CS4125 Seminar Research Methodology for Data Science - Coursework Assignment C

Joe Chen	4703308	P.Chen-2@student.tudelft.nl
Kin Lok Chow	4509447	k.l.chow@student.tudelft.nl
Jos Smalbil	1362933	p.j.smalbil@student.tudelft.nl

April 8, 2018

1 Introduction

We are asked to implement in MATLAB, methods for gradient-based image sharpening and gradient-based image blending. We show our implementations in section 2, describe the functionalities of our implementation and it's usage in section3, show that our implementation performs in a correct way in section 4 and discuss the benefits and limitations of our methods in section 5.

2 Implementations

2.1 Task 1: Gradient-based Image Sharpening

We implemented gradient-based image sharpening in MATLAB using the template provided 'myTask.m'. we construct the algorithm as follows:

1. **Construct gradient matrix G**
   Gradient matrix G selects certain pixels for computing the gradients(both vertical and horizontal gradients) in the next step. Owing to the fact that in each row of G we only calculated one gradient, there will be lots of zeros in the matrix, which wastes the memory. Thus, we create the gradient matrix as a sparse matrix only reserving the corresponding pixel for calculation.

2. **Compute gradients of input image $\bar{g}$**
   We have had the gradient matrix G for computing the input gradients, we implement the formula $\bar{g} = G * \bar{U}$ to obtain the gradients.

3. **Minimize the loss of the sharpening image**
   Here we compute the loss of the sharpened image as: $\min_U ||GU - c_S\hat{g}||^2 + c_{\bar{U}}||U - \bar{U}||^2$, where $\bar{U}$ indicates the pixel values of input image, U as the pixel values of the minimized output image. Note that this formula just minimizes the difference between the input and output image and between the input and output gradient with respect to some tuning parameters. Eventually, as written out in the lecture slides, the loss-minimized image could be considered as solving a linear system $AU = B$ where $A = G^TG + c_{\overline{U}}Id$ and $B = c_sG^T\overline{g} + c_{\overline{U}}\overline{U}$. The output image can be computed as $U = A^{-1}B$.

4. **Return sharpened image**
   Finally, we reshape the image to it's original shape, show the sharpened image compared to the original image and write out the sharpened image to a new image file.

In the algorithm decribed above, we construct the gradient matrix G by first iterating row-wise over the dimensions of the input image and subsequently iterating column-wise. The parameters c_S and $c_{\bar{U}}$ are set as 3 and 0.5 respectively.

2.2 Task 2: Gradient-based image blending

In this part, our purpose is to paste certain objects in one or more images to selected regions of another image. Here we implement gradient-based image blending method to achieve the task. We also need to construct the gradient matrix of the input image, however, this time the gradients are separated into two types: **inner gradients** and **boundary gradients**. Inner gradient indicates the gradients of the pixels which are not the boundaries. And the boundary gradients are the gradients of pixels associated with the boundaries. We need to compute the color difference between our source image and the target(boundary gradients) in order to smooth the image. Our algorithm is described as follows:

1. **Construct gradient matrix G, SG**
   While creating the gradient matrix, we will consider which type of gradients they are. G is the matrix with calculation of both inner and boundary gradients, and sparse matrix SG is gradient matrix of the boundaries. These will be used for the step of separating the inner gradients and the boundary gradients. Here we implement similar methods in the former section for constructing G and add some additional statements inside for constructing SG.

2. **Construct selector S selecting the boundary pixels**
   The use of the selector matrix S is to point out where the boundaries are. This helps us to separate the boundary for computing the boundary gradients and to reconstruct the image as well. Because there will be lots of zeros in the selectors as well(each row only indicates one pixel). Hence, we also construct it as a sparse matrix.

