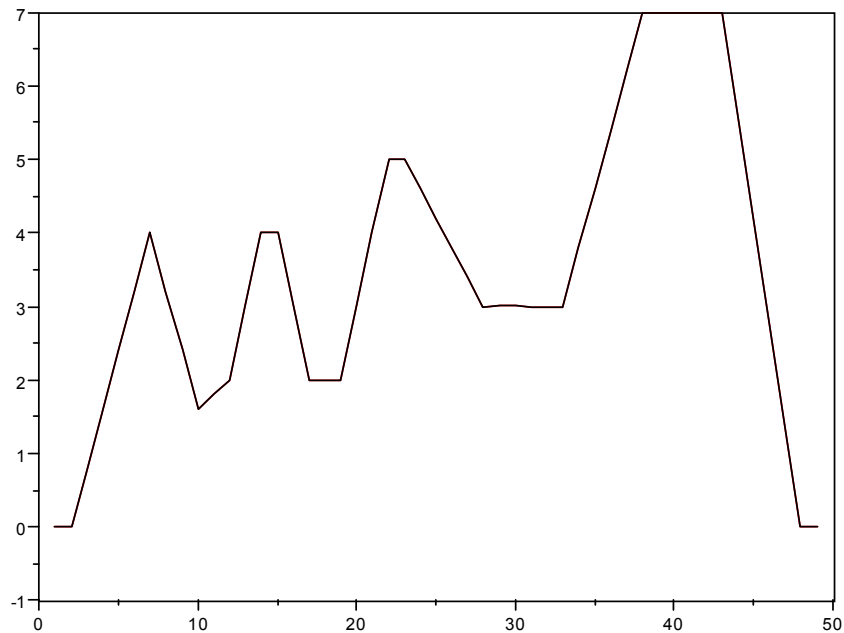


Image 11: Filtered Skyline Image Using $A = [1 \ 1 \ 1 \ 1 \ 1] / 5$

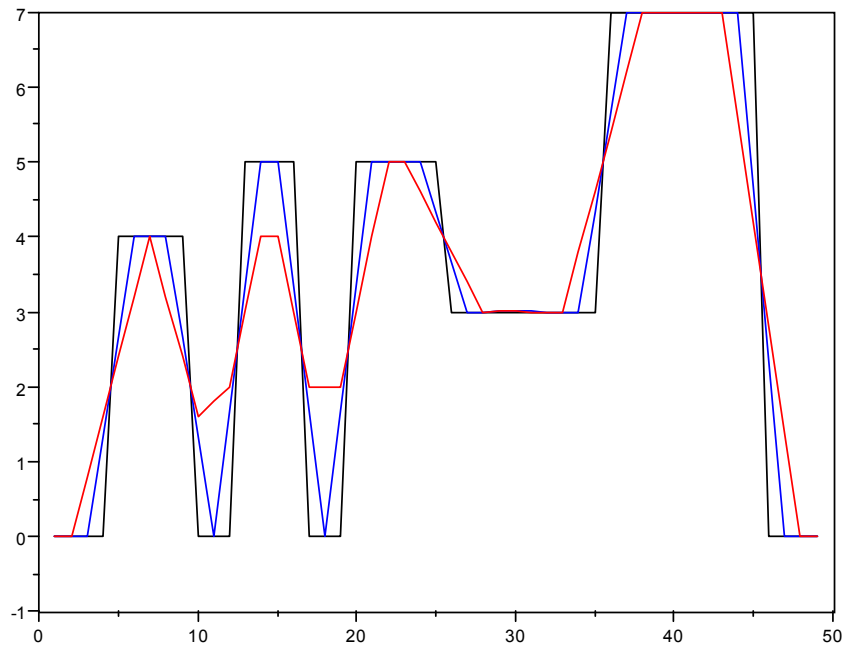


Image 12: Original Skyline Image (Black), Filter 1 (Blue), Filter 4 (Red)

The differences between the basic 3- and 5-pixel kernels are very apparent here. The same exact values are used in both the 3- and 5-pixel kernels, yet the results are dramatically different. The single-pass 3-pixel filters maintained many of the general features of the buildings, even after modifying the values in the kernel. The single-pass 5-pixel filter, on the other hand, does not maintain many of the general features of the buildings. The appearance of this single-pass 5-pixel filter is similar to that of the multiple-pass 3-pixel filter, in that the buildings are quite pointed, with highly angled walls, and small or nonexistent roofs. The angles of the walls produced by this 5-pixel filter approach those produced by the multiple-pass 3-pixel filters. Of particular interest are the first and second buildings. The first building, which originally had a long, flat rooftop, now possesses a very pointed roof. The floor that connects the first and second buildings is no longer flat, but now sloped. This behavior is nonexistent in the rest of the

image. The other floors are surprisingly flat, a deviation from the properties exhibited by the 3-pixel filters. Recall that all the 3-pixel filters produced pointed floors. In this aspect here, the 5-pixel filter is superior to the 3-pixel filter.

Even though the 5-pixel filter excels in particular areas, the overall image that it produces is not as close of a match to the original image than the 3-pixel filter. The walls produced by the 3-pixel filter match those in the original image much more closely. The rooftops are much wider than those in the 5-pixel filtered image. Additionally, the floors produced by the 3-pixel filter actually meet at the same low points as in the original image. Although the floors produced by the 5-pixel filter are flat, they are much higher than the floors in the original image. The effects of this filter on the overall image will be more noticeable than the effects created by the 3-pixel filter. The resulting image will have a higher degree of blurring, and will possess a softer level of brightness.

2.3.2.6 Mean Filter #5

Recall the image produced by Filter #2. The difference between Filters 1 and 2 were the values chosen for the kernel. The sum of the values in the kernel for Filter #2 was greater than the sum of the values in the kernel for Filter #1. The result of that change was most apparent in the walls of the buildings. The walls, for the most part, matched very closely to those in the original image. However, the walls now begin to show a slight curve near their bases. Additional line segments were inserted near the bottom of each wall, features not consistent with the original image.

