Name: Rizu Jain
UIN: 430000753

Date: March 2, 2020

CSCE 689 Computational Photography
Report on Assignment 3

I. Overview

The goal of the project was to perform image blending give a source image and its mask and a target image. This was achieved using two approaches: Pyramid blending and Poisson blending. As a part of graduate credit, mixing gradients was also implemented.

II. Deliverables

The contents of the submission have the following:

Code\	Directory containing the python source files
Report_430000753_RizuJain.pdf	This report.

III. Setup

The assignment was developed in the following environment:

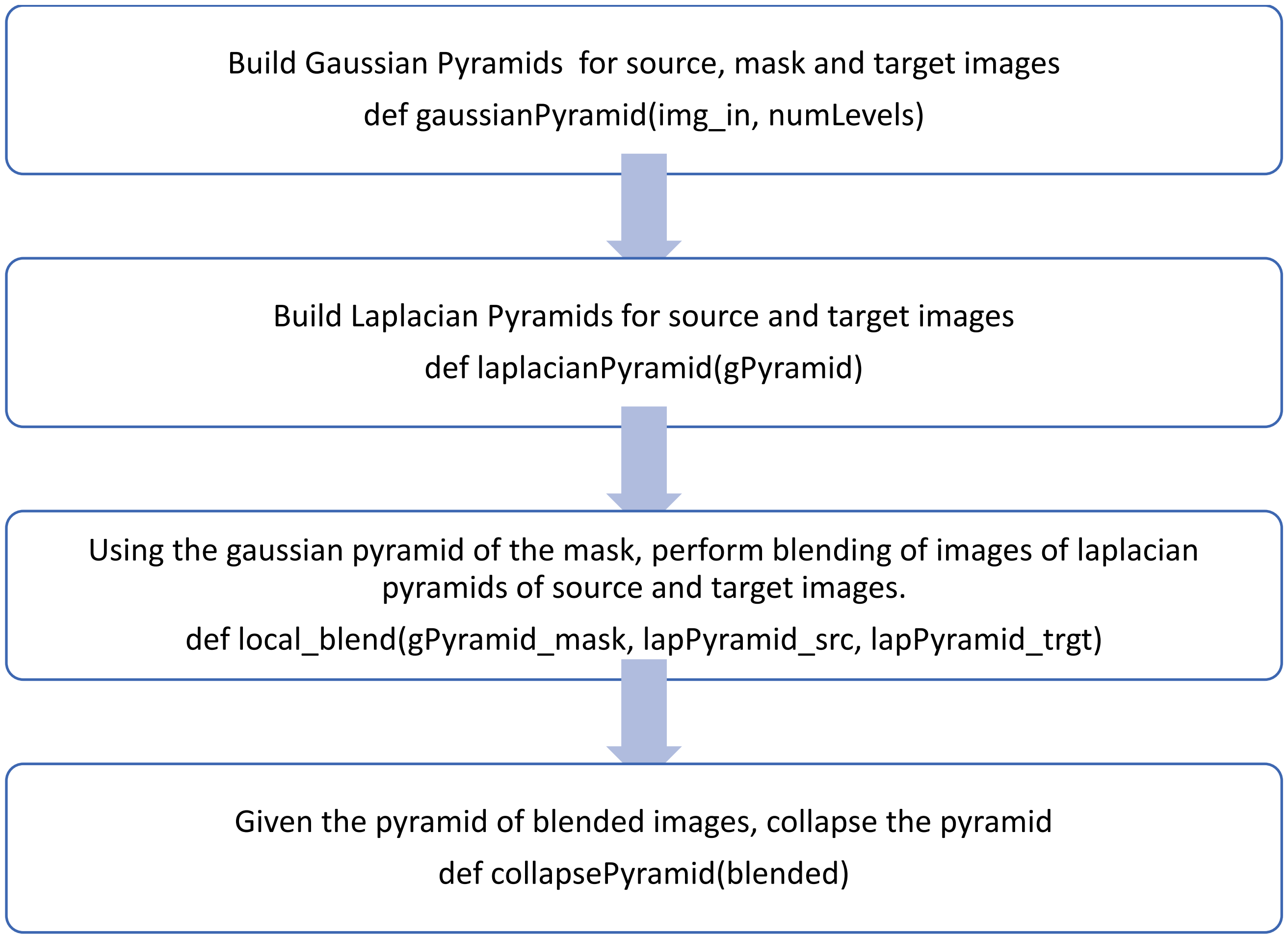
- Host OS: Windows 10
- IDE: Spyder (Python 3.7)