3. **Compute image vector U, U^b**
   The image vector U contains the value of each pixel in the source image, and U^b only contains the pixel value of the boundaries. U^b can be obtained by computing $U^b = S \times U$

4. **Compute the gradients of the source image that stores the source inner gradients**
   The gradient vector $\hat{g}$ now should only have the inner gradients. Therefore, our gradient matrix cannot contain the information of boundary gradients. This kind of gradient matrix can be achieved by dividing G and SG in the previous step. Then the intended vector $\hat{g}$ is obtained as : $\hat{g} = (G - SG) \times U$.

5. **Solve soft constraint problem** We solve the linear function provided by the lecture to obtain our blending image:
   $$(G^T G + \alpha S^T S) \times U = G^T \hat{g} + \alpha S^T U^b \tag{1}$$
   where α is a parameter deciding the weight of our boundary pixel. In the following sections we set $\alpha = 0.9$ for the experiments.

In our work, we separate our code into two situations: (1) having only the source images, and (2) having both source images and their masks. We did not implement the methods of detecting the objects in the source image; therefore, in the first circumstance, we allow the users to select the region(rectangle) for cropping and stitching. On the second circumstance, since we have the mask of the source image, we just directly crop the region off and blend it into the selected region in the target image. This selected region can either be defined by an offset or by letting the user select a position in the target image. We also allow users to stitch multiple source images to the target image by looping the all process repeatedly. The process flow of our work is illustrated in Figure.1.

