# Assignment 1: Filters and edge detection

**Before you start:**
Download the project's files here.

The project's grade is based on 1/3 written assignment and 2/3 code assignment.

**Written Assignment:**
Once you have finished the coding assignment (described next), you'll need to do the written assignment. You can find the written assignment in the file "written-assignment-1.pdf".

**Coding Assignment:**
For this assignment, you'll be implementing the functionality of the ImgFilter program. The program allows you to load an image and apply various images filters.

In the project, you should only have to update one file "Project1.cpp." The buttons in the UI will call the corresponding functions in "Project1.cpp. " Several of the buttons have already been implemented, including "B/W", "Add noise", "Mean" and "Half size. " The implementations of these buttons are in "Project1.cpp", and should provide helper sample code. Please see these functions to understand how to access the pixels in the image.

**What to turn in:**
To receive credit for the project, you need to turn in the completed file "Project1.cpp", the requested images for each task, and the written assignment.

**Tasks:**

1. Implement the function GaussianBlurImage(QImage *image, double sigma) to Gaussian blur an image. "sigma" is the standard deviation of the Gaussian. Use the function MeanBlurImage as a template, and implement the Gaussian blur using a 2D filter.

**Required:** Gaussian blur the image "Seattle.jpg" with a sigma of 4.0, and save as "1.png".

2. Implement the function SeparableGaussianBlurImage (QImage *image, double sigma) to Gaussian blur an image using separable filters. "sigma" is the standard deviation of the Gaussian. The separable filter should first Gaussian blur the image horizontally, followed by blurring the image vertically. The final image should look the same as when blurring the image with GaussianBlurImage.

**Required:** Gaussian blur the image "Seattle.jpg" with a sigma of 4.0, and save as "2.png".

3. Implement the functions FirstDerivImage(QImage *image, double sigma) and SecondDerivImage(QImage *image, double sigma) to filter an image with the first and second derivatives of the Gaussian. "sigma" is the standard deviation of the Gaussian. The first derivative should be computed along the x-axis with a regular Gaussian, while the second derivative should be computed in both directions, i.e. , the Mexican hat filter. Hint: To compute the first derivative, first compute the x-derivative of the image, followed by Gaussian blurring the image (see slide in Filters). You can use a similar trick for the second derivative (see slide in Filters. )

Remember to add 128 to the final pixel values, so you can see the negative values.

**Required:** Compute the first and second derivatives with a sigma of 1.0 for the image "LadyBug.jpg" and save as "3a.png" and "3b.png".

4. Implement the function SharpenImage(QImage *image, double sigma, double alpha) to sharpen an image. "sigma" is the Gaussian standard deviation and alpha is the scale factor (see slide in Filters. ) Hint: Use SecondDerivImage.

**Required:** Sharpen "Yosemite.png" with a sigma of 1.0 and alpha of 5.0 and save as "4.png".

5. Implement SobelImage (QImage *image) to compute edge magnitude and orientation information. SobelImage should display the magnitude and orientation of the edges in an image (see commented code in Project1.cpp for displaying orientations and magnitudes. )

**Required:** Compute Sobel edge filter on "LadyBug.jpg" and save as "6.png".

6. Implement BilinearInterpolation(QImage *image, double x, double y, double rgb[3]) to compute the linearly interpolated pixel value at (x ,y). Both x and y are continuous values (see here. )

**Required:** The function RotateImage is already implemented and uses BilinearInterpolation to rotate an image. Rotate the image "Yosemite.png" 20 degrees and save as "7.png".

7. Implement FindPeaksImage(QImage *image, double thres) to find the peak edge responses perpendicular to the edges. The edge magnitude and orientations can be computed using the Sobel filter you just implemented. A peak response is found by comparing a pixel's edge magnitude to two samples perpendicular to an edge at a distance of one pixel (slide "Non-maximum suppression" in EdgeDetection), call these two samples e0 and e1. Compute e0 and e1 using BilinearInterpolation. A pixel is a peak response if it is larger than the threshold ("thres"), e0, and e1. Assign the peak responses a value of 255 and everything else 0.

**Required:** Find the peak responses in "Circle.png" with thres = 40.0 and save as "8.png".

### Bell and Whistles (extra credit)

(Whistle = 0.01 point, Bell = 0.02 points)


Implement an improved image padded method, i.e. "fixed" or "reflected. "


Try to filter "Yosemite.png" to resemble an Ansel Adams photograph.


Implement BilateralImage(QImage *image, double sigmaS, double sigmaI) to bilaterally blur an image. "sigmaS" is the spatial standard deviation and "sigmaI" is the standard deviation in intensity. Hint: The code should be very similar to GaussianBlurImage. Here's the info about bilateral filtering. (Written assignment 3 and 6 is required to get these points!)


Implement the Hough transform using the peaks found from FindPeaksImage.


Implement your own "crazy" filter. The more original the better.


Create a movie from progressively applying filters. For example, try iteratively applying GaussianBlurImage then FindPeaksImage, fun things happen after several iterations. The more creative the better. You may just output a bunch of images, i.e. , 1.jpg, 2.jpg, 3.jpg, etc. or make a gif animation.