Mahdie Ghane Assignment2_CMPT 769

In this assignment, we are going through Poisson Image blending approach for editing image. For doing so, we need to implement the Poisson equation that has been elucidate in [1]. To make it shorter, the Poisson algorithm edits the blended image by putting the Laplacian at every pixel equal to zero since human's perceptual is more sensitive to gradient rather than . In this circumstance, if we consider 4 neighborhood, the above constraint become to:

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

Equation 1: Poisson equation on Masked pixels[1].

If all of the pixel and its neighbors are under the mask; otherwise, the value at that pixel would be replaced by corresponding value in the target. Here V is the pixel in the final image.

To implement Poisson blending method, at the first time, I used high-computational method in which I check every pixel to see whether this pixel is under the mask or not to generate Matrix A. Then for each channel, I computed the value of b. However, after that I used mentioned advice from the assignment's homepage and defined Matrix A as a sparse. After that, I saved indexes that are under the mask using sub2ind.h function. Then these index have been used to create equation (1) by replacing values. In this regard, if the pixels doesn't include in the mask (Mask value=0), The corresponding coefficient in the A matrix would be zero since this pixels will get the target's value. However, target value at that pixel will contribute to make a matrix b. After generating A, b, we can use / which calculate Matrix x with Less MSE error while Ax=b. Matrix x, gives us the final values in every pixel. It should be mentioned that for all channels, Matrix A is similar and only matrix b need to be created separately. After calculating Matrix x for every channel, I have concatenate them together to make a desired output. By following this algorithm I could successfully get these outputs from images 1:5:

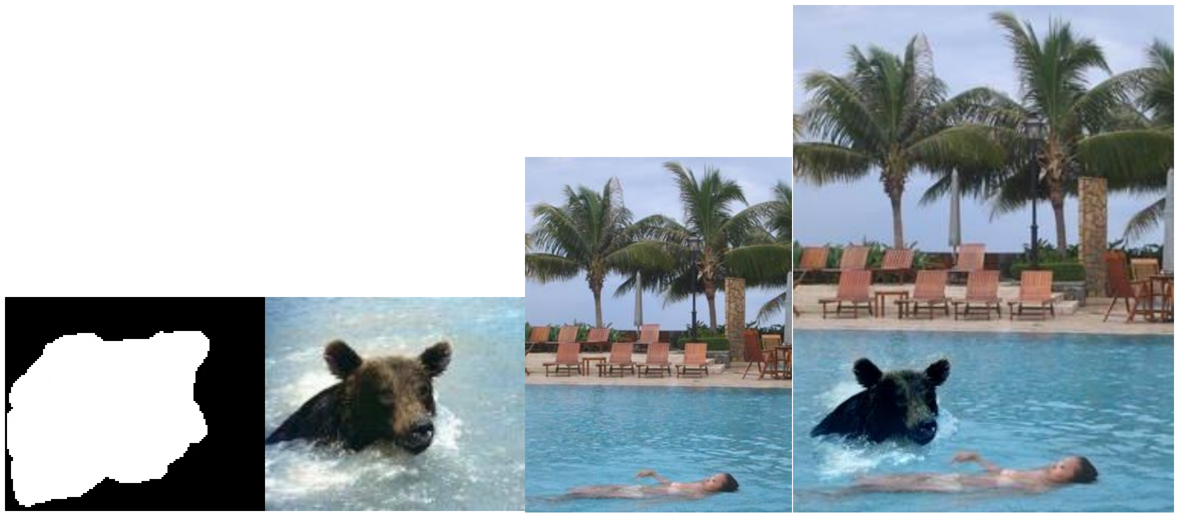


Figure 1: from left to right: Mask1, source1, target1, merged blended result.