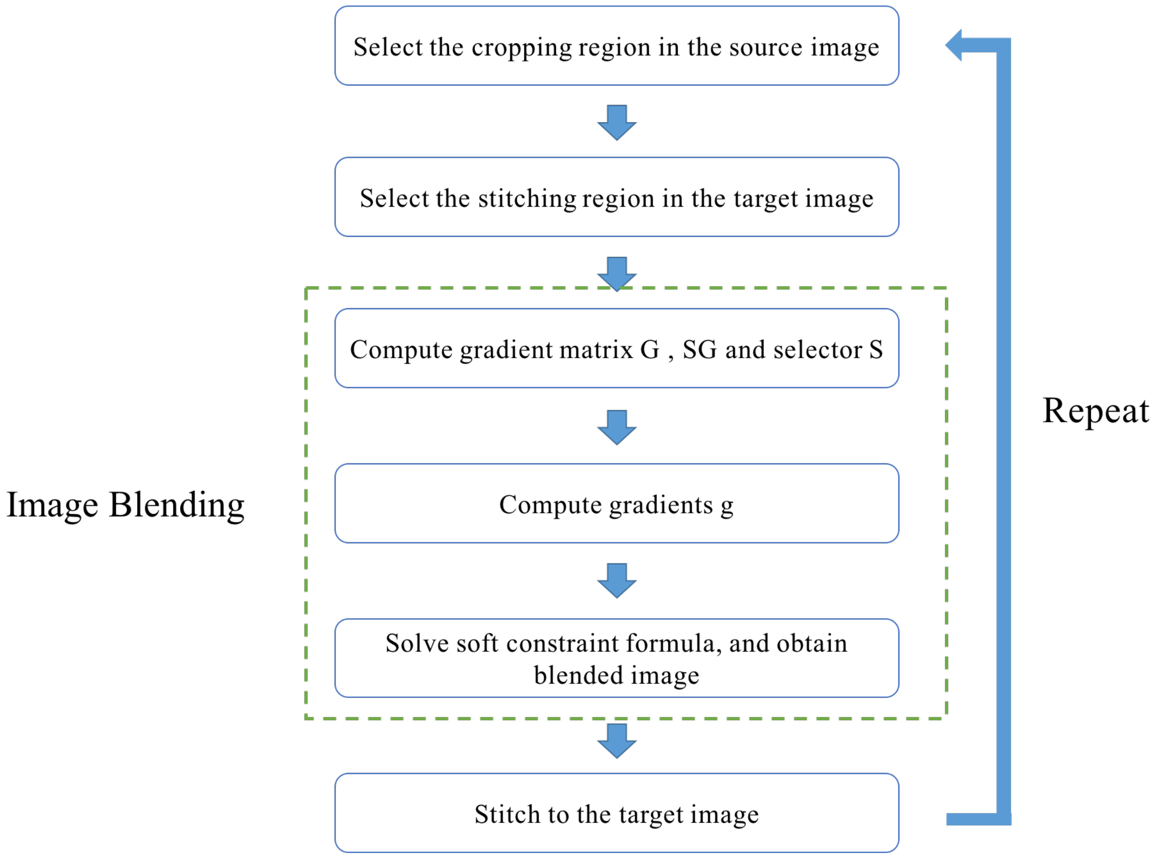

Select the cropping region in the source image
Select the stitching region in the target image
Compute gradient matrix G , SG and selector S
Compute gradients g
Solve soft constraint formula, and obtain blended image
Stitch to the target image
Image Blending
Repeat

Figure 1: Process flow of the gradient-based image blending function

3 Functionality and usage

3.1 Task 1: Gradient-based Image Sharpening

Our implementation for gradient-based image sharpening can be applied to any image of small to moderate size. Note that for images of moderate size the calculation might take some time (few minutes) while for images of larger size (250x250 and above) MATLAB may give an 'out of memory' error.

This implementation could be used in any image-editing software package that wants to add an image sharpening option. The sharpening parameters c_u and c_s could be set by the user itself or the software could choose some default values.

3.2 Task 2: Gradient-based image blending

As mentioned in the previous part, instead of cropping the whole source image to a predefined area in the target image, we provide a functionality in MATLAB in which the user can select any region of the source image for cropping and subsequently selects the region in target image in which the source image is then automatically blended at the corresponding region in our system. However, in MATLAB we are not able to load different image automatically, the user should change the image path manually to stitch the source image to the target image. This is the problem which could not be solved in MATLAB, which makes the system work inefficiently at this point. All in all, our system enables the user to manually select the position/region to blend the source image into the target image and perform well. Some examples of single image blending are shown in Figure 2 and 3, and the multiple stitching part is shown in Figure4

