ECEP 596

HW 1 Notes

Overview

- Assignment 1 is a big set of exercises to code functions that are basic and many of which are needed for future assignments.
- Sample functions are provided at the beginning of the code, so you get an idea how to work with the images in Qt.
- The required functions come from the lectures on filtering, edge finding.
- For each function, an **image** argument will be passed. Your task is to modify the **image** according to different functions.

QImage Class
in the QT package

- The Qimage class provides a hardware-independent image representation
- Some of the useful methods
  - QImage() (and other forms with parameters)
  - copy(int x, int y, int width, int height) const
  - setPixel(int x, int y, uint index_or_rgb) can use function qRgb(int r, int g, int b)
  - width() const, height() const
- The QRgb class holds a color pixel.
- from https://doc.qt.io/qt-5/qimage.html

C++ Prerequisite

- Object by pointer (Project1.cpp, line 17):
  - Qimage *image:
    - image->height(); image->width(); image->pixel(r,c);
    - image->setPixel(…)
- Object by reference (Project1.cpp, line 63):
  - Qimage &image:
    - image.height(); image.width(); image.pixel(r,c);
    - Image.setPixel(…)
- Initialization:
  - image = QImage(w/2, h/2, QImage::Format_RGB32);

Double Arrays

- We've modified the original assignment, which had truncation problems when passing images around.
- Instead, you will pass around arrays of doubles.
- The function ConvertQImage2Double() that we provide will convert a Qimage to a 2D matrix.
- The first dimension handles both columns (c) and rows (r), while the second one specifies the color channel (0, 1, 2).
- Position (c,r) maps to r*imageWidth + c.
- This will lead nicely in HW 2, which also uses doubles.
- You don't have to convert back to Qimage!
- You do have to copy any images that you are going to modify.

C++ Prerequisite

- 2D matrix by pointer (Project1.cpp, line 203):
  - double **image:
    - Image[r*imageWidth+c][0] (access the pixel value of it)
  - Note: ***imageWidth*** *and* ***imageHeight*** *are global variables, you can use directly.*
- 1D array by pointer (Project1.cpp, line 203):
  - double *kernel:
    - Kernel[i] (access the value of it)
- New 2D matrix:
  - double** buffer = new double* [imageWidth*imageHeight]

  Note: delete buffer to avoid memory leak

1. Convolution

- The first task is to code a general convolution function to be used in most of the others.
- void Convolution(double **image, double *kernel, int kernelWidth, int kernelHeight, bool add)
- image is a 2D matrix of class double
- kernel is a 1D mask array with rows stacked horizontally
- kernelWidth is the width of the mask
- kernelHeight is the height of the mask
- if add is true, then 128 is added to each pixel for the result to get rid of negatives.

Reminder: 2D Gaussian function with standard deviation σ

In 2-D, an isotropic (*i.e.* circularly symmetric) Gaussian has the form:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

This distribution is shown in Figure 2.

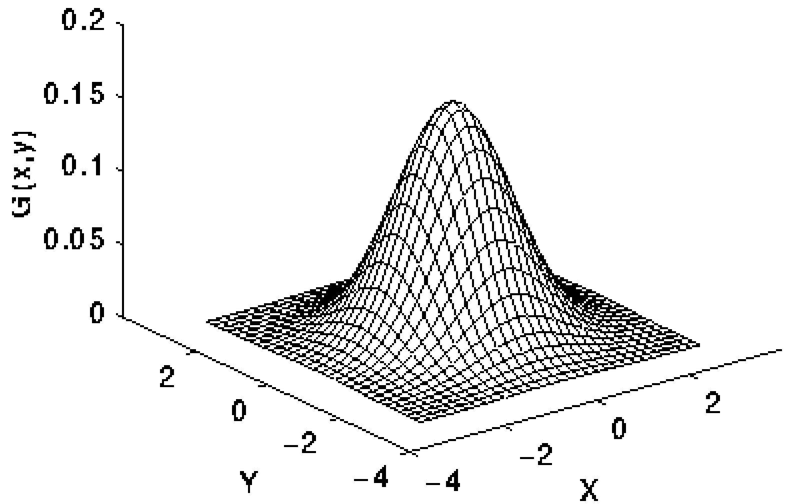

Figure 2 2-D Gaussian distribution with mean (0,0) and σ=1

2. Gaussian Blur

- The second task is to code a Gaussian blur which can be done by calling the Convolution method with the appropriate kernel.
- void GaussianBlurImage(double **image, double sigma)
- Let the radius of the kernel be 3 times σ
- The kernel size is then (2 * radius) + 1

3. First Derivatives of the Gaussian

- void FirstDerivative_x(double **image, double sigma) takes the image derivative in the x direction using a 1*3 kernel of { -1.0, 0.0, 1.0 } and then does a standard Gaussian blur.
- void FirstDerivative_y(double **image, double sigma) takes the derivative in the y direction and then does a standard Gaussian blur
- All of these add 128 to the final pixel values in order to see negatives. This is done in the call to Convolution().

4. Sobel Edge Detector

- Implement the Sobel operator, produce both the magnitude and orientation of the edges, and display them.
- void SobelImage(double **image)
- Use the standard Sobel masks:

-1, 0, 1,
-2, 0, 2,
-1, 0, 1

1, 2, 1,
0, 0, 0
-1, -2, -1