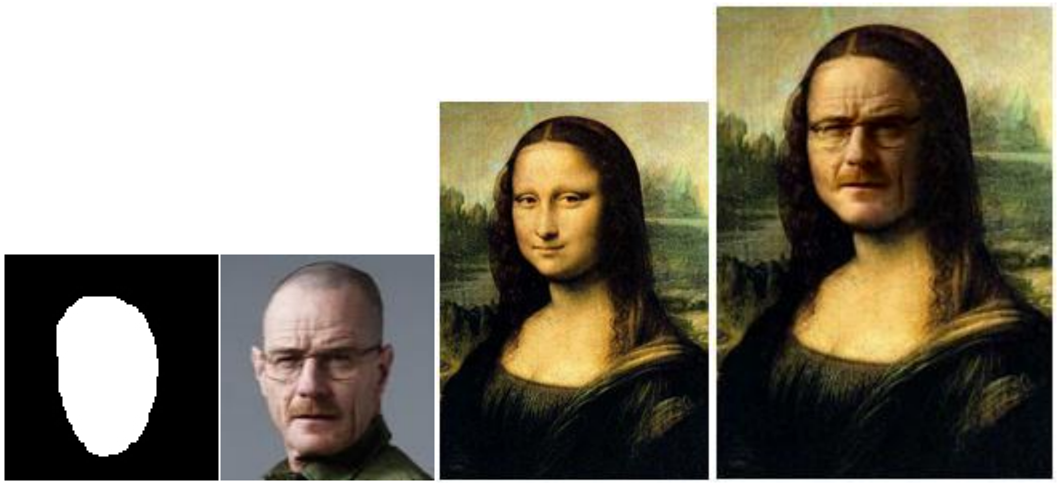


Figure 2: from left to right: Mask2, source2, target2, merged blended result.

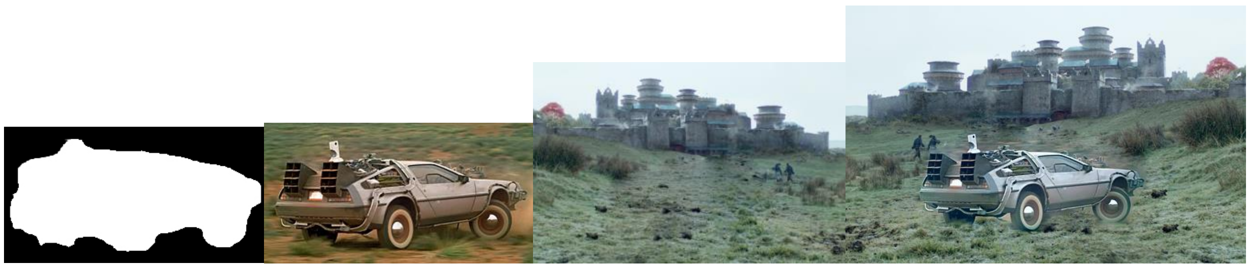


Figure 3: left to right: Mask3, source3, target3, merged blended result.