IV. Task 1: Pyramid Blending

This task is implemented in the function:

```
def PyramidBlend(source, mask, target)
```

The overall process and the functions that implement the subtasks are briefed in the following flow diagram:



Build Gaussian Pyramids for source, mask and target images
def gaussianPyramid(img_in, numLevels)
Build Laplacian Pyramids for source and target images
def laplacianPyramid(gPyramid)
Using the gaussian pyramid of the mask, perform blending of images of laplacian pyramids of source and target images.
def local_blend(gPyramid_mask, lapPyramid_src, lapPyramid_trgt)
Given the pyramid of blended images, collapse the pyramid
def collapsePyramid(blended)

To reduce the size of the image to build the gaussian pyramid, the following function is implemented:

```
def myDownsample(img_in)
```

To expand the size of the image to build Laplacian pyramids and at the end collapse the images in the pyramid, the following function is implemented:

```
def myUpsample(img_in)
```

Both the above functions deploy the cv2.pyrDown() / cv2.pyrUp() for the resizing and subsequent filtering and sampling to build the pyramid.

The results for this task are given in the results section.

Boundary Cases:

The case where upon up sampling in Laplace pyramid and reconstruction in blending, the image of next layer may get bigger than the current is handled by cropping the expanded image.

V. Task 2: Poisson Blending

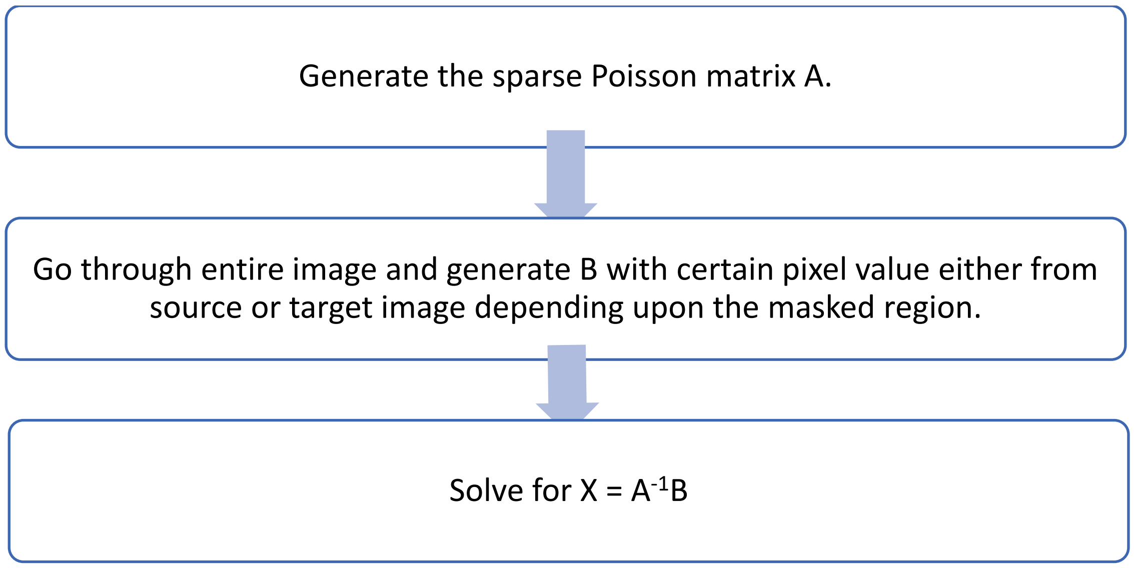
The task here is basically to also determine which pixels from source or target in the masked region should blend into the final target image. Rather than naively copying paste the overall intensity of the pixel, we computer gradient of the source and target image and finally the blended pixel in the target image corresponds to the gradient of the pixel in the source image for the masked region.

Our objective is to solve the corresponding AX = B equation. Here the matrix A represents unknown gradient values and B is represents the desired output gradient and values.

Please note that:

- **Matrix A is only generated once, whereas vector B is generated for each channel.**
- **Sparse matrices are used for efficient computing.**

The steps can be summarized as below:


Generate the sparse Poisson matrix A.
Go through entire image and generate B with certain pixel value either from source or target image depending upon the masked region.
Solve for $X = A^{-1}B$

This is implemented in the function:

```
def PoissonBlend(source_orig, mask, target_orig, isMix)
```

The results for this task are given in the results section.