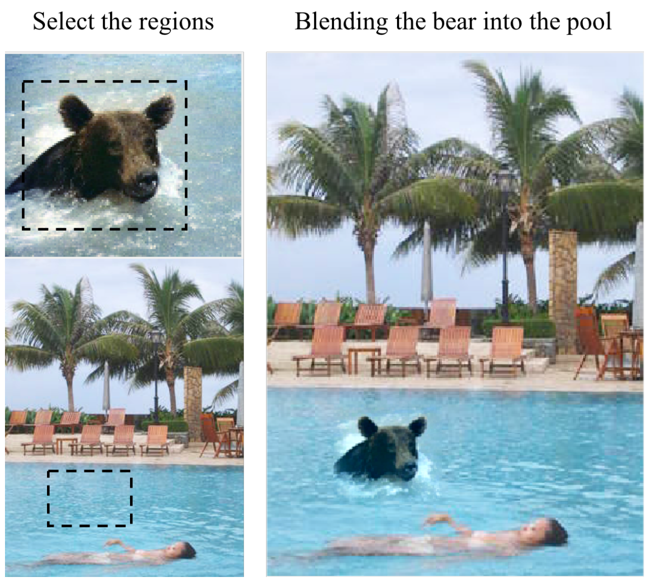


Figure 2: Example of blending bear into the pool

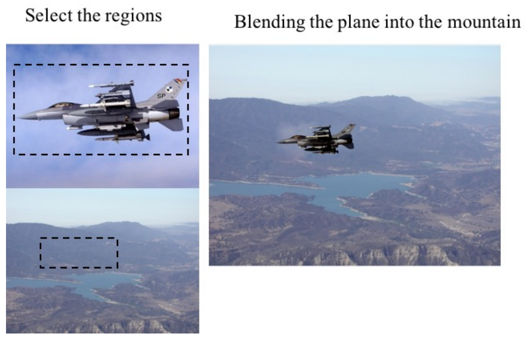


Figure 3: Example of blending the airplane into the mountain

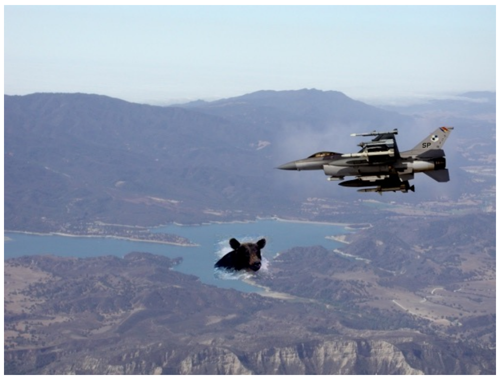


Figure 4: Example of blending both the airplane and bear into the mountain

4 Correctness tests

4.1 Task 1: Gradient-based Image Sharpening

We performed some correctness tests for the gradient-based image sharpening by using the picture of the baseball player as provided per example. We see that our results are similar as in the lecture slides. When performing image sharpening using the MATLAB implementation as explained in 2.1 with parameters $c_s = 3$ and $c_u = 0.5$ we get a well sharpened image of the baseball player 5 while with parameters $c_s = 10$ and $c_u = 0.5$ we get a picture that is sharpened somewhat too much 6, which is as we've expected.

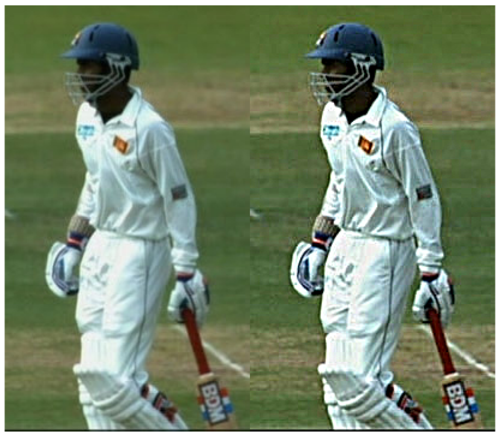


Figure 5: Original image (left) vs sharpened image (right) with $c_s = 3$ and $c_u = 0.5$.

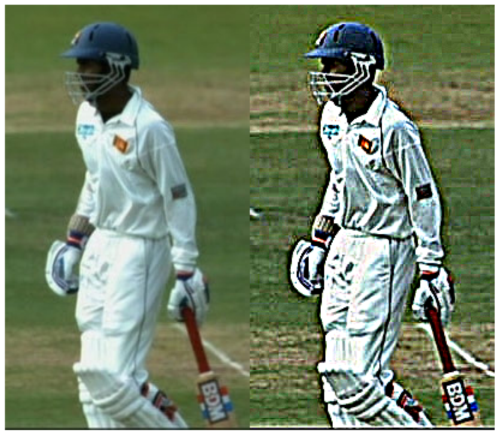


Figure 6: Original image (left) vs sharpened image (right) with $c_s = 10$ and $c_u = 0.5$.

We also tested the image sharpening on other random images, in the below case on a picture of Donald Trump in front of his airplane. The suit, face and plane of Donald Trump are all sharper in our image so from this we can empirically conclude that the image sharpening also works as expected for random images.

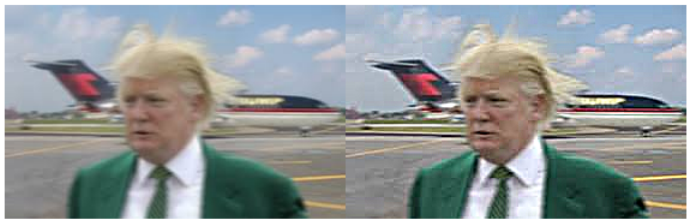


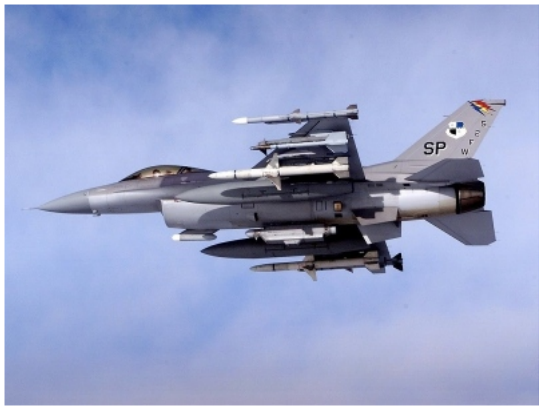
Figure 7: Original image (left) vs sharpened image (right) with $c_s = 3$ and $c_u = 0.5$.