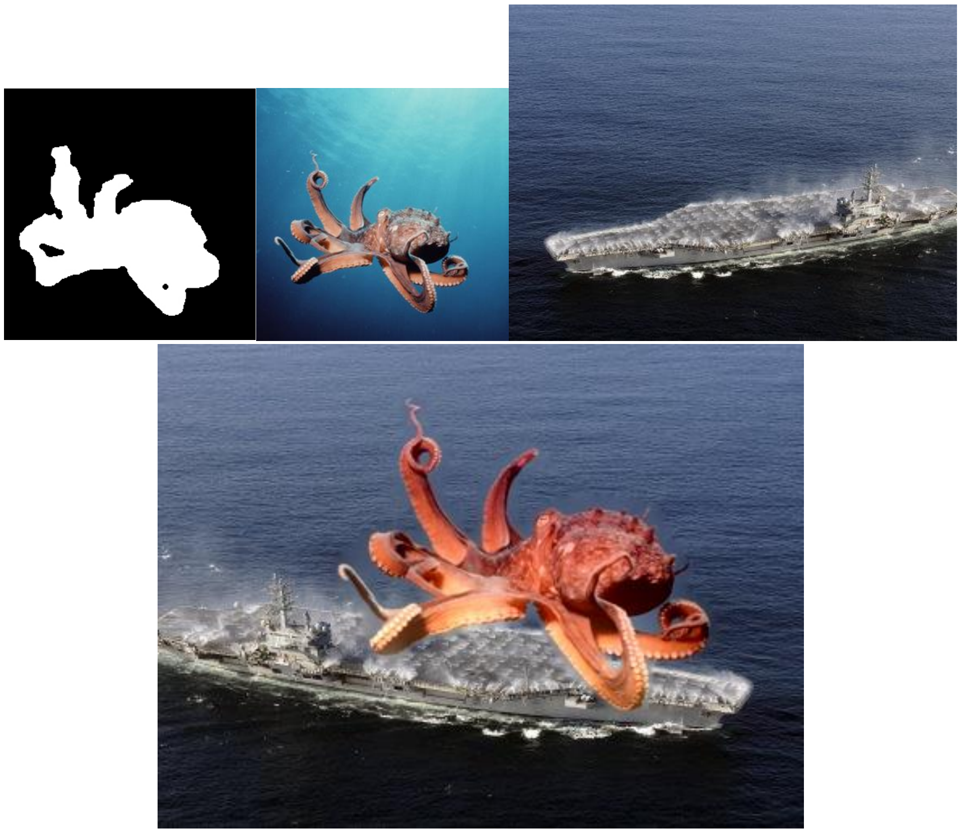


Figure4: Top Mask4, source4, target4. Bottom: merged blended result.

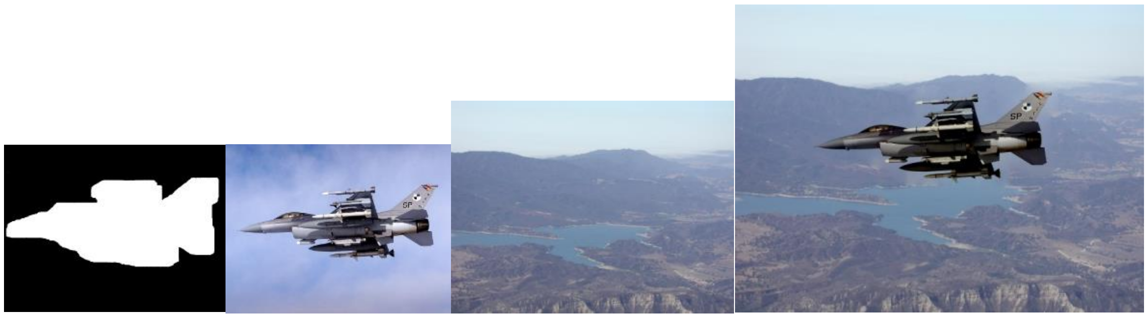


Figure 5: From left to right: Mask5, source 5, target 5, blended result.

However, I faced a problem when I was working with image 6. This problem was because a pixel in the boundary had value one in the mask. This happens because the equation (1) should be generated for every pixel which is inside the mask. Since for the pixels in the boundary, at least one of the neighbors has been missed, the function doesn’t work well. To come up with this issue, at the first line of the imblending function, I check this issue and if there is any pixel in the boundary that is included in the Mask, the output comes from a new function imblend_for_border_mask.h. In this new function, the mask, target and source would be padded by zero. Size of the padding has been considered 1. After that, Matrix x would be calculated like pervious steps. By doing so, I could get the following result from imblending target and source 6:

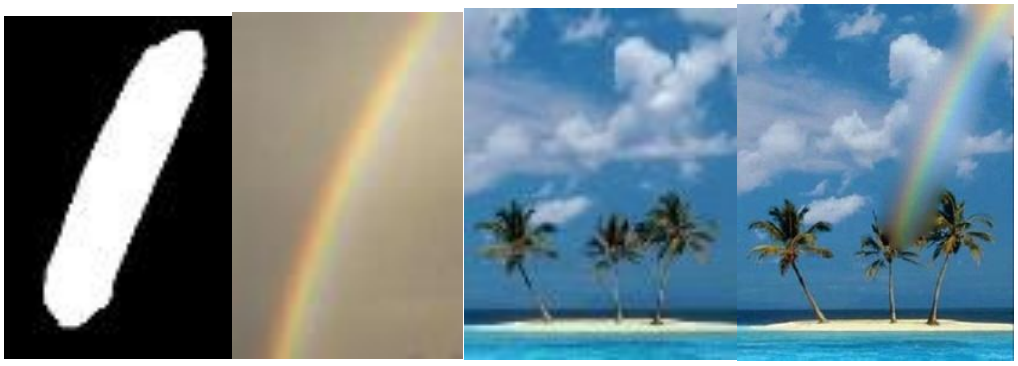


Figure 6:From left to right: Mask6, source 6, target 6 and resulting image.

