**Description of the project:**

For this project, we must develop a Canny Edge Detector (using C++). We will quickly see what are the different steps of the canny edge detector.

1. Reading the image:

At first, it is necessary to read the image chosen by the user (among the 5 available images). We will then extract the different information present in the ".raw" file. The information extracted is the number of rows (2 bytes), the number of columns (2 bytes), the number of bits per pixel (1 byte) and finally the values of all the pixels of the read image (all bytes remaining).

1. Smoothing the image:

The next step is to apply a Gaussian filter to the original image to smooth the image. Smoothing eliminates the noise and many "false edges" beforehand. For this, we will create a Gaussian mask (depending on the value σ entered by the user) and then convolve the original image with this mask.

1. Calculation of derivatives:

The third step is the calculation of the derivatives (in x and y) of the smoothed image. From these, we will be able to determine the magnitude as well as the direction of the gradient. For this, we will apply a Sobel filter. After filtering we get an image containing the edges of the original image (magnitude) as well as the direction of the gradient for each of the pixels. As a reminder, this direction is perpendicular to the edge.

1. Non- maximum suppression:

This step is used to set all isolated pixels that are not part of an edge to 0. For each pixel of the image, if the value of the pixel is less than one of the values of the 2 adjacent pixels (in the direction of the gradient) then the value of the pixel in question is set to 0 because it means that this pixel is not part of the edge.

1. Hysteresis:

This step is the last of canny filtering. It allows to keep only pixels that belong to an edge. Using 2 thresholds (100 and 200), we will classify the pixels into 3 categories ("it's not an edge" (pixel = 0), "it may be an edge" (pixel = 170), "it's an edge” (pixel = 255). After the thresholding, we are going to classify each pixel whose value is equal to 170 ("it may be an edge") in one of the other two categories to obtain the final image with all the edges (only 2 values: 0 and 255). For each of the pixels whose value is 170, if at least one of the 8 surrounding pixels is 255, then the value of the pixel is set to 255. Otherwise, the pixel value is set to 0.

Conclusion: after these steps, we obtain the desired final image. That is, a binary image that represents the edges of the original image.

**Observations and comments:**

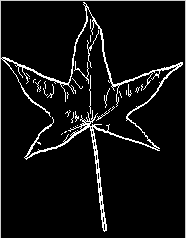
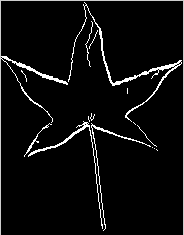
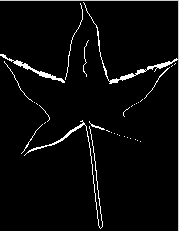
The choice of σ to create the Gaussian mask:

The choice of the Gaussian mask is very important and will greatly influence the result obtained. First, to enclose 98.76% of the area (probability) under the Gaussian distribution, a mask must be truncated with a minimum width, w, of at least w = 5σ. That’s why, in our program, the Gaussian mask generated will be different depending of the value of σ chose by the user. The user will have to choose the value of σ according to the processed image and the desired result. The higher the value, the smoother the filtered image will be, which leads to the disappearance of many edges. At the end of Canny's filtering, the image will only have the most important edges, all the small variations will have disappeared.

On the contrary, if the user chooses a small value, the image will be only slightly smoothed, revealing a lot of details. Thus, on the output image, the number of edges will be much larger than for a high value of σ.

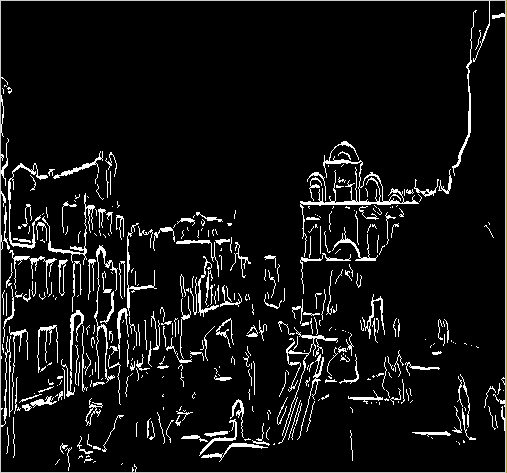
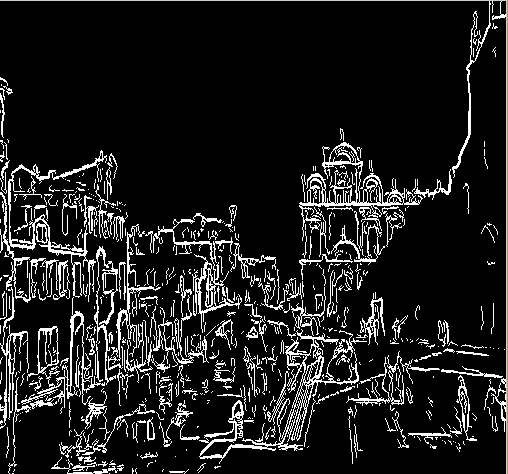
Here are 2 examples to illustrate the above:

Here are the images obtained after filtering Canny of the image "leaf.raw" for three different values of σ.

σ = 0.5 (mask size is 3x3) σ = 1 (mask size is 5x5) σ = 1.5 (mask size is 9x9)

Here are the images obtained after filtering Canny of the image "cana.raw" for three different values of σ.



σ = 0.5 (mask size is 3x3) σ = 1 (mask size is 5x5) σ = 1.5 (mask size is 9x9)

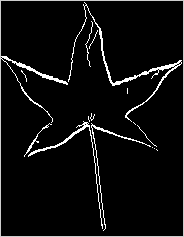
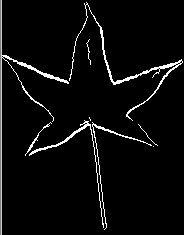
The image "leaf.raw" has few details and few edges. The important information to appear on the output image is the contour of the leaf. Thus, it seems appropriate to take a high value for σ. Only, we also notice that if we choose a value too high as for example σ =1.5, the contours also begin to disappear. That’s why σ = 1 seems to be the best choice.

In contrast, the image "cana.raw" has many details and many edges. To get the best possible result, the goal is to conceal a maximum of these edges so that all the shapes appear on the image at the output of the filter. For these reasons, σ = 0.5 seems to be the best choice.

The choice of the two thresholds:

As explained previously, we use two thresholds (h1 and h2) to classify all the pixels into three categories. From what we have read in the various articles dealing with the subject, it is wise to choose h1 and h2 such that h2 = 2 h1. Pixels whose value is less than h1 will be set to 0 because they are not considered as a part of an edge. Pixels whose value is greater than h2 will be set to 255, it is estimated that their value is high enough to make them part of an edge. Finally, the pixels whose value is between these two thresholds will be set to 127 (subjective value). These pixels will require additional processing to determine whether they are part of an edge or not.

Here are the images obtained after filtering Canny of the image "leaf.raw" for three different thresholds values h1 and h2.

h1 = 50 and  h2 = 100 h1 = 100 and  h2 = 200 h1 = 125 and  h2 = 250

We can see that the threshold choices influence the result (the number of edges that appear on the output image of the filter). The values h1=100 and h2=200 seem to be a good compromise.