**Bilateral filtering for denoising digital images**

Aviv Elkayam & Noy Tourgeman

**What is Bilateral Filter?**

Bilateral Filter is a non-linear, edge-preserving, and noise-reducing smoothing filter for images *(From Wikipedia).* The filtered image looks smoother but remains clear – the edges in it are not getting blurred.   
  
 **How Does it Work?**

Bilateral Filter iterates over the pixel's image and calculates a new value for each pixel by using a weighted average of the neighbor pixels. This behavior is similar to many other filters - for instance, Gaussian filter, but what differentiates it from the others is how it calculate the weights.

Bilateral Filter has two main features, smoothing, and edge-preserving.   
To better understand those features we will introduce the formula and then explain each part of it separately.

**The Formula**

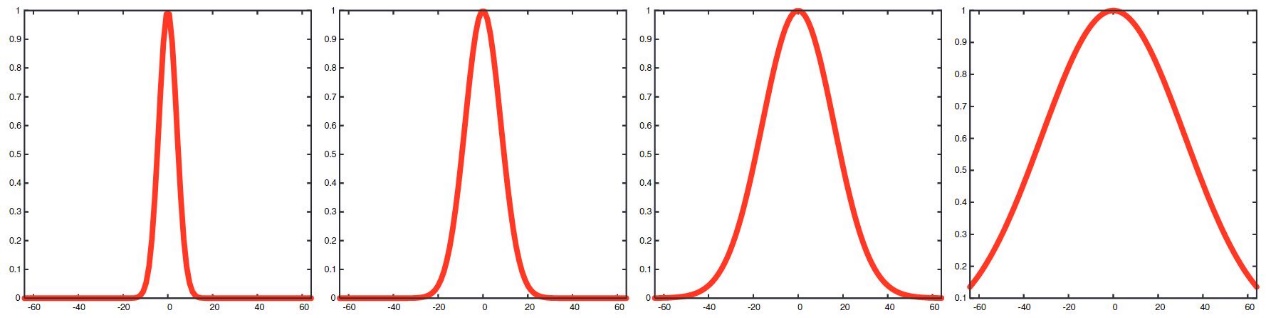
Let us divide the explanation into three parts: the weighted average, the smoothing, the edge-preserving.  
   
  
The Weighted Average   
  
This part is straightforward, when calculating the value of p, we iterate over all the pixels in the image (in practice – neighbor pixels) and calculating the weighted average.

The expression above is the weight of each neighbor pixel q.  
Notice how the weight is divided into two parts

* – responsible for the smoothing effect
* – responsible for the edge-preserving effect

Before we explain how the smoothing and the edge-preserving occurs, we need to understand what Gaussian function is.

Gaussian Function   
  
It is a function that has a symmetric "bell curve" shape. It has a parameter which controls the width of the "bell".

  
  
 Low High

This function formula is as follow

Notice that as we get closer to zero, the values of the function increases. In addition, the higher the is, the values of the function for the same input (except )  
increases too.

The Smoothing

This expression is part of the weight. We can see that greater distance between p and q will result in smaller values of and lower distance between p and q will result in higher values of . This makes closer pixels "weight" more than farther pixels.   
  
but, why does the image get smoother? Smoothing means that for some pixel p, closer pixels will have closer values. In this formula the closer the neighbor pixel is, the higher its weight will be, so the new value of the pixel will be more similar to its neighbors.

The is responsible for determining the distance of the pixels that are still considered "related" to each other. Higher values of will result in a smoother image.

**The Problem with Smoothing**

When we smooth an image, we make the values of the neighbor pixels more similar. But what if we calculate a value for a pixel that is on some edge in the image? It will make the edge blur and therefore the image less clear. To fix this problem let us talk about the edge-preserving part.

Edge Preserving

Here, the closer their values are, the higher the weight will be. In other words, if the difference in their values is high (they are less similar), they are probably on the edge of the image and therefore they will have smaller weights. if the difference in their values is small (they are similar), they are probably not on any edge and they will get a higher weight.

The Is responsible for edge-preserving. The higher the is, the lesser the filter will preserve the edges in the image.

**Conclusion**

The product describe the weight of each neighbor pixel, The first part makes the image smoother while the second part cancels it out if there is an edge between those pixels.

Thus, the image will get smoother on parts that are not near edges (The Smoothing) and will not get smoother on parts that are near the edge (Edge Preserving).

**Pseudo Code**

for each pixel p

for each pixel q in the window

calculate

The window size is .

**Optimal Window Size**

Notice that the parameter of the Gaussian function controls the width of the "bell".

For large enough inputs (with absolute value), the Gaussian function will have values that are close to zero. In our algorithm the input for the Gaussian function is the distance between p and q, so for large enough distance the Gaussian function will have values that are close to zero, and these calculations will not affect the new value of p. That is why we do not have to iterate over the whole image for each pixel and can limit the window to a smaller size.

For small enough inputs (with absolute value), the Gaussian function will have large values that are not negligible to the calculation of the new value of p. If we choose small enough windows, there will be some pixels outside of the window that the distance between them and p will be small and therefore the Gaussian function will have large enough value that can affect the new value of p. By taking a small window, we can miss some of the effects of the Bilateral Filter on the filtered image.

For a window of size , the algorithm will not miss any effect and will not calculate unnecessary calculations.

**Efficient Algorithm**

This algorithm above is naive and its time complexity is where S is the number of pixels in the image. Some algorithms are more efficient, we will examine one of them here.  
  
Separable Kernel [Pham and Van Villet 05]

The algorithm takes a new approach, instead of iterating over all the neighbor pixels it iterates only on the neighbors that are on the same row and then iterating on the neighbors that are on the same column. The time complexity is reduced to . That is because the algorithm iterates twice for each pixel on only one dimension of the window size, while the length of the window is .

This algorithm has a flaw, it creates axis-aligned "streaks" on the image. That happens because the algorithm only takes the average of the pixels that are on the axes and not for the whole window. There may be a situation where the pixels outside of the axes will have a significant effect on the average, but the algorithm will miss it.

**Grayscale Image Examples**

In this section, we will clarify more about how and affect the filter with a set of examples.

**All the examples uses a window size of 11X11 pixels.**

Original Image (275 x 183 pixels)



Results



As we said earlier, is responsible for how far pixels are considered related to each other, in other words how smoothness we want to add to the image. We can see that low adds almost no smoothness and higher adds a lot of smoothness to the picture.

is responsible for the amount of edge-preserving. We can see that with low the edges become clearer, and when the is high the edges become less clear.

Notice that adding smoothness to the image without preserving the edges in it (high ), is adding blurriness. In fact, this is exactly how **Gaussian Filter Works!**

Original Image (550 x 550 pixels)



Results



**Colored Image Examples**

The colored version of this algorithm takes the pixel's RGB values and calculate the Bilateral Filter new value on each one of them separately.

**All the examples uses a window size of 11X11 pixels.**

Original Image (800 x 480 pixels)



Results



Original Image (225 x 225 pixels)



Results



**References**

* Bilateral Filter for Gray and Color Images (Tomasi and Manduchi)   
  <https://users.soe.ucsc.edu/~manduchi/Papers/ICCV98.pdf>
* A gentle introduction to bilateral filtering and its applications   
  <https://www.youtube.com/watch?v=S9Cd_VgegZE&t=6506s>
* Bilateral Filtering Theory and Applications (Sylvain Paris, Pierre Kornprobst, Jack Tumblin, and Fredo Durand)  
  <https://people.csail.mit.edu/sparis/publi/2009/fntcgv/Paris_09_Bilateral_filtering.pdf>