4.2 Task 2: Gradient-based image blending

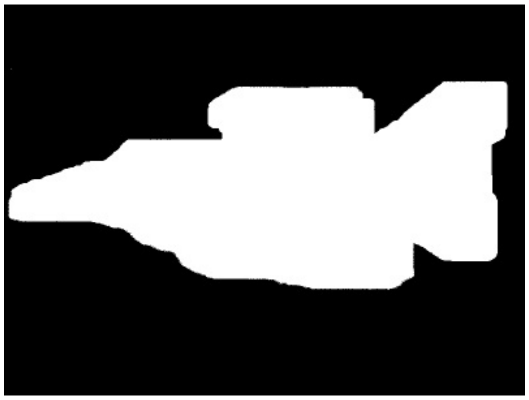
For the gradient-based image blending we again first tried the provided example images. Using the MATLAB implementation as explained in 2.2 and the result is shown in Figure2, where the weight parameter $\alpha = 0.9$. Although generally the bear is blended into the pool well, we still can observe that the wave around the bear in the source image does not fit the pool. This phenomena can also be observed by the airplane image shown in Figure3, the cloud around the airplane still exists in the blended image. Although this effect is not that severe, we still want the image to be real enough. Therefore, we tended to implement mask to eliminate the effect from the outliers. In our system we do not implement the method of detecting and forming the mask. Thus, we just downloaded the mask, and compare whether the effect will be eliminated by using the mask. How to detect the object and to create the mask will be left to the future.

Image blending using the mask

Figure 8 shows the shape of the mask compared to the original source image. We select this mask as our boundary, which is not rectangular now. By cropping the source image based on the mask, we are able to remove most of the background which does not belongs to the object. The blending performance is shown in Figure 9. As we can see right now, the source image fits well to the target image.



(a) Original source image



(b) Mask of the source image

Figure 8: Original image and the mask of the source image

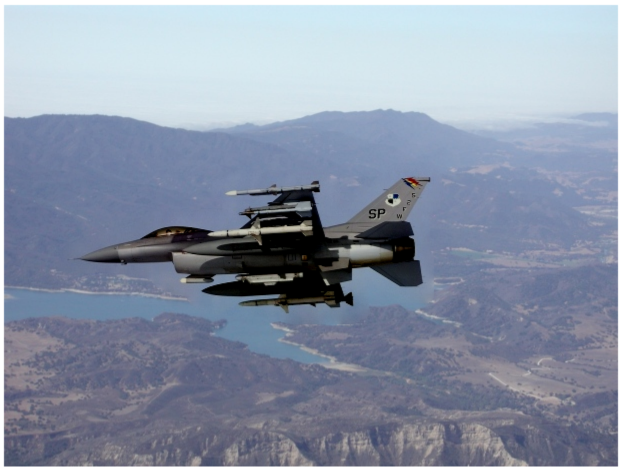


Figure 9: Blended image using the mask with $\alpha = 0.9$.

In addition, we also implement the image provided from the files for more testing. We create the mask of the bear image and use it to blend into the pool, the performance is shown in Figure 10. Compares to the rectangular boundary, using mask indeed will increase the performance of the image blending by eliminating the unnecessary background.

