CS 4125 Seminar in Research Methodology for Data Science Coursework C

Dimitropoulos Georgios: 4727657
Manousogiannis Emmanouil: 4727517
Mastoropoulou Emmeleia: 4743539
Group 13

April 8, 2018

Task 1: Gradient-based Image Sharpening

Image filtering is a well know technique that is extensively used in the area of the computer graphics in image processing implementations. These kind of filters have been developed for tasks like image enhancement, feature detection and segmentation. In [1] effective solutions for various image processing tasks are proposed. More specifically, a new saliency-based sharpen filter and a pseudo image-relighting application has been created.
In this work, we are dealing with the problem of the application of a gradient-based image sharpening on a blurry image. Hence, the main idea is to firstly compute the gradients g_k, after that re-scale them and finally to construct a new (sharper) image whose gradients best match the rescaled gradients. Thus, the input of the algorithm is a blurry image and the parameters c_s, c_u which affect the scaling of the gradients and a term that tries to penalize the deviation from the input image. The output is the sharpened image.
In general, we could say that an image gradient corresponds to a directional change in the intensity or color of an image. Thus, image gradients are used to extract information from images. As a pipeline of the algorithm which we adopt we could say as mentioned before that firstly calculates the gradients, after that rescales these gradients and finally constructs a new (sharper) image whose gradients best fit the rescaled gradients. The basic steps of the algorithms which we adopt are presented below:

Algorithm: Image sharpening

Input: U^* (Pixel values of input image) and the parameters $c_s, c_U{}^*$
Step1. Construction of the gradient matrix G
Step2. Computation of the gradients of the input image $g^* = GU^*$
Step3. Computation of the solution of the minimization problem $min_U(||GU - c_s g^*||^2 + c_U{}^*||U - U^*||^2$
Step4. Return minimizer U

Implementation

Gradient Initially, the blurry image is loaded and saved in a matrix with dimensions $h * w * d$. The variable h corresponds to the height of the image whereas w corresponds to the width of the image and finally d equals three which is the depth of this image. Namely, variable d corresponds to three different channels (RGB) of our colored image. Furthermore, the parameters c_s and c_u are determined as are going to be used in the rescaling part which we mentioned above. Finally, this matrix (our image) is transformed from the 3D tensor to a 2D array in a row-wise manner with w*h rows and d columns.
In the sequel, the gradient function is implemented in order to be able to to construct the gradient matrix G which contains the image gradients. This function takes as an input the variables h and w which corresponds to the height and the width of the source image. We should point out here that the whole procedure is firstly taking place only for the first channel, as after the implementation

of this gradient function, we also apply this function for the other two channels in order to be able to receive a colored image as output.

As an overview here, we could mention that image gradients correspond to the difference of the neighboring pixels of an image matrix in the right or left direction of this pixel (horizontal differences) either in the upper or down direction (vertical differences). In order to be able to track this procedure, we keep the index k of every difference with $k \in 1, 2, ...m$. After that, we initialize three vectors i,j,v with zero values and dimensions $2*h*(w-1)+2*w*(h-1)$, since the total number of the aforementioned differences equal the total number of columns and rows minus one each time, multiplied by total the number of rows or columns respectively. We should mention here in order to give a clearer insight that the vector i contains the row indicators, vector j contains the column indicators and finally vector v contains the values $\frac{-1}{2}$ and $\frac{1}{2}$ in the above corresponding indices i,j. Initially the rows of image matrix were parsed in order to calculate the horizontal differences for each row.The number of rows as we mentioned before is h*(w-1) and were parsed since the pixels in the last column do not have a right neighboring pixel. After that,in a similar manner, the columns of the image matrix were parsed in order to get the vertical differences for each column. Again the number of rows that were parsed is w*(h-1) since the pixels in the last row have no downward neighboring pixel. In the sequel, we used the sparse function which creates a sparse matrix G with the aforementioned vectors i, j, v. A sparse matrix is adopted since in a form of compact data structure, enables us to avoid storing the numerically zero entries in the matrix [2].