VI. Extra Credits: Mixing Gradients

Mixing gradient finds its application when the target image has significant gradient on the areas where the source image is to be blended (in the masked region).

In this case, rather than simply using the source gradient for pixel values in the masked region on the target, a modified gradient is used accounting for the larger target gradient in case.

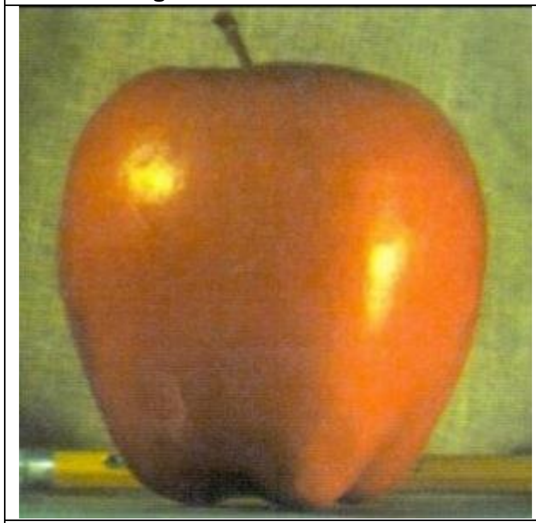
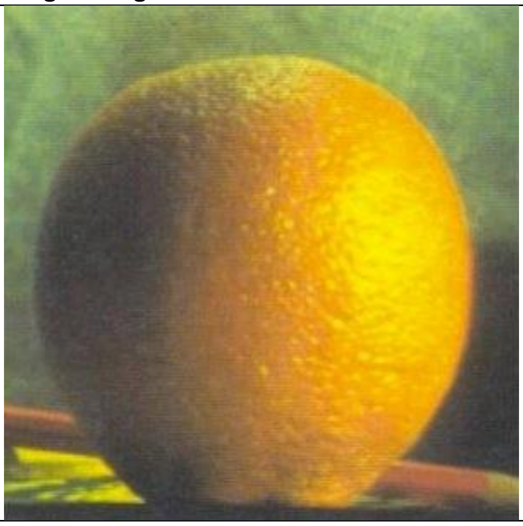
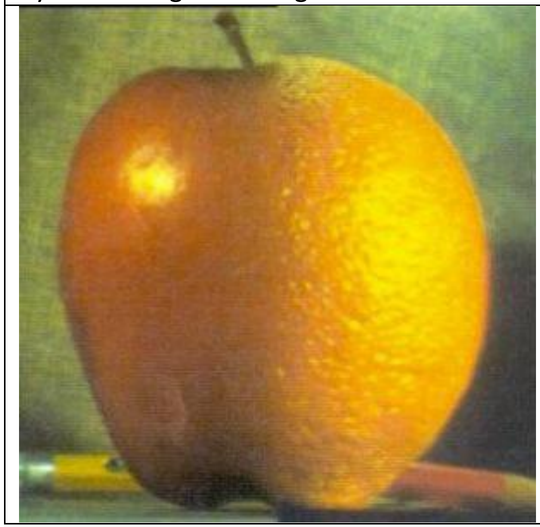
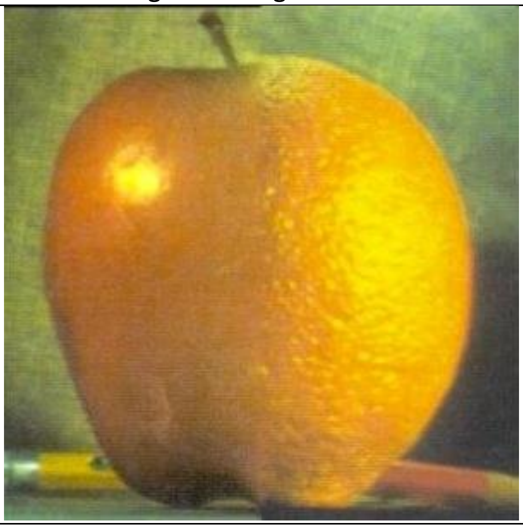
In the code this gets enabled by passing isMix = True (default is False) in the main driver section:

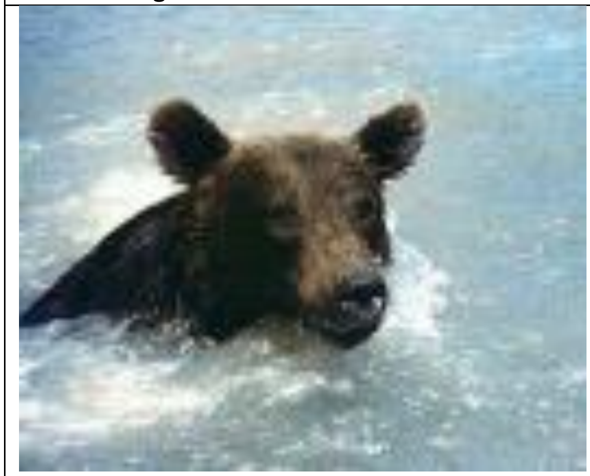
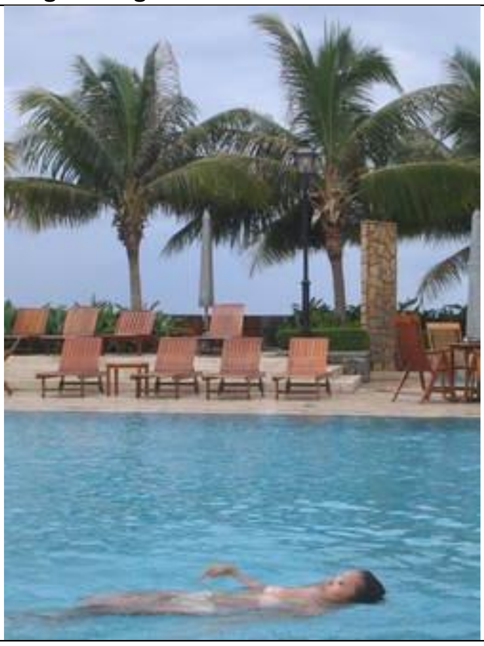
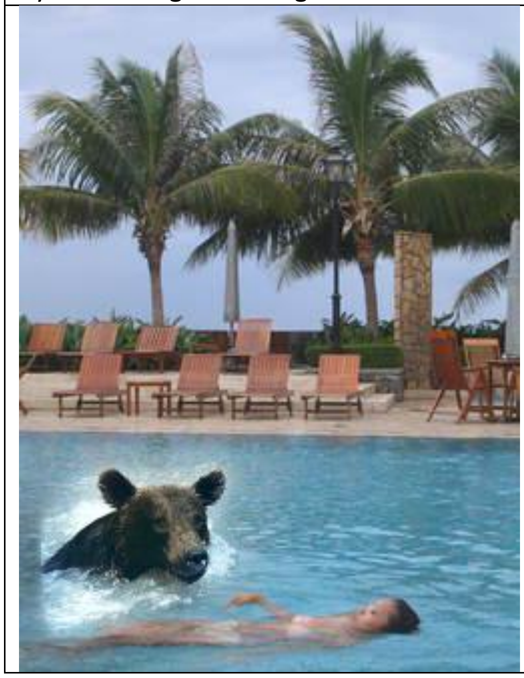
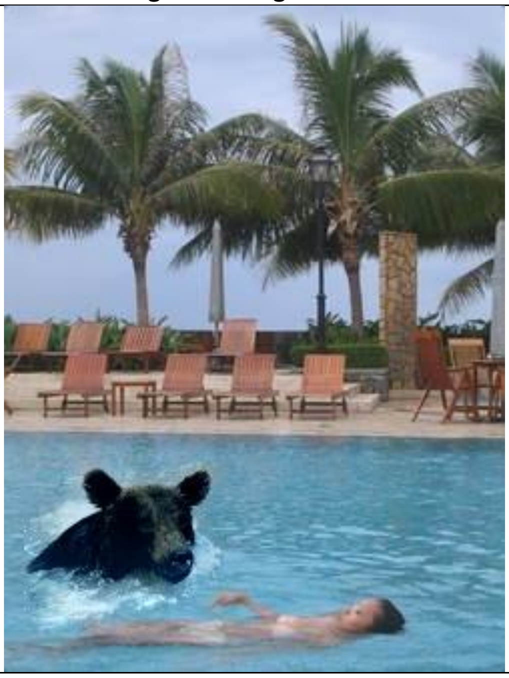
```
    # False for source gradient, true for mixing gradients
    isMix = False
```

