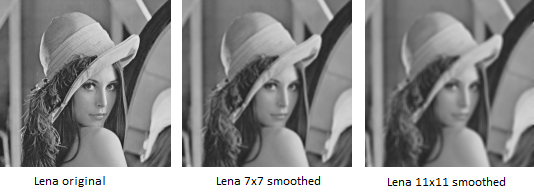
# Smoothing

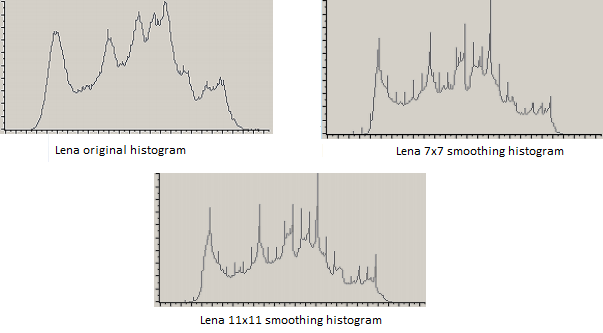
Smoothing in general is very easy to explain. Its purpose is to smoothen the image create a better base for other forms of image processing. Here is an example of a picture that is smoothed:



On the left side you can see the original image. All the pixels are the color they should be and you can see every edge very clear. On the right side you can see the same image, but with smoothened edges. This technique is used to remove certain noise from a picture. When this noise is removed, you can work with the new data.

Smoothing can be done with certain smoothing masks. These masks can be 3x3, 5x5 and bigger. These masks use the values of the pixels around the one you’re editing and that way he can calculate the needed value for the pixel you want to edit. If you do this for all the possible pixels (you cannot do this for the pixels on the edge of the image since you’ve got masks of minimal 3x3) you will get a new image looking like the second in the picture above. As mentioned before, your smoothing mask can vary. The bigger your mask is, the smoother the result will be.



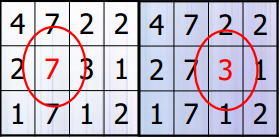


Looking at these histograms you can definitely see difference between the pictures, smoothened or not. The amount of pixels with a certain top value (the tops in the histogram of the original picture) decreases and the values will all be closer to their neighboring values.

But what calculations are done to get these results? There are several algorithms to calculate the needed values. Two of which will be explained here, the Laplacian method and the Gaussian blur method.

## Laplacian

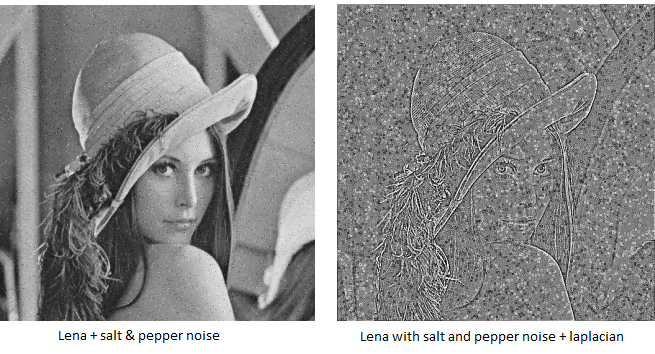
The Laplacian method is an algorithm to detect edges and make a move when one is detected. It selects the 2 neighboring pixels and detects their value. Then the algorithm will solve a simple sum to see if the value of the Laplacian will change from above 0 to below 0, or the other way around. An example and possibly a better explanation will follow.



Looking at this picture we see 2 different grids of 4x4. Imagine you’re currently standing on the spot of the red 7 in the first grid. You can calculate the Laplacian of this picture by finding out what the values are of the neighboring pixels. In this case they are 2 and 3. The calculation that needs to be done is as follows: 3 + 2 - 2∙7 = -9. The first piece of the puzzle, (3 + 2) is the sum of the values of the neighboring pixels. The second piece is the calculation of the value of the current pixel, 2 \* 7. The answer of this sum shows that the laplacian of this pixel is -9. Now we move on to the next pixel. The neighboring pixels have a value of 7 and 1, and its own value is 3. Making the calculations (7+1 – 2\*3 = 2) will show that the answer is 2, so the laplacian of this pixel is 2. We can see that the laplacians of this pixel and the previous one are -9 and 2. The values differ from below 0 to above 0. That means that an edge is detected at the pixel with value [3].

The good part about this formula is that it can tell you exactly where in the pictures you can find edges. This formula, however, also has a bad part. If a picture has noise in it, for instance salt and pepper noise, the formula will detect edges all over the place. In other words, this method can be used on a noisy picture but don’t expect accurate results. As an example of what will be the result of this, here are some pictures.

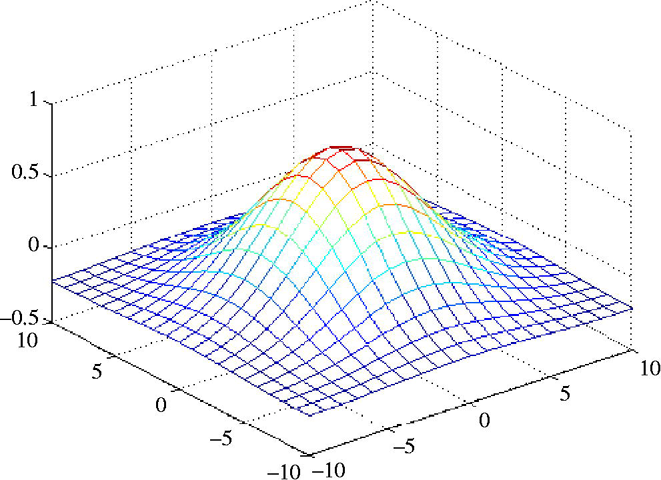




# Gaussian blur

The Gaussian blur method is a simple way of making a picture look smoother. The Gaussian blur method is also a lot more efficient than other forms of smoothening, including the one discussed above. This method is used to remove noise so the user can work with the image. This method, just like the one discussed before, sets the value of a certain pixel depending on the values of its neighboring pixels.

The formula is explained very simple: the value of the pixel you’re currently standing on will be adjusted depending on the weighted average of the pixels around the current one. The weighted average is the amount of times the pixelvalue will be weighing by calculating the average. This may sound difficult, but it actually is very simple. Looking at the following picture you can see something like a mountain.



Looking at this mountain you can say that the top of the mountain is the pixel located at 0,0 (current pixel). Let’s say this middle pixel weighs 100 (like 100%). The more you move down the hill of the mountain, the less the pixels will weigh. The pixels at 50% of the mountain will weigh 50 and so on.

http://upload.wikimedia.org/math/9/5/e/95ecdbb16befd4fdb760fa26c83a4b5e.png

Using the formula shown above we can calculate the new pixel value. The formula works pretty clear: The sigma (σ) represents the blurring factor. The bigger the number, the more blur you create. The e in this formula is a constant value, just like Pi, but with a different value (about 2.7182). The x represents the horizontal pixel value which, together with y (being the vertical pixel value), can be used to target a specified pixel. This means that the x and y represent the pixel you wish to change. If you give them both the value 0, you are standing on the center pixel (the pixel you’re editing). The x^2 + y^2 represent the weight of the pixels in the neighborhood. The farther you move away from the center pixel, the less weight the pixel has.