CPE 462 Introduction to Image Processing and Coding

Image Sharpening Technique Analysis

Nikola Ciric and Alexander Gaskins

13 May, 2022

"I pledge my honor I have abided by the Stevens Honor System"

Signed: *Nikola Ciric, Alex Gaskins*

Abstract

The objective of this project is to compare different image sharpening methods, and assess which ones perform better in both specific conditions as well as in general. Through various image sharpening techniques, the goal is to remediate blurry images that can be quite common and annoying to look at.

After delving further into the concept of image sharpening, our team compromised on solidifying our process entirely in OpenCV, rather than working with other IDEs such as MatLab and Jupyter, as was set by our initial approach. The reasoning behind this was that we could acquire the same results using similar processes, and it would be much easier to test our results in one unified interface.

In order to effectively test different image sharpening techniques in C++, a few commodities were necessary. A C/C++ compiler was needed in order to write and execute code that can work with image sharpening techniques, where we decided to use Visual Studio Code due to its systematic interface. Furthermore, the OpenCV library was required to allow for image sharpening functions to be efficiently written and tested on digital images. Conveniently, this was all we needed to begin writing and comparing image sharpening functions and techniques.

The end result of this project was a terminal-based image sharpening executable that features a menu where a user can select which image sharpening method they would like to use, and compare the options to identify which method works best for their respective image. Through experimentation, it has been found that depending on the image, certain image sharpening methods can provide more clarity than others. For reference, the original image used is also displayed in order to allow the user to compare each result. Also, because the sharpening intensity is vital to specific sharpening methods, the user is also given the ability to provide and alter the intensity value depending on how sharp they want the resulting output image to be.

Introduction

A common issue with many self-proclaimed photographers and average iPhone users, is their tendency to capture blurry images. While their hearts are in the right place, taking a clear photo can require a lot of patience, and patience among individuals is often hard to find in today's world. In an effort to remedy this problem, our group: Nikola Ciric [CPE] and Alexander Gaskins [CPE], plan to test various different methods that can sharpen blurry images caused by lens aberrations or misfocus, in order to increase their visibility.

The principal objective of image sharpening is to highlight transitions in intensity. This can be done in a variety of different ways, but let's start from the beginning and consider: how does image sharpening work? Image sharpening uses **spatial differentiation**, where the strength of the response of a derivative operator is proportional to the degree of intensity discontinuity of the image at the point where the operator is applied. When applied to images, edges and other discontinuities such as noise are enhanced, while areas with slowly varying intensities are deemphasized.

As previously mentioned, this technique can be applied through various different methods, but they are all based on the foundation of spatial differentiation filters, denoted by the following formulas:

First Order:

$$\frac{df}{dx} = f(x+1) - f(x)$$

Second Order:

$$\frac{d^2f}{dx^2} = f(x+1) + f(x-1) - 2f(x)$$

These formulas are the backbone behind image sharpening filters, such as the **Laplacian Filter**. The Laplacian Filter sharpens images using a multivariable function derived from the second order spatial differentiation formula, using the operator:

$\nabla^2 f = \frac{d^2 f}{dx^2} + \frac{d^2 f}{dy^2}$ for a two-dimensional image, which corresponds with the Laplacian mask that convolutes with the input image. This process is similar to other image sharpening filters, with the only difference being the mask that is applied.

A common counterpart to image sharpening filters is the use of a **Gaussian Filter**, which is used to remove noise before sharpening an image. As a result of the sensitivity of image sharpening filters, clarity is enhanced when noise is reduced first. This application is often referred to as a **Gaussian blur**, as it literally blurs the image before sharpening it. Its mask is given by the formula: $g(x,y) = \frac{1}{2\pi\sigma^2} \times e^{-\frac{x^2+y^2}{2\sigma^2}}$ where x is the distance from the origin in the horizontal axis, y is the distance from the origin in the vertical axis, and σ is the standard deviation of the Gaussian distribution. It also features a low-pass transform function $X_{uv}' = X_{uv} e^{-\frac{(\frac{H}{2}-u)^2+(\frac{W}{2}-v)^2}{A^2}}$ that attenuates frequency components that are further away from the center (W/2, H/2), and $A = \frac{1}{\sigma}$ where σ is the standard deviation of the equivalent spatial domain Gaussian filter.[1]