From the results we can see although this method tries to bluer where source image merge with the target image but in boundaries, it doesn't perfect well. For example, we can see from image 3 that the boundaries of the merged image are obvious.

In the second part of the question, it asks us to provide three masks, sources and targets images and work with them. Here are my example:

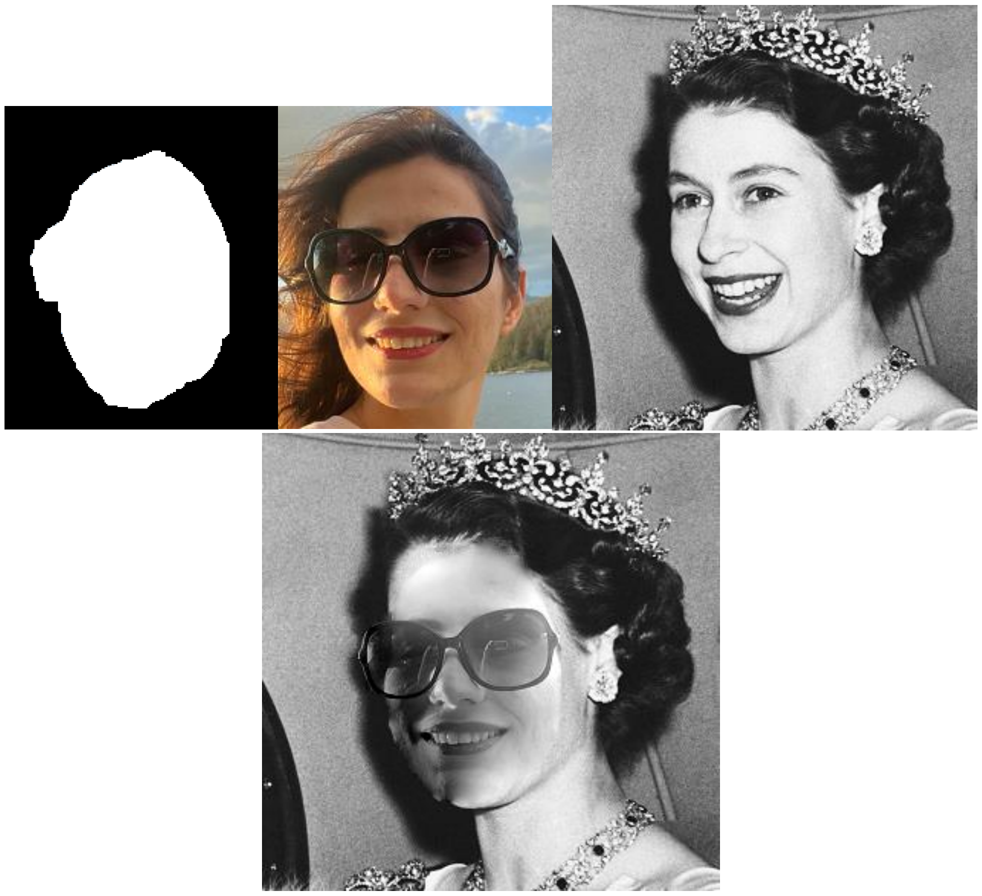
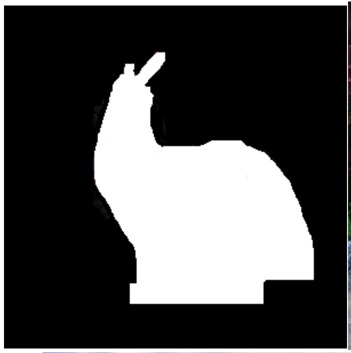


Figure 7:Top: Mask7, source 7, target 7. bottom: resulting image.

