project 2:

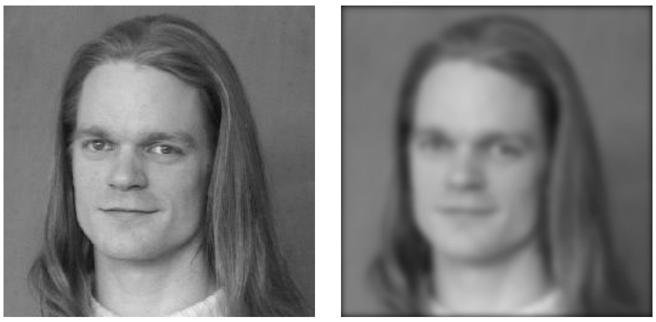
Gaussian- and bilateral filters

solution(s) due:

December 17, 2018 at **12:00** via email to **bauckhag@bit.uni-bonn.de**

problem specification:

task 2.1 Implement functions that smooth a gray value image using a Gaussian filter mask. The following example was created using a mask size of 15×15



1. Implement a naïve convolution using a 2D filter mask; given the size of the filter mask as a parameter, use what has been discussed in the lecture to determine the values of individual bins of the convolution kernel; make sure your kernel is properly normalized; note that you will have to be careful as to how your code handles pixels close to the image border.

2. Implement a solution based on convolution where you exploit the fact that the Gaussian is a separable function; given the size of the filter mask as a parameter, use what has been discussed in the lecture to determine the values of individual bins of the convolution kernel; note that you will have to be careful about how your code handles pixels close to the image border.

3. Implement a solution where you exploit the fact that convolution in the space domain corresponds to multiplication in the frequency domain, i.e. a solution where you incorporate Fourier transforms of the image and the filter mask; given the size of the filter mask as a parameter,

use what has been discussed in the lecture to determine the variance σ^2 of the convolution kernel; consider the fact that the image is of size $M \times N$ and the kernel of size $m \times n$; what does this imply, if you want to multiply their Fourier transforms? Note that using Python's `fftshift` function may simplify things.

4. Use either Python's functions `time.time()` (on Linux machines) or `time.clock()` (on Windows machines) to determine the runtime of the different approaches; figure out the details of time keeping in Python by yourself; working on an image of your choice, consider filter masks of size 3×3, 5×5, $\ldots$, 21×21 and run your code 10 times for each parameter setting; create a plot that compares the average run-times of the methods for the different filter sizes, i.e. plot run-times versus filer size.

task 2.2 When computing gradient images, it is beneficial to apply the gradient operator to a smoothed version of the input image in order to ensure robustness against noise. That is, in practice one usually considers operations like this

$$h(x, y) = \nabla\big(f(x, y) * g(x, y)\big)$$

where $g(x, y)$ is a Gaussian filter kernel.
Implement a function that computes gradient images using this idea. In your implementation, make use of the fact that

$$\frac{\partial}{\partial x}\big(f(x, y) * g(x, y)\big) = f(x, y) * \frac{\partial}{\partial x} g(x, y)$$

and likewise for $\frac{\partial}{\partial y}$; again, pay attention to proper normalization; run your code for different filter sizes; create gradient magnitude images of the images `bauckhage.jpg` and `clock.jpg` to compare the effects of different parametrizations.

task 2.3 Implement a recursive filter for Gaussian image smoothing. Since convolving an image with a 2D Gaussian kernel is a separable operation, it is sufficient to consider the implementation of a 1D recursive filter which is applied consecutively.
Proceed as follows: For a 1D signal $x[n]$, compute the 1D filter response $y[n]$ as the sum of a causal and an anti-causal filter, i.e.

$$y[n] = \frac{1}{\sigma\sqrt{2\pi}}\left(y^+[n] + y^-[n]\right) \tag{1}$$

where the causal filter is given by

$$y^+[n] = \sum_{m=0}^{3} a_m^+ x[n-m] - \sum_{m=1}^{4} b_m^+ y^+[n-m] \tag{2}$$

and the anti-causal filter is given by

$$y^-[n] = \sum_{m=1}^{4} a_m^- x[n+m] - \sum_{m=1}^{4} b_m^- y^-[n+m] \tag{3}$$

Given the standard deviation σ of the isotropic Gaussian kernel as an input parameter for your implementation, compute the coefficients of the causal filter as

