Program 2

Due Oct 21, 2019 by 1:15pm **Points** 20 **Submitting** a file upload **File Types** zip

Goals

The goals of this assignment are for you to implement linear filtering and edge detection and observe the differences in edge detection for different levels of smoothing. Note that all of the computations in this assignment will use grey-scale images (not color), so start with the "grey" byte, rather than "red," "green," and "blue."

Linear filtering

As we discussed in class, we can write linear filtering (convolution) using the following equation:

$$O\left(r,c\right)=\sum_{u=y_1}^{y_2}\sum_{v=x_1}^{x_2}K(u,v)I(r-u,c-v)$$

Note that (x_1, x_2) and (y_1, y_2) describe the extent of the kernel in the x and y dimensions, respectively. You should first implement a function that performs such a convolution for an **arbitrary** two-dimensional kernel.

- Your function should take two images as parameters (the kernel should be represented as a small image).
- It should also take as inputs the coordinates of the "origin" of the kernel. Normally, this will be in the center of the kernel, but in some cases, it is ambiguous (such as a 2x1 differentiation kernel).
- Note that the image library allows pixels to store integers and floating-point values instead of red/green/blue/grey bytes (when desired). The inputs and outputs to your convolution routine should use floating-point pixel values to prevent the unnecessary loss of information. You will need to convert the input image to floating point at the start and convert back to grey-scale bytes before writing the image to disk. All other operations should be performed in floating point.
- During the convolution, you should treat any pixel outside the image as the same color as the nearest pixel inside the image.

Iterative smoothing

An effect similar to smoothing with a Gaussian can be achieved by repeatedly smoothing with the following small kernels:

$$\begin{bmatrix} 1/4 \\ 1/2 \\ 1/4 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1/4 & 1/2 & 1/4 \end{bmatrix}$$

Implement image smoothing by convolving an image with both of the above kernels repeatedly. The number of repetitions should be input using a batch file. Useful values range between 0 and 20 or so. You should hardcode these two kernels in your code (and the ones below).

Edge detection

Compute the image gradients in the x and y directions (separately) by convolving with the following kernels:

$$\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Note that convolution with the gradient kernels and edge detection are performed only once after all of the smoothing has been performed. Detect the edges in the image by finding pixels where the gradient *magnitude* is at least 10.0 and the magnitude is at a maximum along the direction of the gradient. (This is non-maxima suppression - see slides from lecture 4.) You will need to perform interpolation on the gradient magnitudes to do the non-maxima suppression correctly. During the non-maxima suppression, if you are at a border pixel and one (or more) of the pixels to interpolate from is outside the image, use the closest location inside the image for that pixel.

Overall, your program should read the input from “test2.gif” and output two images: “smooth.gif” is the image after smoothing; “edges.gif” is the result of your edge detection (edges have intensity 255 and non-edges have intensity 0). Use writeGreyImage to output them to disk (after converting the smoothed image from floating point back to byte). See below for a test image and example outputs.

Resources

- Here is a **flowchart** (https://canvas.uw.edu/courses/1332532/files/58262706/download?download_frd=1) showing how your code should operate.
- Your code should run with this **batch file** (https://canvas.uw.edu/courses/1332532/files/58262699/download?download_frd=1) .
- Here is a new **test image** (https://canvas.uw.edu/courses/1332532/files/58262702/download?download_frd=1) .
- My results after 0 smoothing iterations: **smooth0** (https://canvas.uw.edu/courses/1332532/files/58262704/download?download_frd=1) , **edges0** (https://canvas.uw.edu/courses/1332532/files/58262712/download?download_frd=1)

- My results after 2 smoothing iterations: **smooth2** **(https://canvas.uw.edu/courses/1332532/files/58262707/download?download_frd=1)** , **edges2** **(https://canvas.uw.edu/courses/1332532/files/58262703/download?download_frd=1)**
- My results after 5 smoothing iterations: **smooth5** **(https://canvas.uw.edu/courses/1332532/files/58262710/download?download_frd=1)** , **edges5** **(https://canvas.uw.edu/courses/1332532/files/58262705/download?download_frd=1)**
- My results after 10 smoothing iterations: **smooth10** **(https://canvas.uw.edu/courses/1332532/files/58262708/download?download_frd=1)** , **edges10** **(https://canvas.uw.edu/courses/1332532/files/58262711/download?download_frd=1)**
- For two iterations of smoothing, here are my **gx,** **(https://canvas.uw.edu/courses/1332532/files/58262701/download?download_frd=1)** **gy** **(https://canvas.uw.edu/courses/1332532/files/58262700/download?download_frd=1)** , and **gmag** **(https://canvas.uw.edu/courses/1332532/files/58262698/download?download_frd=1)** images (output using writeFloatImage)