Task1. After the implementation of the the gradient function, the function Task1.m calls the gradient.m function with arguments h,w in order to obtain the gradient matrix G. Our next step would be the computation of vector g which contains the image gradients based on the equation $g^* = GU^*$. The main role of the function Task1.m is to solve a linear system in order to be able to find the minimizer U which finally returns the sharpened image. The linear system which we solve is given by: $(G^T G + c_U{}^* Id)U = c_s G^T g^* + c_U{}^* U^*$. The resulting images before and after sharpening are depicted below:

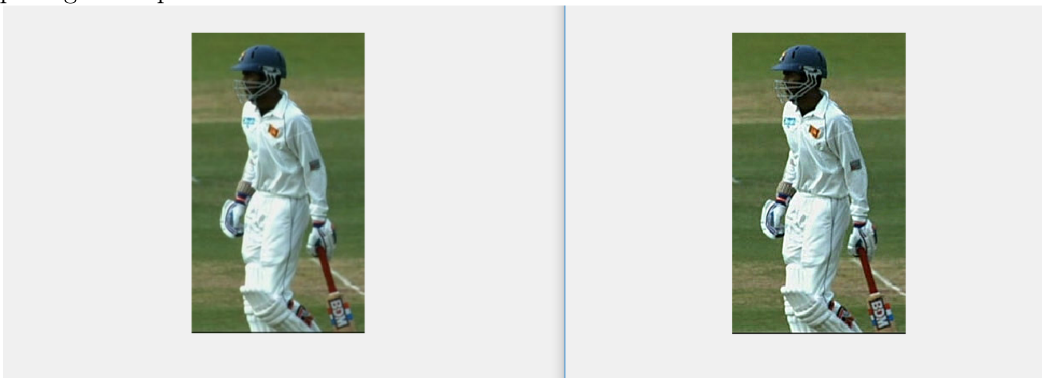

Figure 1: Blurry Image(Left) and Sharpened Image(Right)

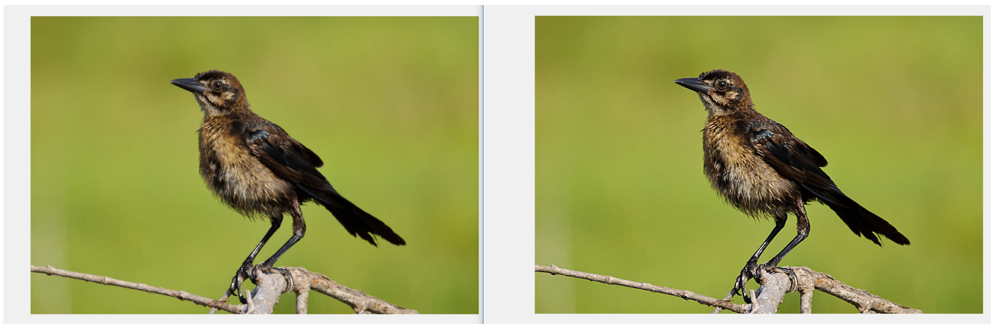

Figure 2: Image(Left) and Sharpened Image(Right)

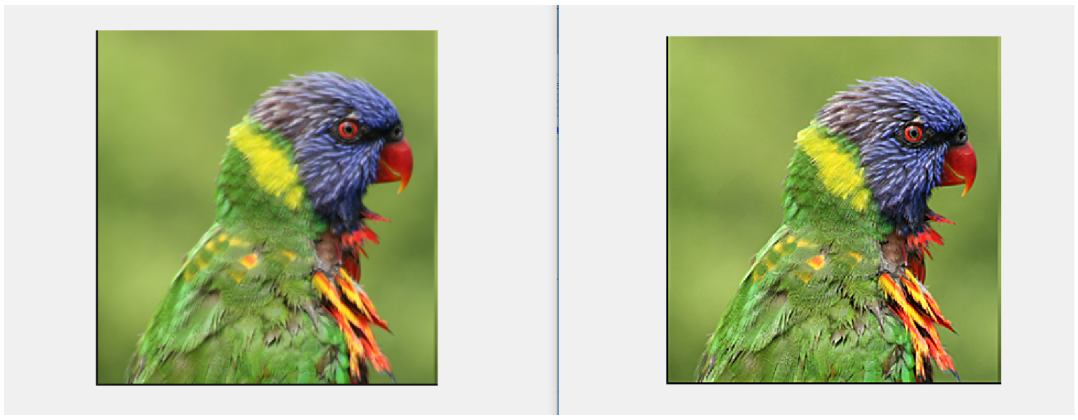

Figure 3: Image(Left) and Sharpened Image(Right)

Task 2: Gradient-based Image Blending

In the Gradient-based Image Blending our target is to blend two images by creating a new blended image that contains both the source image and the target image. This specific implementation is based on the paper 'Poisson Image Editing' Pérez, et al [3]. The goal of this assignment is to blend a source image picture into a target image as well as possible. The simplest method of source would be to just copy and paste the pixels from one image directly into the other. Unfortunately, the differences in this case will be very notable, even if the backgrounds are similar. In order to get rid of these seams we try to create a mixed seamless cloning process which relies on mixing gradients from the source and destination for a better blending.