The same changes will be performed using the 5-pixel filter, to explore the effects thereof. The kernel chosen for this filter was: $A = [1 \ 4 \ 10 \ 4 \ 1] / 20$. The following images demonstrate the filtered image, and the original and first 5-pixel filter used:

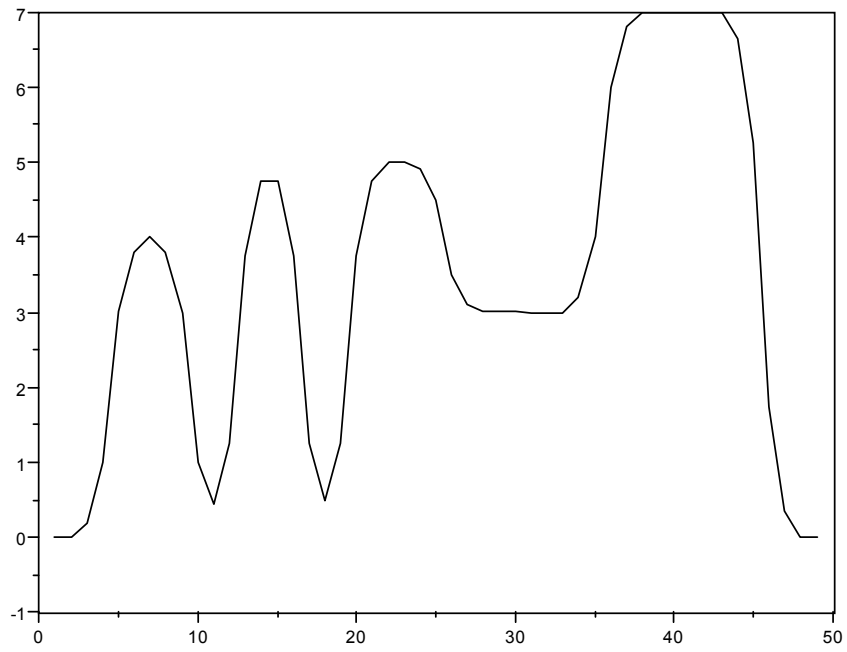


Image 13: Filtered Skyline Image Using $A = [1 \ 4 \ 10 \ 4 \ 1] / 20$

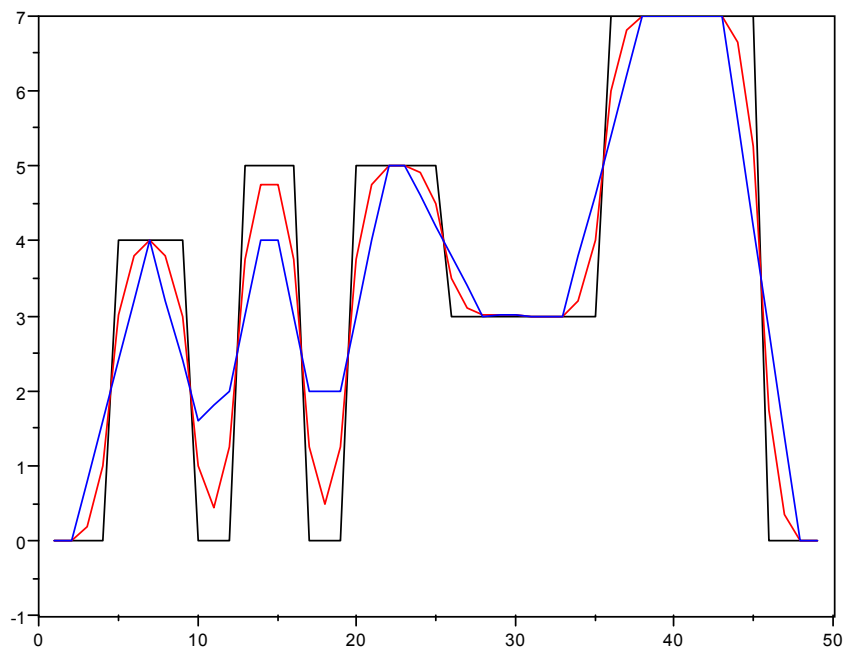


Image 14: Original Skyline Image (Black), Filter #4 (Blue), Filter #5 (Red)

The results of the changes in the kernel of the 5-pixel filter are similar to the results found when changing the kernel in the 3-pixel filter. As observed in the 3-pixel environment, the weaker 5-pixel filter produced highly angled walls that did not match those seen in the original image, whereas the stronger 5-pixel filter produced walls that closely matched those seen in the original image. Again, like the stronger 3-pixel filter, the stronger 5-pixel filter produced a slight curvature in the walls by inserting extra line segments in the walls near the base and roof. However, the stronger 5-pixel filter accentuates this effect much more strongly. The walls have two extra line segments at the base and roof, further enhancing the curving effect. Recall that the 3-pixel filter only inserted one of these line segments into each wall at the base and roof. Another difference between the strong 3- and 5-pixel filters is the level of the short floors in each. The 3-pixel filter produced short floors that touched the short floors in the original image. Those made by the 5-pixel filters are much higher than the original floors. Fortunately, they are lower than the floors produced by the weaker 5-pixel filter.

One additional property that is not consistent between the strong and weak 5-pixel filters is the appearance of the floors in each. The short floors created by the weak 5-pixel filter were nice and flat, consistent with the original image. The short floors made by the stronger 5-pixel filter, however, are pointed, just like those created by each 3-pixel filter. This results in loss of information in the filtered image.

Analyzing the resulting image as a whole reveals that the stronger 5-pixel filter produces an image that more closely matches the original image. Despite the fact that the floors are no longer flat, the base and roofs possess curvature, and the short floors do not touch the floors in the original image, the stronger 5-pixel filter matches the original

image in more areas than the weaker filter. This is most apparent in the walls produced by each. The angles of the walls of the stronger 5-pixel filter are nearly identical to those in the original image. Those in the weaker 5-pixel filter are a very crude approximation of the original walls, and do not match closely. Even though the curvature in the rooftops is not part of the original image, they do provide a closer approximation of the roof than the sharp points produced by the weak filter. The amount of rooftop area that matches the original image is the same in both the strong and weak filter. The difference between the two is the angle of the walls to the either side of the roof. The weak filter produces angles that are quite sharp to either side of the roof, which does not match close at all to the roof in the original image. The strong filter produces additional line segments that introduce less extreme angles. These extra line segments, coupled with the less extreme angles, provide a more accurate approximation to the rooftops, even though they are still not exact.

2.3.2.7 Mean Filter #6

The result of repeated filtering is very apparent, as demonstrated by the techniques outlined with Filter #3. Therefore, these effects will not be explored using 5-pixel filters. What is more important is to analyze once again the implications of using yet a larger kernel to filter the original skyline image. Instead of using a 3- or 5- pixel kernel, the next filter will implement a 7-pixel kernel similar to the weak 3- and 5-pixel filters: $A = [1\ 1\ 1\ 1\ 1\ 1\ 1] / 7$. The following images demonstrate this filter: images demonstrate the filtered image, and the original and first 5-pixel filter used:

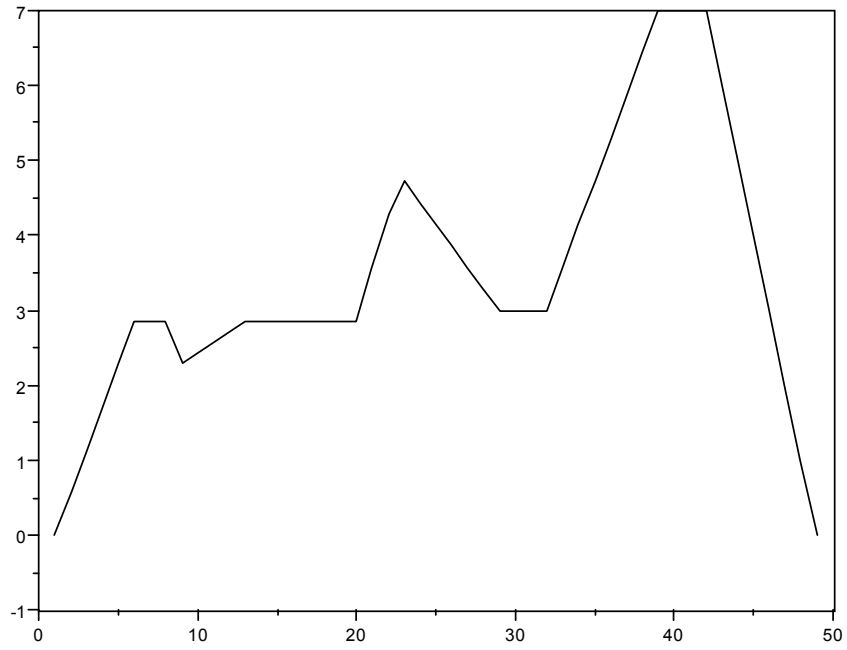


Image 15: Filtered Skyline Image Using $A = [1\ 1\ 1\ 1\ 1\ 1\ 1] / 7$

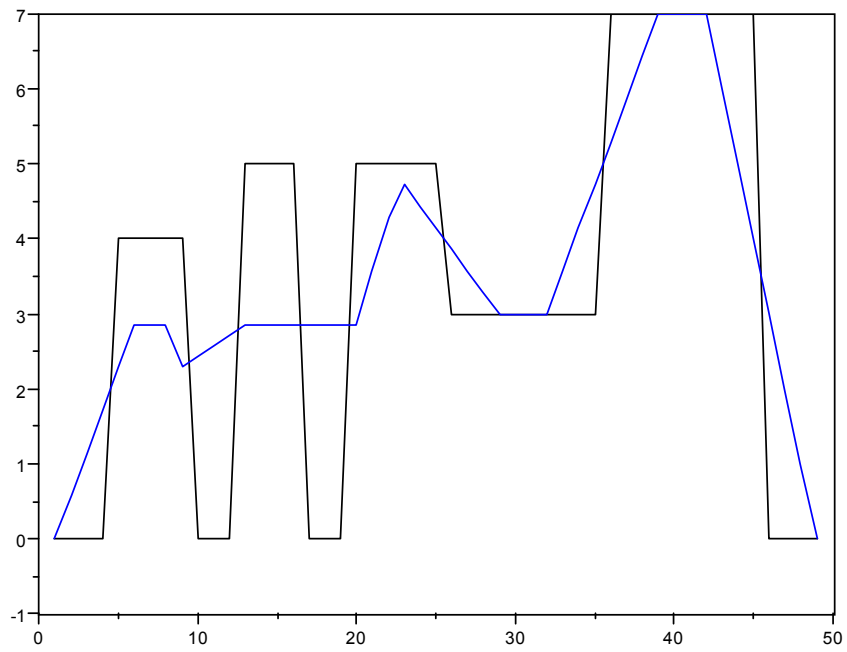


Image 16: Original Skyline Image (Black), 7-Pixel Filter (Blue)

It is apparent that this filter is on the verge of failing the second portion of the original image filtering goal. The filtered image does not resemble the original image at all. Large features have been entirely removed. The first two buildings no longer exist, except for a small bump representing the first building. The third building now exists only as a small peak. The last building, the tallest of the group, does possess part of its roof, but now has walls that are so highly angled that they no longer resemble the original walls whatsoever. What is most surprising is the total removal of the floor in most areas. The floor is nonexistent in the left side of the image. In fact, the floor and the roof of the leftmost buildings seem to have been averaged, and a plateau exists in that area. This is a rather crude and simple approximation of this portion of the image.

The image as a whole will possess vastly different brightness levels than the original image. Save for a few small sections, the filtered image will appear much darker than the original image. Most of the points in the filtered image are much lower than those in the original image, corresponding to a reduction in brightness levels. The only areas that will appear brighter are the floors. The rough approximations of the floors (represented by the highly sloped angles) are much higher than the original floors. Needless to say, a lot of information is lost when using this filter. It may be the case that this filter removes the noise that the original image may possess, but the filter also removes a great deal of the relevant information in the original image. It is because of this that this filter would not be recommended to use for attenuating noise in the original image. All of the previously examined filters—both single- and multiple-pass—perform superior to this filter. Once again, it is clear that adding pixels to the kernel has a great impact on the filtered image.

2.3.2.8 Mean Filter #7

The last filter examined was clearly not of much use in accurately attenuating noise. But for the sake of argument, it is necessary to examine a stronger 7-pixel filter. The values chosen for the stronger 7-pixel filter were conveniently from Pascal's triangle: $A = [1\ 6\ 15\ 20\ 15\ 6\ 1] / 66$. The following is the filtered result, along with a comparison between the original image and both 7-pixel filters:

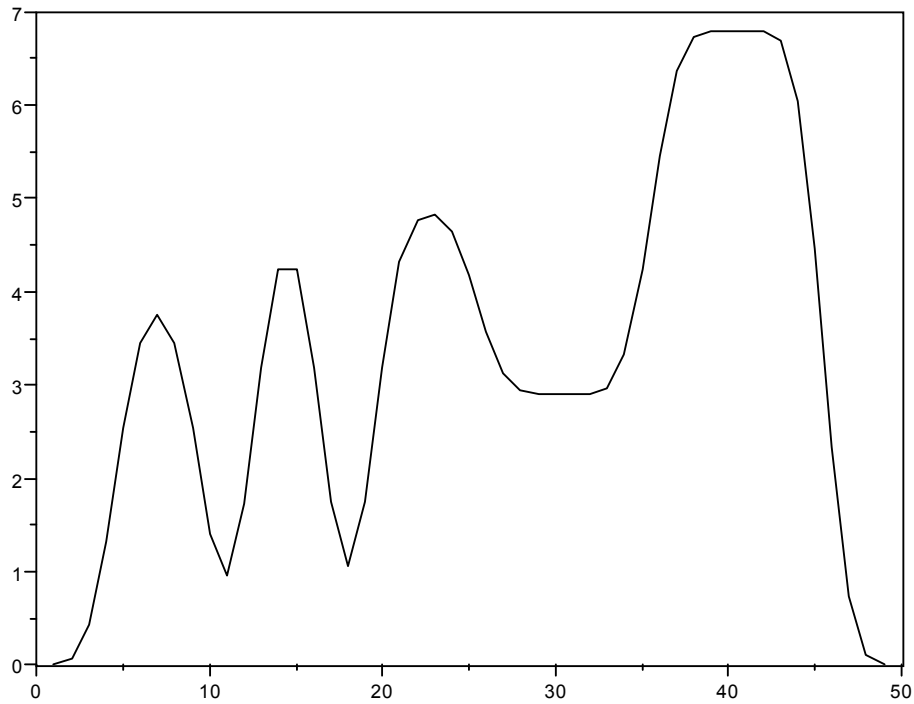


Image 17: Filtered Skyline Image Using $A = [1\ 6\ 15\ 20\ 15\ 6\ 1] / 66$

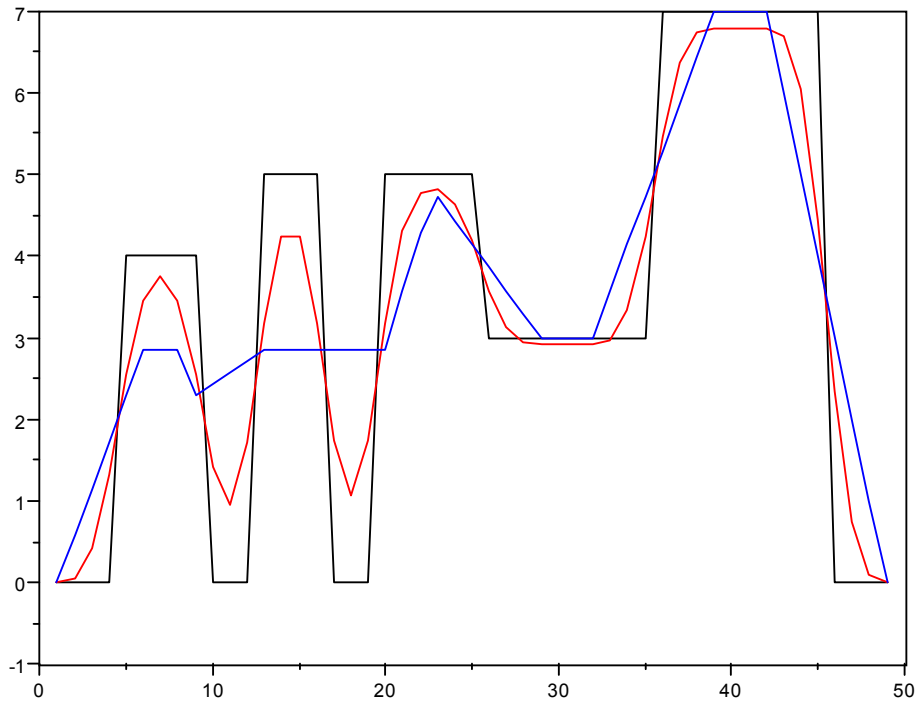
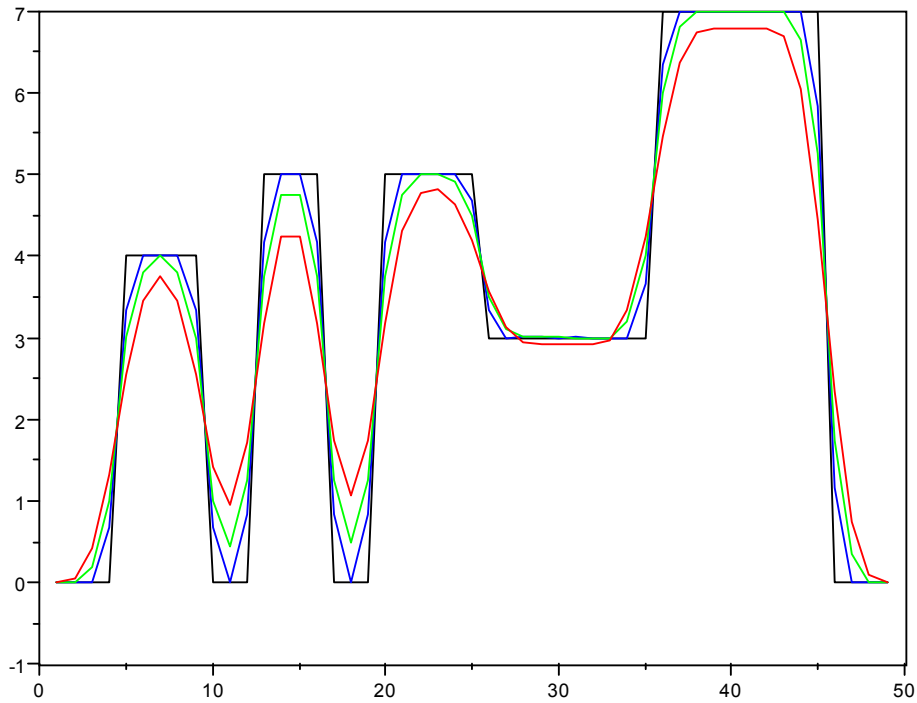


Image 18: Original Skyline Image (Black), Filter #6 (Blue), Filter #7 (Red) Image

As was the case with the 3- and 5-pixel filters, strengthening the 7-pixel filter resulted in an image that is a much more accurate approximation of the original image. The difference in image quality is most apparent in the 7-pixel filter. The image produced by the weak 7-pixel filter did not resemble the original image at all. The image produced by the stronger 7-pixel filter is a vast improvement. Many of the features of the original image are preserved in the filtered image.

When strengthening each size of filter described, each filter produced images that all have the same inherent properties: curvature at the base and the rooftops, walls that closely match those in the original image, pointed floors that are higher than those in the original image, and a general appearance of a hilly landscape rather than a uniform cityscape. The following image demonstrates the stronger 3-, 5-, and 7-pixel filters:



**Image 19: Original Skyline Image (Black),
Filter #2 (Blue), Filter #5 (Green), Filter #7 (Red)**

The effects of using a kernel with more pixels becomes apparent. As more pixels are added to the kernel, the resulting filtered image loses more and more data from the original image. With each increasing step in kernel size, the features of the filtered image become more flat. The angles of the walls deviate more and more from those in the original image. The bases of the walls begin to feature more curvature. The rooftops also feature more curvature, and start to sink lower in height. The floors begin to rise in height away from those in the original image.

Unique to the stronger 7-pixel filter is the long floor (or short rooftop) to the immediate right of the center of the image. The 3- and 5- pixel filters did a fairly good job of producing a flat surface for this area. The 7- pixel filter, on the other hand, added a

bit of curvature to that portion of the image. The result is similar to a sagging rooftop. The 7-pixel filter also produced more line segments in the walls, rooftops, and floors of the skyline. The result is a higher degree of curvature present in every feature of the filtered image.