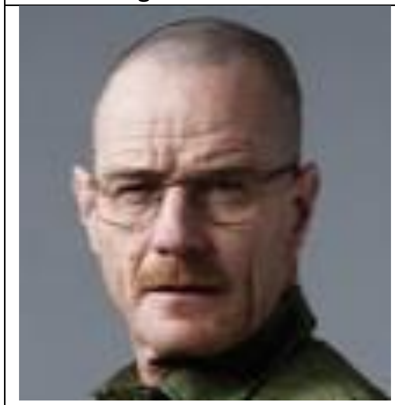
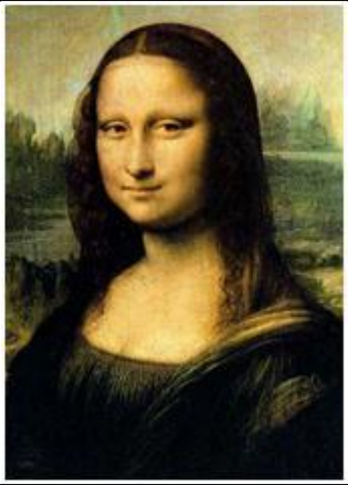
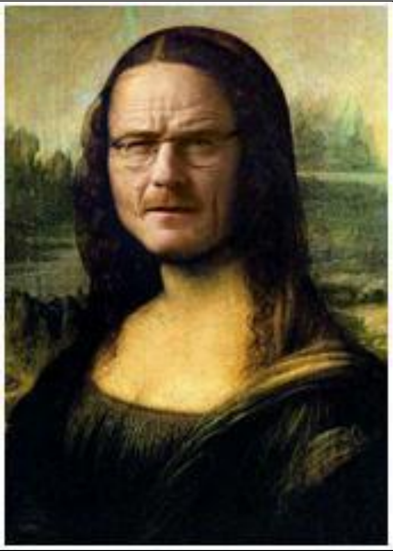
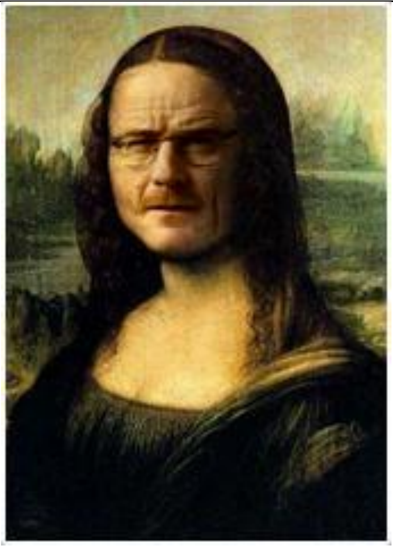
The results for this task are given in the results section.