Algorithm: Gradient-based Image Blending

Sometimes it is necessary to keep some features of the destination image when performing cloning. When the objects that we need to insert contain holes, are slightly transparent and the background of the destination is highly textured then it is better to mix the gradients of the source and destination images. The solution for this problem involves using mixed gradients. At each point in area of interest where changes will take place, we keep the stronger of the variations in destination or in source, based on the following. Using a 4-connected neighborhood in a 2d image, the equations below denote that for a single pixel in our image, at coordinate (i,j) which is fully under the mask(in our case the Whole source picture) we would have the following equation:
$|N_p|f_p - \Sigma_{qinN_p} f_q = \Sigma f_q^* + \Sigma v_{pq}$,

where

$|N_p|$ is the number of neighbors. In our case 4.

f_p,f_q is the value of pixel p and q respectively.

and v_{pq} is the gradient between p and q
We try to combine these the sum of the above equation in a way that it is representative for each pixel, as this can make the our computations for each final pixel value much easier.After implementing the above equation for the 4 neighours of each pixel in the x and y axis, we get the following for each i,j pixel in pur final image:

$4 * f(i,j) - f(i-1,j) - f(i+1,j) - f(i,j-1) - f(i,j+1) = 4 * s(i,j) - s(i-1,j) - s(i+1,j) - s(i,j-1) - s(i,j+1)$

where $s(i,j)$, is the result of the mixed gradient at point i,j.
Hence we are trying to solve a linear system of type $Ax = b$, where x is a vector with all pixels of the blended image, b is a vector containing all the mixed gradients of the blended image and A is a sparse matrix. The content of the sparse matrix will be such, that it results in a system where each pixel value multiplied by 4 minus all neighboor pixel values will be equal to the mixed pixel's

gradient. This is based on the formula we described before.The implementation of this equation is in the MixedGradients.m file.

Main

Initially we read and convert the source and target images to double precision. The next step of the algorithm is to give to the user the interactive opportunity to specify the polygonal region of interest in the destination image, where the source image will be placed. This is done with the use Matlab roipoly function. The result of the ROI function is a logical image displayed in figure 5. The values of this image are 1 in the selected region of interest and 0 anywhere else. After that, we rescale the source image to fit in the size of the selected region and then pad it with zeros in order to have the same size with the destination image (figure 4). Having completed the above steps,we then call the MixedGradients function with the previous two images and the destination image as input and as output we return the computed blended image. The MixedGradient function is described below.

MixedGradients

Input: : i) The source image in the selected region of interest of the target image, padded with zeros ii) The mask that determines in which position of the target the source will be blended into iii) The target(destination) image
Step 2: We initialize a sparse matrix G and and a vector g, which will contain all gradients of the computed blended image. The length of the vector is equal to the number of pixels that need to be 'blended', so it is equal to the number of logical 1 of the ROImask matrix.There is no need to compute the rest, as their value will be equal to the value of the destination image of course.
Step 3: We store the indices of all pixels that are located inside the region of interest, where the source will be blended into.
Step 4: We compute the values of each element in the sparse matrix, based on the algorithm described in the previous section. We add 4 value in the position that corresponds to the pixel itself and -1 value for each one of the 4 neighbor pixels in the y or x axis, if existing.
Step 5: We compute the gradient for each pixel of the blended final image. The gradient in the pixels outside the ROI will be equal to the gradients of the destination image. On the contrary, the values of the gradients inside the ROI will be equal to the gradient that has the largest absolute value, between the source image and the original destination image at this pixel.
Step 6: After computing G and g , we now simply have to solve the linear system $GU = g$. Our final solution is of course $U = \frac{g}{G}$.
Step 7: We combine our blended image U with the 'out of ROI' destination image and we return it.

