# Edge Thinning and Double Thresholding

Today's exercise is focused on implementation of edge thinning and subsequent double thresholding to obtain clean edges.

## Non-maxima Supression

So far, we have used the Sobel operator or other convolution operators to obtain edges in images. For this exercise, we need edges found using central difference, see Eqs. (1) and (2).

$$f_x(x,y) = \frac{f(x-1,y) - f(x+1,y)}{2} \tag{1}$$

$$f_y(x,y) = \frac{f(x,y-1) - f(x,y+1)}{2} \tag{2}$$

These edges are rather thick. Our goal is to get edges of image represend using a single pixel edges. This is called edge thinning and we will use non-maxima supression to achieve this goal. What non-maxima supression does is that it supresses (sets to 0) all values that are not maximum in ther context. By context, we mean the close neighbour of a pixel. In a 1D situation (Fig. 1), we leave unchanged only pixels that satisfy the following equation

$$E(x-1) < E(x) > E(x+1) \tag{3}$$

As can be seen, the retained value have to be greater than the values on the left and right. A simple illustration is depicted in Fig. 1.

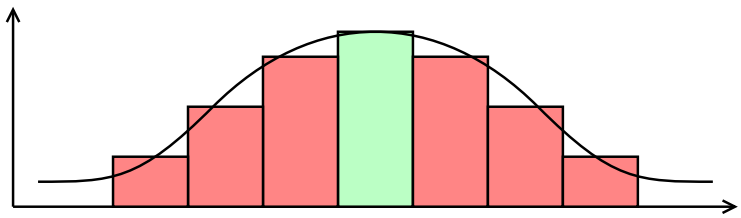

Figure 1: An example of 1D non-maxima supression. Green and red bars represent function values at specific positions. The green value is retained, red values are non-maxima, so will be set to 0.

The 2D case is a bit more complicated and requires computation of left and right coordinate for an oriented edge. We can see that values $|E_{-\Theta}|$ and $|E_{+\Theta}|$ have to be computed by linear interpolation of pixel values as is depicted in Fig. 2. The interpolation equations are presented in Eqs. (4) and (5).

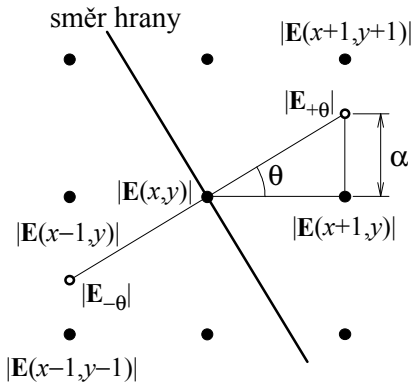


Figure 2: An edge with corresponging gradient values (note that we use Cartesian coordinate system with the $(0, 0)$ coordinate in the bottom left, so adapt your code to respect OpenCV's $(0, 0)$ image coordinate in the top left corner).

$$|E_{+\Theta}| = \alpha|E(x+1, y+1)| + (1-\alpha)|E(x+1, y)| \tag{4}$$

$$|E_{-\Theta}| = \alpha|E(x-1, y-1)| + (1-\alpha)|E(x-1, y)| \tag{5}$$

## Double Thresholding

At this point, we have new image with edge magnitudes only at the centers of edges. We have to separate real edges from random image pertubations. To achieve this, we create two thresholds $t_1$ and $t_2$ that will be set experimentally. The only rule is to keep $t_2 > t_1$.

Double thresholding checks each edge magnitude $E(x, y)$ and if it is greater than $t_2$ we set the pixel at coordinate $(x, y)$ in the output image to white (255). If the edge magnitude $E(x, y)$ is less than $t_2$ and greater than $t_1$ and is located next to the coordinate that has been already set as an edge, we set this coordinate as edge pixel too.

This algorithm may be easily implemented using recursive function. This function checks each edge magnitude and labels the coordinate as an edge, it recursively calls itself at coordinates of top, bottom, left, and right neighbouring pixels.