$$\begin{aligned}
a_0^+ &= \alpha_1 + \alpha_2 \\
a_1^+ &= e^{-\frac{\gamma_2}{\sigma}}\big(\beta_2 \sin\frac{\omega_2}{\sigma} - (\alpha_2 + 2\alpha_1)\cos\frac{\omega_2}{\sigma}\big) + e^{-\frac{\gamma_1}{\sigma}}\big(\beta_1 \sin\frac{\omega_1}{\sigma} - (2\alpha_2 + \alpha_1)\cos\frac{\omega_1}{\sigma}\big) \\
a_2^+ &= 2e^{-\frac{\gamma_1+\gamma_2}{\sigma}}\big((\alpha_1 + \alpha_2)\cos\frac{\omega_2}{\sigma}\cos\frac{\omega_1}{\sigma} - \cos\frac{\omega_2}{\sigma}\beta_1 \sin\frac{\omega_1}{\sigma} - \cos\frac{\omega_1}{\sigma}\beta_2 \sin\frac{\omega_2}{\sigma}\big) \\
&\quad + \alpha_2 e^{-2\frac{\gamma_1}{\sigma}} + \alpha_1 e^{-2\frac{\gamma_2}{\sigma}} \\
a_3^+ &= e^{-\frac{\gamma_2+2\gamma_1}{\sigma}}(\beta_2 \sin\frac{\omega_2}{\sigma} - \alpha_2 \cos\frac{\omega_2}{\sigma}) + e^{-\frac{\gamma_1+2\gamma_2}{\sigma}}(\beta_1 \sin\frac{\omega_1}{\sigma} - \alpha_1 \cos\frac{\omega_1}{\sigma}) \\
b_1^+ &= -2e^{-\frac{\gamma_2}{\sigma}}\cos\frac{\omega_2}{\sigma} - 2e^{-\frac{\gamma_1}{\sigma}}\cos\frac{\omega_1}{\sigma} \\
b_2^+ &= 4\cos\frac{\omega_2}{\sigma}\cos\frac{\omega_1}{\sigma}e^{-\frac{\gamma_1+\gamma_2}{\sigma}} + e^{-2\frac{\gamma_2}{\sigma}} + e^{-2\frac{\gamma_1}{\sigma}} \\
b_3^+ &= -2\cos\frac{\omega_1}{\sigma}e^{-\frac{\gamma_1+2\gamma_2}{\sigma}} - 2\cos\frac{\omega_2}{\sigma}e^{-\frac{\gamma_2+2\gamma_1}{\sigma}} \\
b_4^+ &= e^{-\frac{2\gamma_1+2\gamma_2}{\sigma}}
\end{aligned}$$

where you set the coefficients $\alpha_i, \beta_i, \gamma_i, \omega_i$ according to the following table

	α_i	β_i	γ_i	ω_i
$i = 1$	1.6800	3.7350	1.7830	0.6318
$i = 2$	-0.6803	-0.2598	1.7230	1.9970

Finally, once the causal filter coefficients a_m^+ and b_m^+ have been computed, compute the coefficient of the anti-causal filter according to

$$
\begin{aligned}
b_m^- &= b_m^+ \quad \forall m \\
a_1^- &= a_1^+ - b_1^+ a_0^+ \\
a_2^- &= a_2^+ - b_2^+ a_0^+ \\
a_3^- &= a_3^+ - b_3^+ a_0^+ \\
a_4^- &= -b_4^+ a_0^+
\end{aligned}
$$

Make sure your code handles image boundary conditions! Apply the filter to smooth the images `bauckhage.jpg` and `clock.jpg` using increasing values for the parameter σ. Carry out time keeping experiments as in the last project.
Note: If you are programming in Python, the recursive filter may actually be slower than the built-in convolution functions which internally call highly optimized C code. However, those of you who program in C/C++ will see the light! The recursive filter is typically faster than filtering in the frequency domain!

task 2.4 Implement a bilateral filter

$$h[x, y] = \gamma \cdot \sum_{i=-\frac{m}{2}}^{\frac{m}{2}} \sum_{j=-\frac{m}{2}}^{\frac{m}{2}} G_\rho\Big[f[x-i, y-j] - f[x, y]\Big]\, G_\sigma[i, j]\, f[x-i, y-j]$$

of size $m \times m$ where γ is a normalizing factor and G_ρ is a discrete univariate Gaussian and G_σ is discrete bi-variate Gaussians with respective standard deviations ρ and σ.
Apply your filter to the images `bauckhage.jpg` and `clock.jpg` and experiment with different choices for ρ and σ.

task 2.5 Prepare a presentation about your solutions and results. When you are going to present your work (i.e. give a talk in front of your fellow students and professor), it should contain up to 12 slides and not take more than 10 minutes. Make sure your presentation is clearly structured and answers questions such as “what is the task/problem we considered?”, “what kind of difficulties (if any) did we encounter?”, “how did we solve them?”, “what are our results?”, “what did we learn?”, ...

general hints and remarks

- Send all your solutions (code, resulting images, slides) in a ZIP archive to bauckhag@bit.uni-bonn.de
- Remember that you have to successfully complete all three practical projects (and the tasks therein) to be eligible to the written exam at the end of the semester. Your grades (and credits) for this course will be decided based on the exam only, but –once again– you have to succeed in the projects to get there.
- Not handing in a solution implies failing the course.
- Your project work needs to be *satisfactory* to count as a success. Your code and results will be checked and your presentation needs to be convincing.
- If your solutions meets the above requirements and you can demonstrate that they work in practice, it is a *satisfactory* solution.
- A *good* to *very good* solution requires additional efforts especially w.r.t. to elegance and readability of your code. If your code is neither commented nor well structured, your solution is not good! A very good solution requires additional efforts towards the quality of your project presentation in the colloquium. Your presentation should be well timed, consistent, and convincing. You are also very much encouraged to extend your work to other images and to explore the behavior of the Fourier transform to a greater extend than asked for in this project. Be proactive and strive for very good solutions!.