Computational Photography
Homework 2 – Poisson blending

Amiya R Panda
UIN – 727006179

TASK 1 :

Use gradient-domain editing to blend a source image into a target image.

To realize it the function Poissonblend(source, mask, target) is implemented. I have also included Poissonblend1_notcompile() and Poissonblend2_notcompile() which are two other approaches of solving the problem. The details of the function is given below.

The target image is the background onto which a region of the source image is copied and blended; the region of the source image to be transferred is given by a binary mask.

Inputs:

source: The image from which pixels will be transferred onto the target.

mask: A binary matrix specifying the region of the source image that should be blended into the target.

target: The image into which the selected source region is blended; this serves as the background for blending.

Outputs:

result: An image of the same dimensions as the source/target, representing the output of the gradient domain blending.

This function assumes that the inputs, source, mask, and target, all have the same width and height. To simplify edge cases, we add a 1-pixel border around the source, target, and mask. For the source and target images, the extra 1-pixel border is created by copying pixel values along the edge, essentially extending the original images. For the mask, the extra 1-pixel border just consists of all 0s, since we do not want that border to be selected. This border is removed after blending. We reshape the source and target to have dimensions t_rows*t_cols x 3, turning each color channel into a column vector. This greatly simplifies later computations, which are performed across all color channels simultaneously.

We construct the matrix A efficiently from a set of three vectors: row_vec has entries that represent row indexes of A; col_vec has entries that represent column indexes of A, and value_vec has entries that represent the values at specific positions inside A. These three vectors are correlated, such that the final matrix A will have (row_vec(index), col_vec(index)) = value_vec(index). Thus, for each entry in A, we add one entry into each of row_vec, col_vec, and value_vec.

Each row of the sparse matrix A represents a linear equation; this variable is used to keep track of the current row inside A. The matrix A has one equation for each pixel in the target image; we iterate through them, and insert the appropriate values in the matrix A and the corresponding values in b.

If the current pixel location is not in the mask, the final value in the blended image is the same as the original value in the target image, so we insert a 1 in the matrix A, and copy the target value into the appropriate position of the vector b.

OUTPUT :

IMAGE 01 -

```
Elapsed time is 3.28808 seconds.
```

IMAGE 02 -

```
Elapsed time is 1.64071 seconds.
```

IMAGE 03 -

```
Elapsed time is 8.07813 seconds.
```

IMAGE 04 -

```
Elapsed time is 6.18928 seconds.
```

IMAGE 05 -

```
Elapsed time is 13.5735 seconds.
```

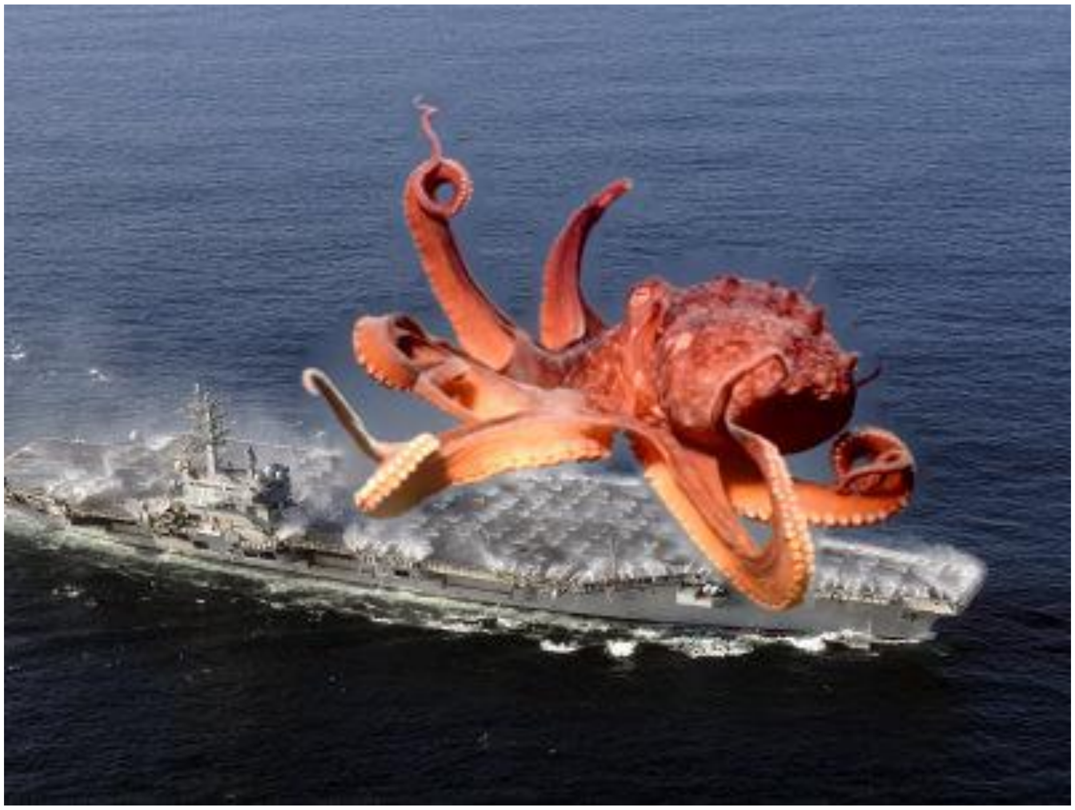
IMAGE 06 -

```
Elapsed time is 3.40078 seconds.
```

IMAGE 07 -

```
Elapsed time is 3.24148 seconds.
```

This method does not scale to image 6 and image 7. For that, mix gradient method is required.



SP