An example of the Gaussian blur in use is with an **Unsharp Mask Filter**, which involves subtraction of an unsharp mask from the input image. An unsharp mask is simply a blurred image that is produced by spatially filtering the specimen image with a Gaussian low-pass filter. Thus, the mask function applied is the same as that of the aforementioned Gaussian filter. The size of the Gaussian kernel mask is a function of the parameter σ, and the size of the kernel mask determines the range of frequencies that are removed by the Gaussian filter. In general, increasing the size of the kernel mask causes the Gaussian filter to remove a greater number of spatial frequencies from the unsharp mask image. The unsharp mask is then subtracted from the original image using the equation: $F(x,y) = \frac{c}{2c-1} I(x,y) - \frac{(1-c)}{2c-1} U(x,y)$ where **F(x, y)** represents the brightness value of a pixel at the coordinate **(x, y)** in the filtered image, and **I(x, y)** and **U(x, y)** represent the brightness values of the corresponding pixels in the original and unsharp mask (blurred) images, respectively. The constant **c** controls the relative weightings of the original and blurred images in the difference equation.[2]

By incorporating first order derivative filters, edge detection can be applied to identify regions in an image where the brightness of the image changes sharply. Three operators that can be used to do this are **Roberts**, **Sobel**, and **Prewitt** edge detectors. The Roberts edge detector is one of the first edge detectors used in image processing, and is often referred to as a cross-gradient, as it operates using the concept of cross-diagonal differences. However, because it uses a 2x2 mask, it has some drawbacks in its usage. Namely, it requires more calculations, and considers a smaller number of neighboring pixels in each iteration.

To solve this, the size of the mask must be increased. This change led to the development of the Sobel and Prewitt operators, both of which use a 3x3 mask instead of a 2x2 mask. Unlike the simple diagonal convolution used in the Roberts edge detector, the Sobel edge detector is designed to place more emphasis on the pixels closer to the center of the mask. While the Prewitt edge detector is similar to the Sobel edge detector, its mask does not emphasize certain areas, resulting in the Prewitt edge detector specifically targeting horizontal and vertical edges.[3]

Evidently, there are many methods that can be used to deblur an image, and as shown from the above information, it all boils down to how spatial differentiation is applied to an image. Combinations of multiple methods may even yield a sharper and more immaculate result, such as through the use of both a Laplacian filter and a Sobel edge detector. The possibilities are endless, but through the following experimentation using OpenCV, various common methods will be tested to explore how well they work.

Project Outcomes

The first part of this project was to decide what filtering techniques to use. As mentioned before there are five different methods that can be used by themselves or in conjunction with other methods to help improve the sharpness of an image. The first filter that the team wanted to incorporate was the Laplacian Filter, a commonly used second order derivative filter, which can be described by $\nabla^2 f = \frac{d^2 f}{dx^2} + \frac{d^2 f}{dy^2}$. In order to translate this into code with c++ and openCV, we simply created a function that would house all the necessary code required as seen in the image below. The function will take in two variables, one being the image that would be filtered, and the other an intensity value of the sharpness that the user would input into the driver code. To perform the actual Laplacian filter we simply called the openCV function Laplacian(), which would take the input image and calculate the Laplacian Mask of it. We then scale the output mask to an 8 bit format and calculate the absolute value to make sure we can add it up properly in the future. An additional feature in this function is for the user to be able to perform a gaussian filter smoother filter on the Laplacian mask since one drawback to second order derivatives is that it amplifies noise. This added choice would be up to the users discretion and observe the mask and decide from there. Finally to produce a sharped image we follow the formula $g(x,y) = f(x,y) + c(\nabla^2 f(x,y))$ with f(x,y) being the input image, and ∇^2f(x,y) being the Laplacian Mask. The constant c will depend on if the middle value of the laplacian filter is positive or negative. In our case c has the value -1 because according to the openCV documentation if the filter size used in the Laplacian() function is 1 then a filter with a middle of -4 is used, therefore as according to Gonzalez and Woods in *Digital Image Processing* we should use c=-1. When we combine all this information and apply it to Figure 1 we will get a sharpened image as seen in the Figure 2.[4] As it can be seen there is a clear improvement in Figure 2 such that the image is not as blurry, however it is distorted a little bit due to the sheer detail in the original picture.