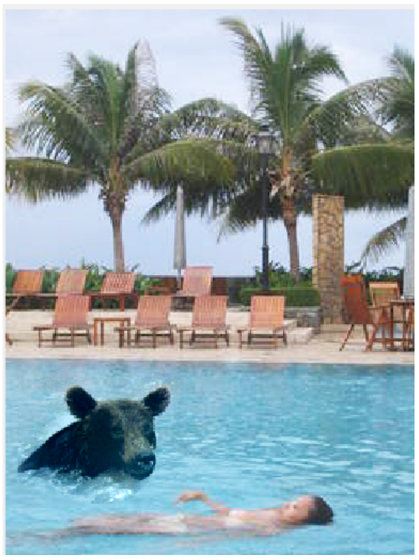


Figure 10: Blended image combining with the mask with $\alpha = 0.9$.

5 Evaluation

5.1 Task 1: Gradient-based Image Sharpening

The benefit of our gradient-based image sharpening method is that it can be applied to any moderate sized image and this can be done using arbitrary sharpening parameters. One limitation of our implementation is that it fails to sharpen large images of size greater than 250x250 because of memory issues. Another limitation is that for moderate sized images the calculation to obtain the gradient matrix may take some minutes of time.

5.2 Task 2: Gradient-based image blending

As we mentioned above, the benefits of our gradient-based image blending method are that it can be applied to any two images with arbitrary blending parameter and additionally the calculation is performed in nearly instant time for images up to moderate-size. Also, we are able to select arbitrary region of our source image and blend it to an arbitrary selected region in the target source, which makes it much flexible and user friendly. Thus, the method we implemented could very well be used in any image-editing software that wants to add image blending options.

There are some limitations that if the user does not have the mask of the source image, the performance will decrease because of the redundant background. If the user has the mask of the source image, the boundary effect caused by the background will decrease, which makes the blending image look more smoothly. However, in our system we didn't implement methods helping the user detect the object and create the mask. Thus, the performance of our blended image right now really depends on whether we have the mask or not and whether the background in the blending area and the source image are complex or not. Figure 11 demonstrates our evaluation. Because the background of the sky is simple, we could easily blend the balloon and a sun well into the target image.

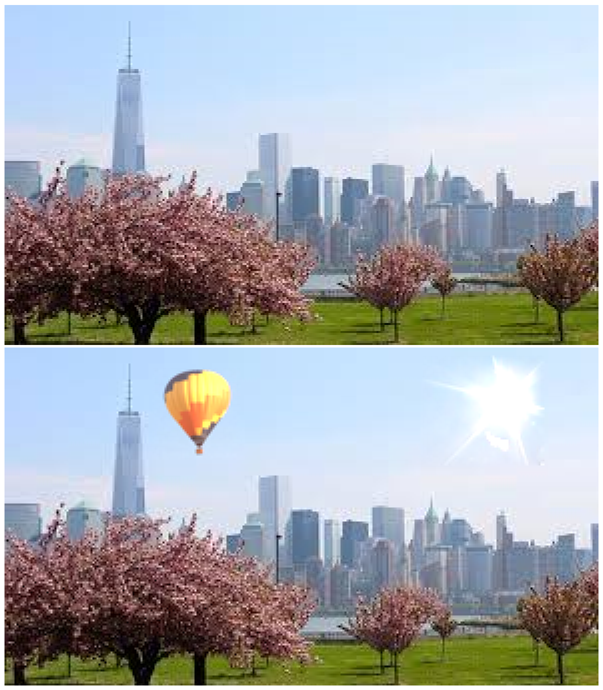


Figure 11: Example of blending a balloon and a sun into an empty sky

Another limitation in our method is that the user has to manually select whether he or she will use a mask for the source image since our implementation does not take this into account. Another limitation is it's user friendliness, although the (region selection) steps the user needs to perform are as expected and quite straight-forward it might be confusing for new users or users unknown to MATLAB.

5.3 Conclusion

We can conclude that our MATLAB implementation of both image sharpening and image blending could be used in any image-editing software that wants to add sharpening and/or blending as image editing options for images of up to moderate size. Our advice would be to use a kind of small painting brush for the image sharpening because of the amount of calculation time our method takes for larger areas. Our image blending method could be implemented for an arbitrary amount of moderate sized source images, they can be blend into an arbitrary large target image automatically using our MATLAB implementation.