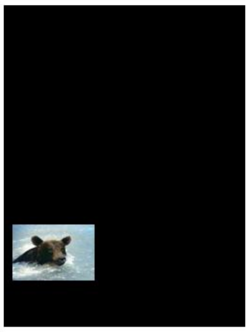
Figure 4: Adapted source image

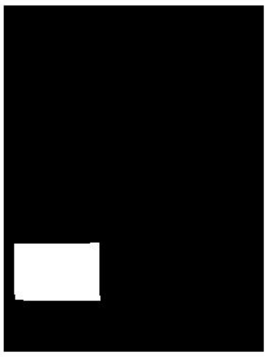
Figure 5: Region of interest in the destination image

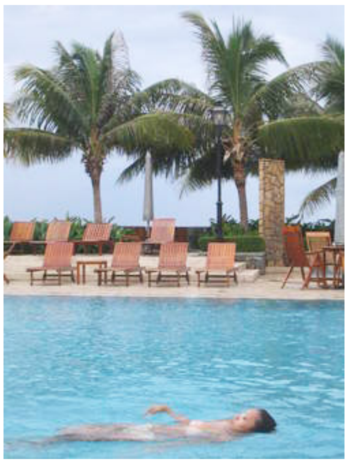
Figure 6: Destination Image

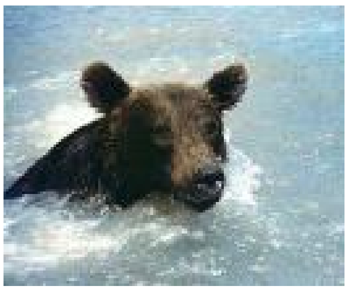
Figure 7: Source Image

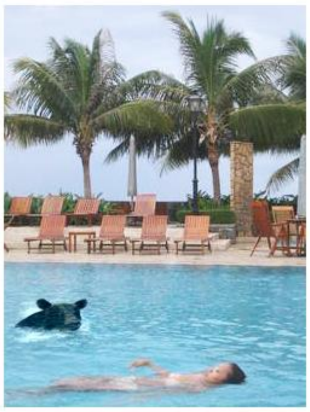
Figure 8: Blend Image

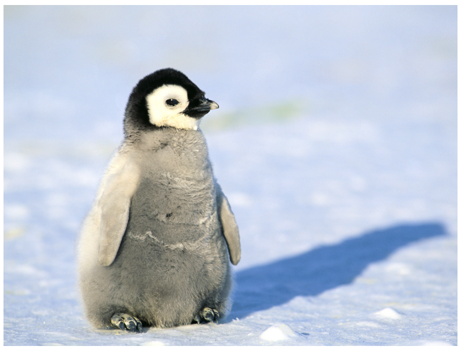
Figure 9: Source image

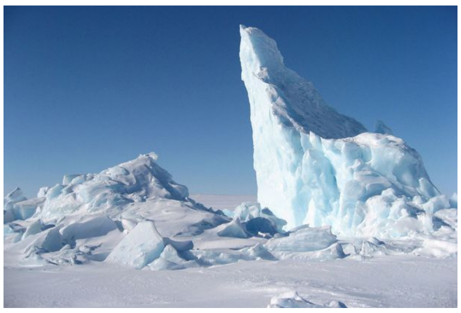
Figure 10: Target Image

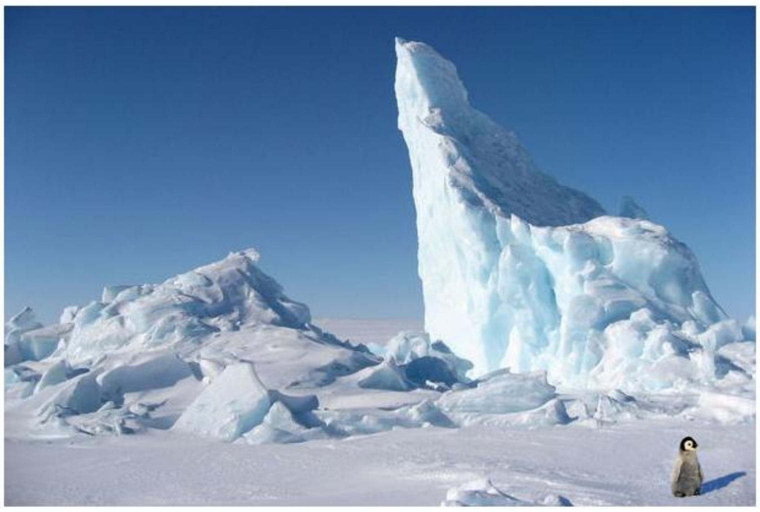
Figure 11: Blend Image

References

[1] Bhat, Pravin, et al. Gradientshop: A gradient-domain optimization frame- work for image and video ltering. ACM Transactions on Graphics (TOG) 29.2 (2010): 10.

[2] Davis, Timothy A., Sivasankaran Rajamanickam, and Wissam M. Sid- Lakhdar. A survey of direct methods for sparse linear systems.Acta Numerica 25 (2016): 383-566.

[3] Patrick Perez, Michel Gangnet Andrew, Blakeá, Microsoft Research UK: Poisson Image Editing (2003)