Figure 1: Input

Figure 2: Sharpened Image

In addition to the Laplacain filter, there are many other sharpening filters that we can use that are not second order filters so it won't amplify the noise as much. These filters are known as filter order filters and in the case of a 2D image, it would be the gradient. The coding for these three filters will follow a similar style as the second order filter as that everything will be grouped into their own functions to make it easier to call them in the driver code. All three of the gradient filters are mainly used in edge detection applications, however with a very slight modification they can be used to sharpen an image. The first and simplest filter that will be discussed is the roberts filter : $G_x = [-1, 0; 0, 1]$, $G_y = [0, -1; 1, 0]$ As seen in the Roberts function in our code.. As a result of the layout of the filter, the Roberts filter mainly focuses on the cross diagonal differences. Since openCV does not have a built in Roberts filter function, the team had to maily code a definition of the filter in the x and y directions. To perform the convolution we call the built in opeCV function filter2D(), which actually performs correlation, however since the filter is symmetrical it will do the same job as standard convolution. In order to follow the formula for successfully edge detection we have to apply this convolution both the x and y directions and calculate the magnitude by the formula: $mag(\nabla f) = \sqrt{g_x^2 + g_y^2}$. This function can simply be done in openCV by calling the magnitude function and passing in both g_x and g_y and a place to store it called g_{xy}. This new matrix g_{xy} is actually the edge detected image of the input image. In order to use this as a mask and sharpen the input image, the user will be asked to specify an intensity value that will determine how sharp the image is, and finally to get the sharp image the g_{xy} is simply added to the input image as seen in Figure 3. The sharped image uses an intensity of 3, it is clearly less blurry than the input image from Figure 2, however it is slightly brighter and with a greater intensity than 3, white halos around the building would be showing.

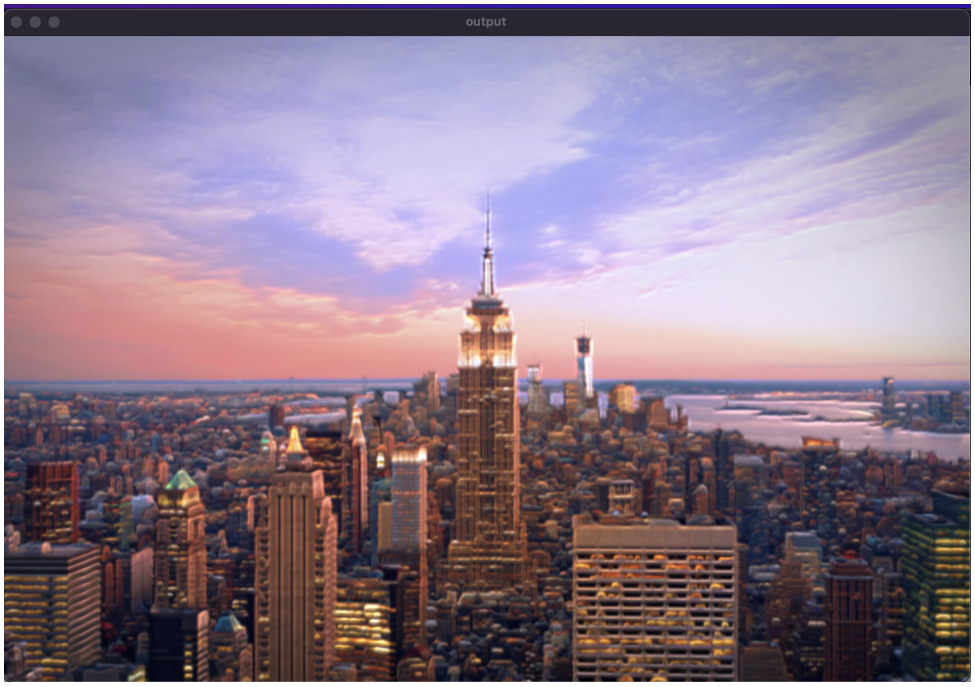

Figure 3: Roberts Shapren with Intensity of 3

The next two filters that will be discussed are the Prewitt and Sobel filters , which are both gradient filters like the Roberts, however they are 3 by 3 filters meaning they will have better edge detection and amplify the horizontal and vertical pixel gradients more than the roberts. The Prewitt operator is defined as $G_x = [-1\ -1\ -1; 0, 0, 0; 1, 1, 1]$, $G_y = [-1\ 0\ 1; -1\ 0\ 1; -1\ 0\ 1]$ and the Sobel operator is only slightly different by having a value 2 pixel near the center and defined as $G_x = [-1\ -2\ -1; 0, 0, 0; 1, 2, 1]$, and $G_y = [-1\ 0\ 1; -2\ 0\ 2; -1\ 0\ 1]$, shown in their respective functions in our code. The set up for both of these functions is identical to the Roberts filter function, in which we take the convolution using both the x and y direction filters of the sobel filters. Now in order to combine them we have to find the same magnitude function as seen in the roberts filter discussion and we do so utilizing openCVs built in function. Finally the user enters an intensity value that is passed into the function call in the driver code. Now that we have this new G_{xy} that is the edge detected version of the input image, we simply add it up with the original input image to get the sharpened image using the Sobel filters. This process is the exact same for the Prewitt filters, the only difference being the values put into the filters. The results of applying these sharpening filters

on the image shown in Figure 2 can be seen in Figures 4 and 5 for the sobel and prewitt filtering respectively. As it can be seen in both Figures the results are very similar to the Roberts filter, however upon closer examination the resulting pictures are slightly more sharp even though a lower intensity value is used. This curious observation is due to the fact that both the sobel and prewitt filters are larger and therefore will work more efficiently in amplifying the edges than the smaller Roberts filter counterpart.