The image quality of the stronger 7-pixel is superior to that of the weak 7-pixel filter. More of the original information is retained by the stronger filter, and is present in the resulting image. However, adding the extra pixels to the kernel did not improve image quality. In fact, there is a slight reduction in image quality. The stronger 5-pixel filter exhibited the same properties, but the effect is more apparent with the 7-pixel filter. All of the vertical features in the original image came out flatter in the filtered image. This is most noticeable in the curvature present, along with the less extreme angles of the walls. The floors are higher in the filtered image, and the roofs are lower. The bases of the walls sit farther away from the bases in the original image, which results in buildings that are wider than the originals. The hilly landscape effect is more apparent here as well, thanks to the aforementioned features of the new skyline.

2.4. Numerical Analysis

The comparisons between each type of filter on the skyline image were done in a purely graphical sense. If a numerical analysis is deemed necessary, the values that compose each image can be compared side-by-side to illustrate the effectiveness and efficiency of each filter.

The quantitative data can give more insight as to the appearance of each image presented in the previous section. The differences between each filtered image and the original skyline image become more apparent when the cold, hard numbers are analyzed and compared. Observing the results of these functions in a quantitative manner also provides more insight as to the function and behavior of convolution, as well as the accuracy of computers. Any values that are extremely small, that are on the order of 10^{-10} to 10^{-16} represent the accuracy of the machine used to calculate these values. Keep in mind that these values may change from machine to machine. Regardless of the machine used, any of the values within this range should be considered to be an approximation of the value zero.

This data also gives insight into the accuracy of the SciLab `convol()` function. The `myconvol()` function was not used for this analysis because it is still in the beta testing stages. The SciLab `convol()` function, even though it was proven to be less accurate in some situations than the `myconvol()` function, is a more mature function that has undergone more testing than `myconvol()`. It was therefore considered to be the better function to use for this analysis. The same or similar results could be achieved using the `myconvol()` function, but using `convol()` ensures that no unforeseen bugs that might be present in `myconvol()` will effect any results.

The following list displays the values that constitute each image, as calculated on one particular machine:

Skyline:

column 1 to 11

! 0. 0. 0. 0. 4. 4. 4. 4. 4. 0. 0.!

column 12 to 22

! 0. 5. 5. 5. 5. 0. 0. 0. 5. 5. 5.!

column 23 to 33

! 5. 5. 5. 3. 3. 3. 3. 3. 3. 3. 3.!

column 34 to 44

! 3. 3. 7. 7. 7. 7. 7. 7. 7. 7. 7.!

column 45 to 49

! 7. 0. 0. 0. 0.!

Mean Filter #1:

column 1 to 14

! 4.441D-16 4.441D-16 4.441D-16 1.3333333 2.6666667 4. 4. 4.
2.6666667 1.3333333 -4.441D-16 1.6666667 3.3333333 5.!

column 15 to 31

! 5. 3.3333333 1.6666667 1.388D-16 1.6666667 3.3333333 5. 5.
5. 5. 4.3333333 3.6666667 3. 3. 3. 3. 3.!

column 32 to 48

! 3. 3. 3. 4.3333333 5.6666667 7. 7. 7. 7. 7. 7. 7. 7.
4.6666667 2.3333333 -4.441D-16 6.661D-16!

column 49

! 3.331D-16!

Mean Filter #2

column 1 to 15

! 1.110D-15 6.661D-16 0. 0.6666667 3.3333333 4. 4. 4.
3.3333333 0.6666667 -1.776D-15 0 .8333333 4.1666667 5. 5.!

column 16 to 32

! 4.1666667 0.8333333 2.776D-16 0.8333333 4.1666667 5. 5. 5.
5. 4.6666667 3.3333333 3. 3. 3. 3. 3. 3.!

column 33 to 49

! 3. 3. 3.6666667 6.3333333 7. 7. 7. 7. 7. 7. 7. 7.
5.8333333 1.1666667 0. 0. 6.661D-16!

Mean Filter #3, 2 Passes:

column 1 to 13

! 4.441D-16 6.661D-16 0.4444444 1.3333333 2.6666667 3.5555556
4. 3.5555556 2.6666667 1.3333333 1. 1.6666667 3.3333333!

column 14 to 26

! 4.4444444 4.4444444 3.3333333 1.6666667 1.1111111 1.6666667
3.3333333 4.4444444 5. 5. 4.7777778 4.3333333 3.6666667!

column 27 to 44

! 3.2222222 3. 3. 3. 3. 3. 3. 3.4444444 4.3333333 5.6666667
6.5555556 7. 7. 7. 7. 7. 7. 6.2222222!

column 45 to 49

! 4.6666667 2.3333333 0.7777778 2.220D-16 7.772D-16!

Mean Filter #4:

column 1 to 22

! 8.882D-16 - 8.882D-16 0.8 1.6 2.4 3.2 4. 3.2 2.4 1.6 1.8 2.
3. 4. 4. 3. 2. 2. 2. 3. 4. 5.!

column 23 to 46

! 5. 4.6 4.2 3.8 3.4 3. 3. 3. 3. 3. 3.8 4.6 5.4 6.2 7.
7. 7. 7. 7. 7. 5.6 4.2 2.8!

column 47 to 49

! 1.4 0. 2.220D-16!

Mean Filter #5:

column 1 to 20

! -2.220D-16 -4.441D-16 0.2 1. 3. 3.8 4. 3.8 3. 1. 0.45 1.25
3.75 4.75 4.75 3.75 1.25 0.5 1.25 3.75!

column 21 to 44

! 4.75 5. 5. 4.9 4.5 3.5 3.1 3. 3. 3. 3. 3. 3.2 4. 6.
6.8 7. 7. 7. 7. 7. 7. 6.65!

column 45 to 49

! 5.25 1.75 0.35 - 1.110D-15 3.331D-16!

Mean Filter #6:

column 1 to 12

! - 2.220D-16 0.5714286 1.1428571 1.7142857 2.2857143 2.8571429
2.8571429 2.8571429 2.2857143 2.4285714 2.5714286 2.7142857 !

column 13 to 24

! 2.8571429 2.8571429 2.8571429 2.8571429 2.8571429 2.8571429
2.8571429 2.8571429 3.5714286 4.2857143 4.7142857 4.4285714 !

column 25 to 38

! 4.1428571 3.8571429 3.5714286 3.2857143 3. 3. 3. 3.
3.5714286 4.1428571 4.7142857 5.2857143 5.8571429 6.4285714 !

column 39 to 49

! 7. 7. 7. 7. 6. 5. 4. 3. 2. 1. 2.220D-16 !

