UNIVERSITÄT TÜBINGEN
LEHRSTUHL FÜR COMPUTERGRAFIK
PROF. DR.-ING. HENDRIK P.A. LENSCH
JIEEN CHEN
14.06.2018

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

COMPUTATIONAL PHOTOGRAPHY
ASSIGNMENT 7

Submission deadline for the exercises: Thursday, 28. Jun 2018 before 11:59.

Instructions: Upload the source code to your solution **(no images please, just the plain code)** in the ILIAS system at:
ILIAS Computational Photography SoSe2018
This assignment sheet has a total of 160 points.

7.1 Simple Image Denoising (30 Points: 5, 5, 2, 18)

In the lecture you have learned that noise is inevitable in digital imaging. In the exercise 5, you were asked to reduce noise in an image by averaging multiple shots of the same scene. However averaging multiple images of moving objects/dynamic scenes under dim lighting conditions (think of a live concert for example) is a delicate task. Also increasing the exposure time to gather more light is not an option due to motion blur. As a result there is only a very faint light signal forming the image on the sensor. Using typical CCD or CMOS sensors, this signal can not be amplified without also amplifying noise. Hence, denoising is an important image processing task.
There is a big variety of algorithms that reduce noise from images in a post processing step. In this exercise you will implement a measure to determine the signal-to-noise ratio (SNR) in an image, given the noise-free ground truth image. You will also implement a very easy, yet effective filter to remove noise from an image, namely the so-called *Bilateral Filter*.

(a) Ground Truth

(b) Noisy Image

a) Implement a function `MSE(img, reference)`, that calculates the Mean Squared Error (MSE). Compare and validate your implementation with the MATLAB implementation `immse(img, reference)` on the provided noisy and ground truth image.

b) Implement a function `PSNR(img, reference)` that calculates the Peak Signal-to-Noise Ratio (PSNR). Compare and validate your implementation with the MATLAB implementation `psnr(img, reference)` on the provided noisy and ground truth image.

c) Use a simple Gaussian filter to remove noise from the `noisy` image and compare the PSNR value of the result with the PSNR value of the input image. Last time you implemented your own Gaussian filter - this time you may use MATLAB's built-in version.

d) Implement a function `BilateralFilter(input, sigma_s, sigma_r)` where `input` is the noisy input image, `sigma_s` the standard deviation for the spatial domain filter and `sigma_r` is the standard deviation in the color domain. Compare the result and PSNR with the result of the Gaussian filter. (HINT: you might want to use a `waitbar` to show the progress of the filter. Don't forget to close the waitbar at the end of the function.)

7.2 Non-Local Means Filter (70 Points: 20, 5, 5, 20, 10, 10)

With the Bilateral Filter you have implemented your first non-trivial filter for image denoising. In this exercise you will create a simple implementation of a more advanced denoising filter: the *Non-Local Means Filter* (NLM filter). We use the same functions as in Exercise 7.1 to compare the results of the NLM filter. The NLM filter can be vectorized and thus be implemented efficiently in MATLAB - however the vectorization is rather complicated and obscures the individual steps of the algorithm. In contrast, our step-by-step implementation will operate pixel-wise and therefore be slow. You might want to use the small `ground_truth` and corresponding `noisy` images in the script `denoise.m` for debugging purposes.

Mathematically, the NLM filter can be formulated as:

$$u(p) = \frac{1}{C(p)} \int_{\Omega(p)} v(p)\, w(p,q)\, dq \qquad \text{with} \qquad C(p) = \int_{\Omega(p)} w(p,q)\, dq$$

Where p is a pixel position, $u(p)$ the filter output color value at that position, $v(p)$ the unfiltered pixel value at position p, $w(p,q)$ a weighting function and $\Omega(p)$ an area around pixel p, which is a subset of the entire image. So the filter output is a weighted average (mean) over the image patch $\Omega(p)$. The weighting function $w(p,q)$ computes the similarity between the pixel p and q. The similarity is computed by comparing a small patch around p with a small patch around q.

a) A crucial component of the NLM filter is the comparison of image patches. The similarity $w(p,q)$ between two image patches P_1 and P_2 centered respectively at p and q is defined as

$$w(p,q) = e^{-\frac{\|P_1 - P_2\|^2}{2\sigma^2}}.$$

Implement a function `SimilarityWeight(patch1, patch2, sigma)` which returns the similarity between two given patches in this manner.

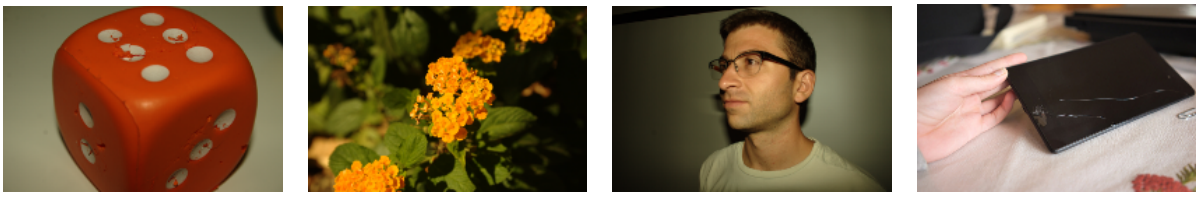
b) The NLM filter considers small square-shaped patches around pixels to compute similarities between pixels. Pad the input image in such a way that even pixels on the border can be compared. (HINT: The radius of the patch is given by `similarity_r`.)

c) For a pixel at position $p = (y, x)$ extract the patch for the similarity comparison with all other pixels in the search radius.

d) Iterate over the pixels in the search radius around $p = (y, x)$ and compute the similarity of those pixels using the function from a). The pixel at position $p = (y, x)$ is also contained in this search window - however, it gets a special treatment. Instead of reporting the similarity 1 on that pixel we use the maximum similarity amongst all other pixels in the window to weight it. Use the similarity as a weight to compute the average pixel value and sum up all used weights.

e) Compute the filtered pixel value and write it to the result image. It can happen that the similarity of all pixels in the search window was zero (horrible image or wrong parameters). In that case just copy the input pixel to the result. Otherwise normalize the computed pixel average using the summed weights according to the formula above.

f) Compare the results to the methods in 7.1. Try to find good parameters for all three methods (Gauss, Bilateral, NLM) on one of the given images in Figure 1. Create a PDF with images that show results for 3 different parameter settings and explain the impact of the parameters. Also compare the PSNR values. Do they represent your visual impression of the reconstruction quality? (HINT: Don't rise the search and similarity window radii too much. Try figuring out the best σ values.)



(c) Dice Image (d) Flowers Image (e) Man Image (f) Broken Tablet Image

Figure 1: Test Images