For this example, I transferred image of mine to the grey space. Also, the mask and source has been shifted and fixed using the fiximages.h function. The hard shift has been found out through paint by finding the exact pixels in the borders of face in the target image. The mask has been generated by by paint since it had more flexibility rather than provided function. Also, since the masks have been created by paint, there is a possibility that a pixel hasn't solid black or white color. Therefore, function make_sure_mask.h has been created to check this issue and fix the mask. For above example, it is possible to create a new function for only gray image to reduce computational waste but it'd decrease the generality of the function.



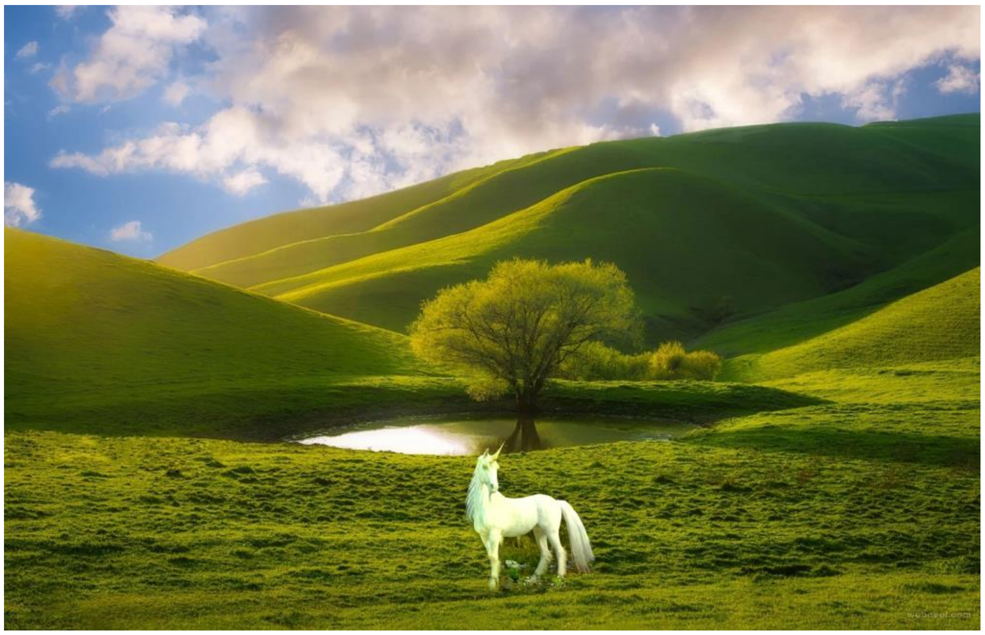


Figure 8:first row: Mask8, source 8,second row: target 8. Last row: resulting image.

Ex3:

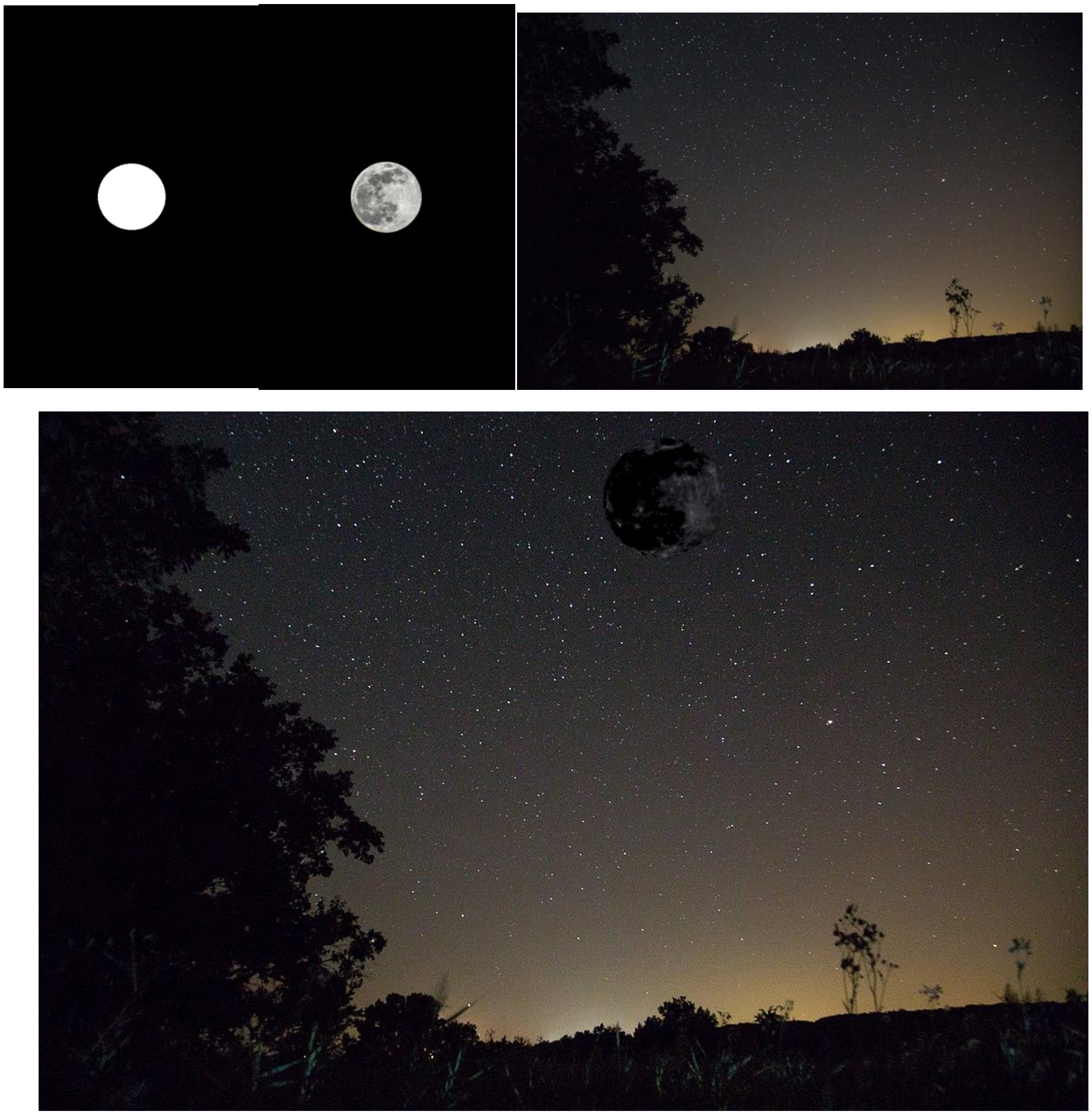


Figure 9:Top: Mask9, source 9, target 9. bottom: resulting image.

From these two last examples we can see another issue with the Poisson blending method. Here we can see from resulting images of 8 and 9 that since Poisson method tries to reduce the gradient in the merged parts, in may negatively affect the image and damage it. Here, we can see that in example 9, the bright moon has become to dark moon ☹ And also, we can see from example 2 (image 8) that our unicorn in the merged image has become green☹

To solve the moon problem, paper has mentioned gradient mixed method which would be described below:

As it mentioned in reference paper [1], the system of equations could be designed in another way in which for every pixel, the gradient in the source and target image be compared and the largest one be considered as a final gradient value of that pixel. Then matrix would be generated in this regard and system of equation could be solve. Also, another way for coming up those mentioned issue is weighted Poisson blending which we have done in our project. In this approach, we can make not a solid but also grey mask and weight source and target points in this regard. The different of this method with transparent coefficient is that the coefficient value differentiates for every pixel.

Images could be found through this link:
https://drive.google.com/drive/folders/1FMmFh08OYLka8BKX5Q0-Joln0diysj0G?usp=sharing

Reference:

1. Pérez, Patrick, et al. “Poisson Image Editing.” *ACM Transactions on Graphics*, vol. 22, no. 3, 2003, pp. 313–18, doi:10.1145/882262.882269.