Mean Filter #7:

column 1 to 12

! 4.441D-16 0.0606061 0.4242424 1.3333333 2.5454545 3.4545455
3.7575758 3.4545455 2.5454545 1.4090909 0.9545455 1.7272727 !

column 13 to 24

! 3.1818182 4.2424242 4.2424242 3.1818182 1.7424242 1.0606061
1.7424242 3.1818182 4.3181818 4.7727273 4.8181818 4.6363636 !

column 25 to 36

! 4.1818182 3.5757576 3.1212121 2.9393939 2.9090909 2.9090909
2.9090909 2.9090909 2.969697 3.3333333 4.2424242 5.4545455 !

column 37 to 48

! 6.3636364 6.7272727 6.7878788 6.7878788 6.7878788 6.7878788
6.6818182 6.0454545 4.4545455 2.3333333 0.7424242 0.1060606 !

column 49

! 4.441D-16 !

2.5. Removing Filtered Effects (Inverse Convolution)

2.5.1 Description

At some point, it might be of interest to perform an operation on a filtered image that reverses the effects of the filter. In a sense, the operation will “undo” the filter on the image of interest. Accomplishing this task involves the same technique used to compute the inverse of a convolution, since a filter simply involves convolution. The inverse of a convolution was briefly discussed in Part 1. This section will expand upon the knowledge discussed previously, and put that knowledge to use.

Recall that the inverse of a convolution will never be exact. The matrix that is calculated to perform the inverse must have infinite size in order to be 100% accurate. This, of course, is impossible with present-day technology. Therefore, the best that can be achieved is an approximation of the inverse of a convolution. If the calculated matrix is large enough (i.e., enough terms are used to solve for the undetermined coefficients), then the approximation will become more accurate. The first terms in the resulting matrix will be the approximate terms sought after in the inverse, followed by a group of zeros, and terminated with the original values negated. For all useful purposes, the zeros and negated values are extraneous values; they are not needed for any calculation of any sort. Special attention was brought to them to make it apparent that they do exist. But for the actual application of this process, these values can be discarded.

Applying this technique to image filtering is fairly trivial. All that is needed is one term from the original convolution, and the convolved result. In terms of image filtering, the required inputs are the filtering kernel, and the filtered image. The result of this process is the original image prior to filtering.

2.5.2 Algorithm

This algorithm can be programmed in any desired programming environment. SciLab was specifically chosen to handle this algorithm because of its simple syntax and ease of manipulating matrices, as well as images. Since the methods outlined in the previous part makes use of matrices, it makes most sense to utilize the matrix functions already provided with SciLab. This avoids having to write extra code to handle matrices specifically, and then having to write the algorithm on top of that code.

The following demonstrates the SciLab implementation of this algorithm:

```
function [in] = convolinv(b,ia)

//Date:      March 15, 2005
//Author:    Tom S. Lee
//Purpose:   To demonstrate an inverse convolution function

//Inputs:
//  b: one term from the convolution a * b
//  ia: the result of the convolution a * b

//Return values:
//  in: value of second term of the convolution a * b
//  inv_d: inv_d * b = identity matrix

//Initialize "identity" matrix for inverse
size_id = 2 * max(size(ia));
size_inv = max(size(ia)) + size_id - 1;

d = zeros(size_id,1);
d(1) = 1;

in_b = zeros(size_id,size_inv);

for i = 1:size_id

    //Fill in first term of convolved result
    for j = 1:max(size(b))
        index = j+i-1;

        //ensure that the matrix does not grow too large
```

```

        if index <= size_id then,
            in_b(j+i-1,i) = b(j);
        end;
    end;
end;

//Calculate inverse matrix
inv_d = lsq(in_b,d);

in_big = convol(inv_d,ia);

size_in = max(size(ia)) - max(size(b)) + 1;
in = zeros(size_in,1);

//resize output, to remove extraneous values
for i = 1:size_in
    in(i) = in_big(i);
end;

in = in';

endfunction

```

A basic example demonstrating this algorithm is shown below:

```
-->a = [1 2 3 4 5 6]
```

```
a =
```

```
! 1.  2.  3.  4.  5.  6. !
```

```
-->b = [1 2 3 4]
```

```
b =
```

```
! 1.  2.  3.  4. !
```

```
-->ia = convol(a,b)
```

```
ia =
```

```
! 1.  4.  10.  20.  30.  40.  43.  38.  24. !
```

```

-->c = convolinv(b,ia)

c =

! 1.  2.  3.  4.  5.  6. !

-->

```

The result c is approximately the same as the original value a . This value c is *approximately* the same as the original value a , not *exactly* the same, because of the reasons outline above. A way to prove this is by observing the difference of c and a :

```

-->c - a

ans =

1.0D-11 *

! 1.114131  0.0454747  0.0369482  0.2671641  0.3637979

0.3410605 !

-->

```

Theoretically, the result of $c - a$ should be 0. However, due to the finite amount of storage space available to store results, the values must be truncated to a point. The result is slight errors in the answers. They are fairly small errors, nearly on the order of the ϵ of the machine. But it is wise to keep these errors in mind when using these results in other calculations. Over time, these small errors will grow, and will further throw off any answers.

The algorithm computes the inverse of the convolution by solving a set of undetermined coefficients, whose number is determined by the size of the original convolved matrix:

```

//Initialize "identity" matrix for inverse
size_id = 2 * max(size(ia));
size_inv = max(size(ia)) + size_id - 1;

```

This dynamic sizing ensures that a proper number of undetermined coefficients are used to solve the problem. If the problem size becomes large, using a hard-coded value may become insufficient to accurately solve the problem. Likewise, using a hard-coded value may prove to create a set too large for the application, and precious computation time is wasted. This method ensures both accuracy and speed.

The next step is to create and determine the coefficients that fit the particular problem. Recall the manual method of convolving two matrices. For example:

$$\begin{array}{cccc} & 1 & 2 & 3 & 4 \\ 2 & 1 & & & \end{array}$$

The top matrix is the convolved result, and the bottom shifting matrix is the set of undetermined coefficients that need to be solved. The result of this convolution must be the identity matrix. The undetermined coefficients will produce the inverse of the convolution operation. When convolved with the original convolved result, the answer will be the identity matrix. To achieve this, a system of equations is set up. Each equation corresponds to each intermediate operation and result of the shifting of one matrix over the other. The first three equations that result from computing the first three shifts in the above example are as follows:

$$\begin{array}{cccccc} & & 1 & 2 & 3 & 4 \\ b & a & & & & \\ & b & a & & & \\ & & b & a & & \\ \hline & 1 & 0 & 0 & 0 & 0 \\ 1*a + 0*b = 1 & & 2*a + 1*b = 0 & & 3*a + 2*b = 0 & \end{array}$$

Once all the equations have been determined for this system, they can be arranged as a matrix system. A simple matrix solution for system of equations can be used to solve for the undetermined coefficients. To solve the system, the matrices can be arranged in a simple $Ax = b$ format, where A = the original convolved result (top matrix), b = the identity matrix, and x = the undetermined coefficients. Solving for x will provide the values of the undetermined coefficients, and will create the inverse convolution for the originally specified matrices. Common techniques to solve for x include LU factorization, QR factorization, or a least squares solution.