Figure 4: Sobel Filter with Intensity 0.7

Figure 5: Prewitt Filter with Intensity 0.7

The last way that our team has explored the area of image sharpening is by using a technique called Unsharp Masking. This technique was commonly used in darkroom photography but now commonly used in digital image processing. Unsharp masking is currently used regularly in photoshop and the team has implemented it using the OpenCV library. The basis of unsharp masking is blurring the input image and then subtracting the blurred image from the input image, to create a so-called "unsharp" mask. This process can be generalized according by these two equations $g_{mask}(x, y) = f(x, y) - \overline{f}(x, y)$, $f(x, y)$ is the original input image, and $\overline{f}(x, y)$is the blurred image which yields the unsharp mask $g_{mask}(x, y)$. This mask, in reality, is actually the high pass filtered version of the input image. Using this unsharp mask the formula: $g(x, y) = f(x, y) + kg_{mask}(x, y)$, k is some user inputed value to control the intensity of the sharpness, and $g(x, y)$ is the final sharpened image.

This process can be seen in the unsharp Mask function in our code, and the result is shown below in Figure 6:

Figure 6: Unsharp Masking with intensity 15

Overall it can be seen that all five of these techniques work to help unblur a slightly blurry image. However each of them has their own good uses and drawbacks. We specifically choose a picture that has a lot of detail and different colors to try and show what a casual photographer would capture in a day to day life. Based on our results the best technique that yields the least amount of distortion in the final image would be the unsharp masking, as it is the most clear with least amount of distortion but still slightly blurry. The laplacian filter was the best for unblurring the picture the most however it caused some distortion and made the picture look darker. Finally the three first order gradient filters were the best for empshizig the big lights coming for the buildings while also doing a decent job de-blurring the picture, however the user can not control the intensity as much since a lot of noise comes in at intensities of around 5. Furthermore we have included a histogram equalization function in both gray scale and color in case if the user wants emepshive the contrast on the gray scale images or emphasize the intensity values on colored images.

Project Properties

Compilation Guide

To compile this program simply type in the terminal g++ -g -std=c++11 finalTest.cc `pkg-config --cflags --libs opencv4`. This works for mac and linux and we were unable to test it for windows as no one in the group has a windows pc. To run the program type ./a.out and pass the path to the image file you want to sharpen. For example ./a.out input.png. Once running follow the terminal messages to guide you in your image sharpening process.

Source Code

https://github.com/cupokoffi8/SharpenImage/blob/main/finalTest.cc

Team Member Contributions

Overall the project was split evenly between the two members. Alex was working on finding research on different image sharpening techniques and Nikola found sources to help translate that into c++. Nikola was tasked with coding the project with Alex helping to debug. Both Alex and Nikola worked evenly on the report.

References

[1] A. Dogra and P. Bhalla, “Image sharpening by gaussian and butterworth high pass filter,” *Biomedical and Pharmacology Journal*, 03-May-2015. [Online]. Available: https://biomedpharmajournal.org/vol7no2/image-sharpening-by-gaussian-and-butterworth-high-pass-filter/. [Accessed: 05-May-2022].

[2] K. R. Spring, J. C. Russ, M. Parry-Hill, T. J. Fellers, and M. W. Davidson, “Unsharp Mask Filtering,” *Unsharp Mask Filtering - Java Tutorial | Olympus LS* [Online]. Available: https://www.olympus-lifescience.com/en/microscope-resource/primer/java/digitalimaging/processing/unsharpmask/#:~:text=The%20unsharp%20mask%20filter%20algorithm%20involves%20subtraction%20of%20an%20unsharp,a%20Gaussian%20low-pass%20filter. [Accessed: 05-May-2022].

[3] G. N. Chaple, R. D. Daruwala, and M. S. Gofane, “Comparisions of robert, Prewitt, Sobel operator based Edge Detection Methods for real time uses on FPGA,” *IEEE Xplore*, 06-Feb-2015. [Online]. Available: https://ieeexplore.ieee.org/document/7095920. [Accessed: 08-May-2022].

[4] González Rafael C. and R. E. Woods, *Digital ImageProcessing*. New York, NY: Pearson, 2018.