This entire process is handled in this section of code:

```
for i = 1:size_id
    //Fill in first term of convolved result
    for j = 1:max(size(b))
        index = j+i-1;

        //ensure that the matrix does not grow too large
        if index <= size_id then,
            in_b(j+i-1,i) = b(j);
        end;
    end;
end;

//Calculate inverse matrix
inv_d = lsq(in_b,d);
in_big = convol(inv_d,ia);
```

This algorithm utilizes a least squares solver that is native to the SciLab library. The least squares solution will yield the most accurate result for this problem when compared to LU or QR factorization. Additionally, the least squares solver handles rank deficient matrices with no complaint, which can affect the result and function of a LU or QR factorization. Most specifically in SciLab, if a LU or QR factorization method is chosen,

and the calculated system of equations produces a rank deficient matrix, SciLab will determine that the QR or LU factorization methods are insufficient for the particular problem, and will automatically revert to a least squares method to solve the problem. A warning is displayed in the SciLab console of this occurrence:

```
-->convolinv(b,ia)
rank defficient. rank = 102
```

As stated earlier, computing the inverse convolution will produce a matrix with the actual values, followed by a string of extraneous values. Since the values that are desired are produced in the first portion of the result, the rest of the matrix can be discarded. The last step of the algorithm is to truncate the result and keep only the required elements:

```
size_in = max(size(ia)) - max(size(b)) + 1;
in = zeros(size_in,1);

//resize output, to remove extraneous values
for i = 1:size_in
    in(i) = in_big(i);
end;

in = in';

endfunction
```

Keep in mind that the result is approximate, not exact. Nevertheless, the result is the closest value that can be achieved with the limitations of modern computing. For the purposes of basic convolution and image filtering operations in the context of this discussion, the small errors that are produced in this operation are acceptable.

Ultimately, it is up to the operator to determine the degree of accuracy necessary, and the necessary steps can be taken to achieve that goal.

2.5.3 Implementation

Since image filtering involves the same exact technique as matrix convolution, the inverse convolution algorithm described above can be used directly to reverse a filtering process on an image. In order to remove the filtering effect on an image, the function will require the kernel of the filter used, and the filtered image.

Assume that a poor filter was implemented on an image, and it was necessary to remove the effects of that filter on the image. Out of the mean filters examined, Filter #6 was by far the worst at maintaining the information of the original image:

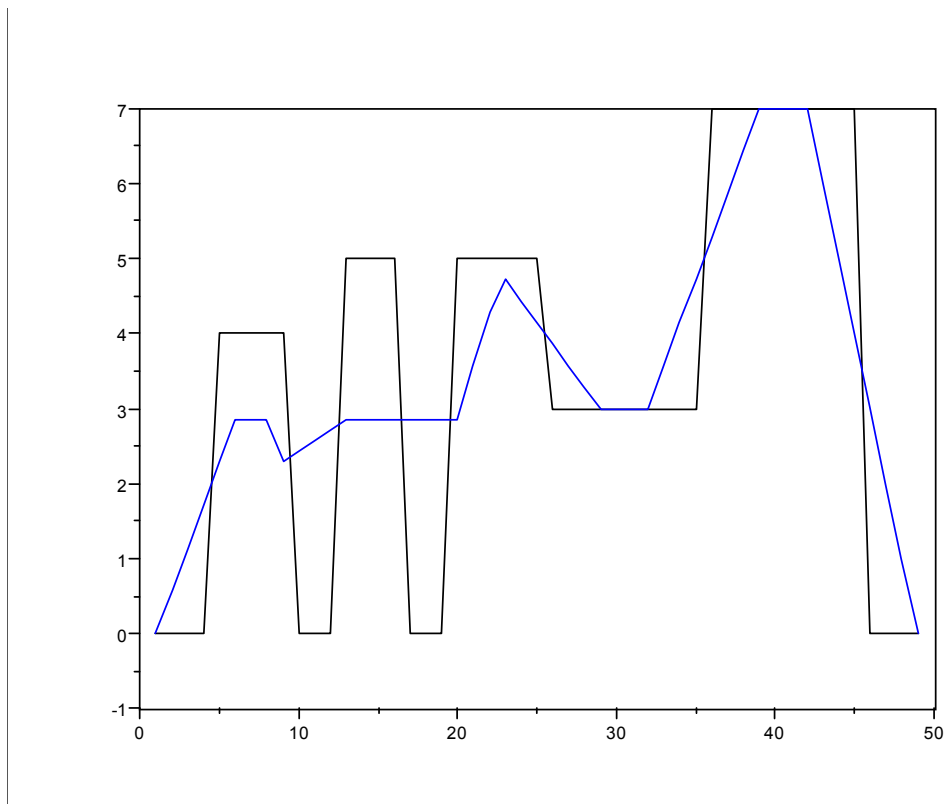


Image 20: Original Skyline Image (Black), 7-Pixel Filter (Blue)

Removing this effect on the filtered image only involves running the image through the inverse convolution algorithm to achieve the original image. The result of applying the inverse convolution function on the filtered image is as follows:

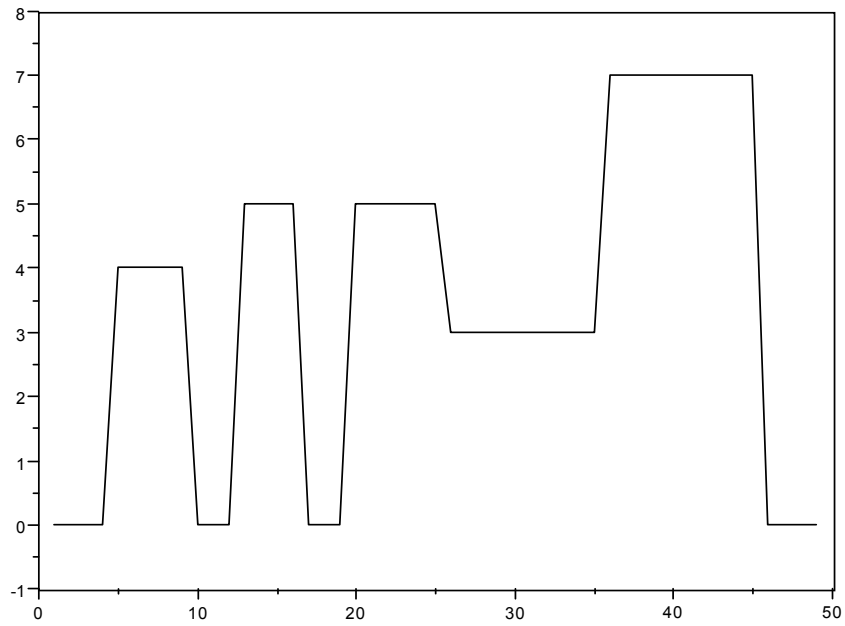


Image 21: Previously Filtered Image, After Inverse Convolution Function

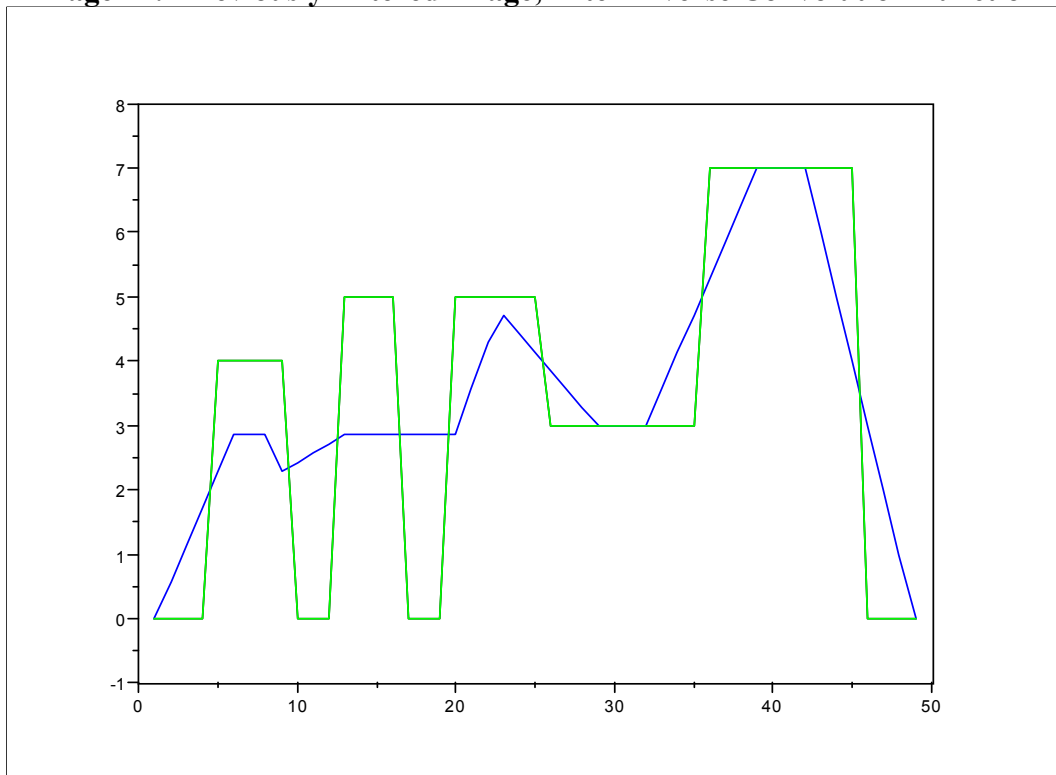


Image 22: Original Skyline Image (Black), 7-Pixel Filter (Blue), After Applying Inverse Convolution Function (Green)

Image 22 does in fact contain three different images. It just so happens that the green and black images are identical.¹ Mathematically speaking, the line segments that make up each image are colinear. In SciLab, since the green image was displayed last, that image will appear on top of the black image that was previously drawn. If the green image were drawn first, then only the black image would be visible. Either way, both images are identical.

There is one caveat to obtaining these results. Recall that the original filtering done on the original skyline image produces a result that is larger than the original image. In order to properly display the results, the filtered data needed to be truncated. Recall that when performing convolution manually, the shifting of the second matrix over the first (i.e., applying the kernel to the image) required zeros to be artificially inserted at either ends of the first matrix. Those values are not necessary for displaying the new filtered result. In order to ensure the filtered result is the same size as the original image, the values that were calculated using those zeros need to be removed. Each filtered image displayed reflected this procedure.

When performing the inverse convolution to remove the filtering effects on the filtered image, it is necessary to obtain the pre-truncated filtered result. Using only the center terms in the inverse convolution function will produce an inverted result that is smaller than the original image. In terms of graphing the results, the inverted result will be the approximation of the original image, with a few of the terms truncated off either end. The number of terms that will be truncated depends on the number of terms used in

¹ In reality, the green image is approximately the same as the black image, for reasons discussed earlier. To the human eye, and due to the scope of the graphs, the images are identical for all practical purposes. The term “identical” will be used in this context to describe these images, but do keep in mind that the images are in fact *almost* or *nearly* identical.

the kernel. If a small kernel is used, only a couple of terms will be missing on either end of the inverted image. If a larger kernel is used, several terms may be absent, increasing the degree of image data loss. The data that is preserved after this operation is still valid, but cannot be compared directly to the data in the original image. In order to perform a direct comparison of data in each image, the inverted image must be shifted over the appropriate amount in order for the data values to match up. This occurrence is similar to having two observers recording a car's movement across a stretch of road. One observer starts timing before the second observer, and stops timing after the second observer. Even though both observers recorded data for different lengths of time, the data that both observed simultaneously is still the same. The only difference between the two sets of data is that the first observer took note of the car's behavior prior to and after the second observer recorded data. In order to make direct comparisons between the two sets of data, it will be necessary to only compare the data of the first observer during the time interval the second observer was recording data. The same principle applies in this situation with the filtered images. To properly compare image data between the original and inverted images, it is necessary to properly align the two images, and compare only the data pieces that are common between the two.

Image 22 was generated using the original pre-truncated filtered image. This way, the result that the algorithm gave was the same size as the original image, and no further special steps were required to be able to perform a direct comparison. If the final filtered image were used to obtain the inverted image, the process would still be straightforward. The inverted image would just have to be padded with zero values (or some other chosen value) to line up the two images.

2.5.4 Asside

The idea that a filtered effect can be removed with general ease might sound appealing. But there is one important factor that determines whether or not this technique will work effectively. In order to *accurately* remove the filtering effect from an image, the *exact* filter must be known (i.e., the kernel values must be known). This operation is not fullproof, and requires a specific set of information to work properly in a given problem set. If this information is not available, then it is impossible to fully reverse the filtering effect from the image. An educated guess as to the filtering effect used can be substituted in this operation. Keep in mind that the result of inverting the filter will not be the same as the original image. It is possible to get a result that is acceptably close to the original image, but not the same as the original image. In this circumstance, research must be conducted to determine the characteristics of the filtering effect on the image. Through this research, the proper values can be chosen to input into the filter removal operation, in hopes of retaining the information encoded in the original image.