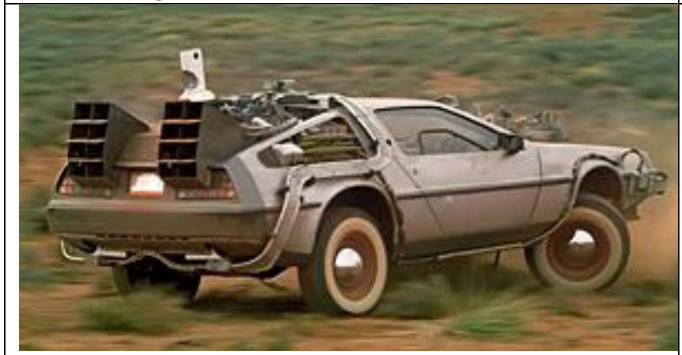
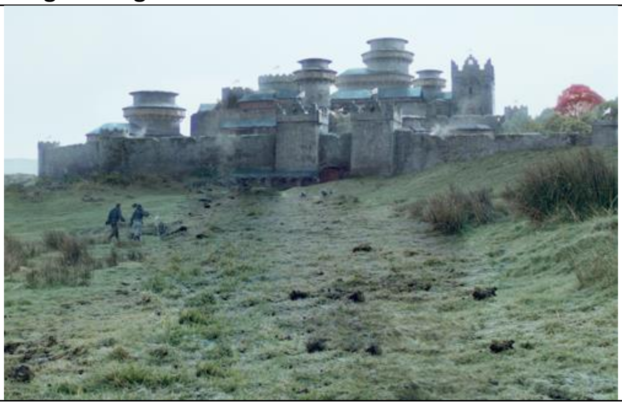
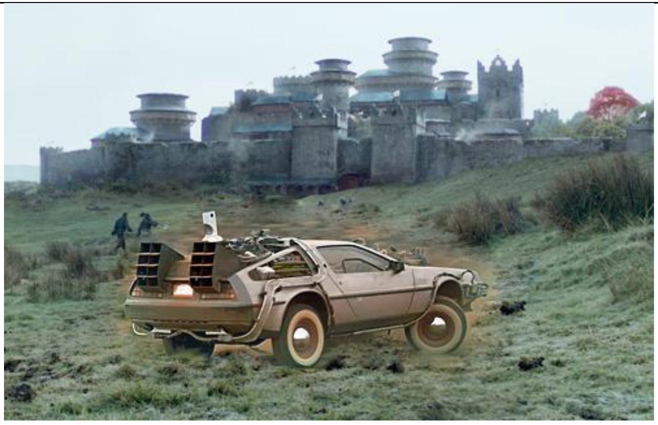
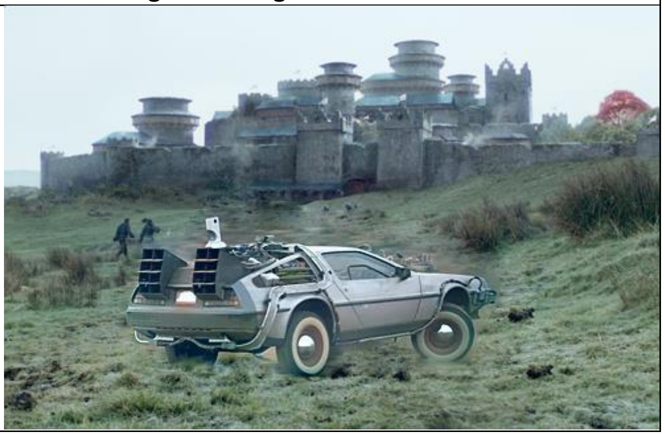
VII. Observations & Results

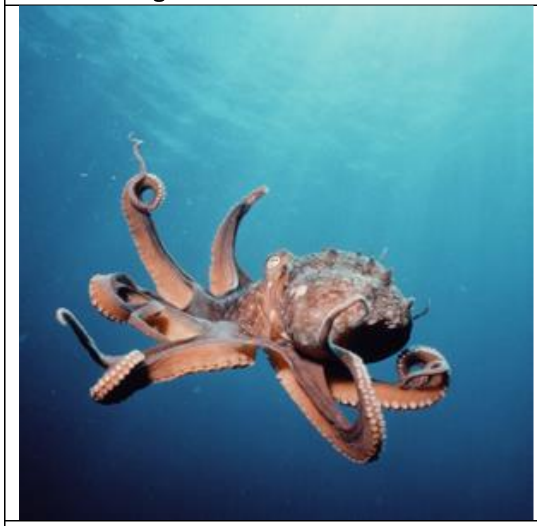
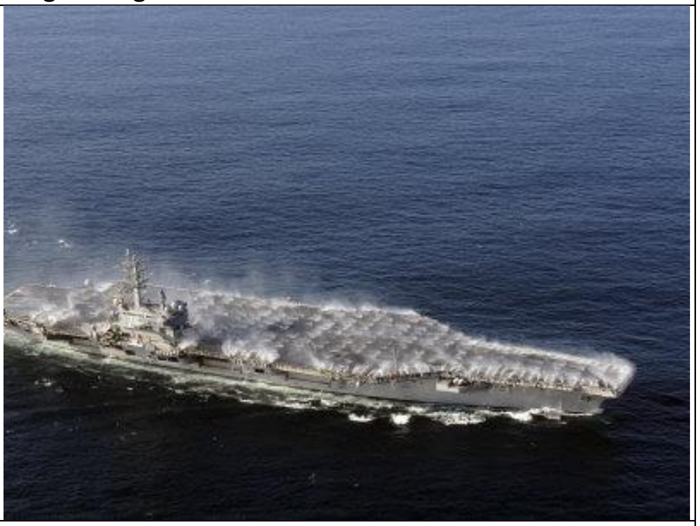
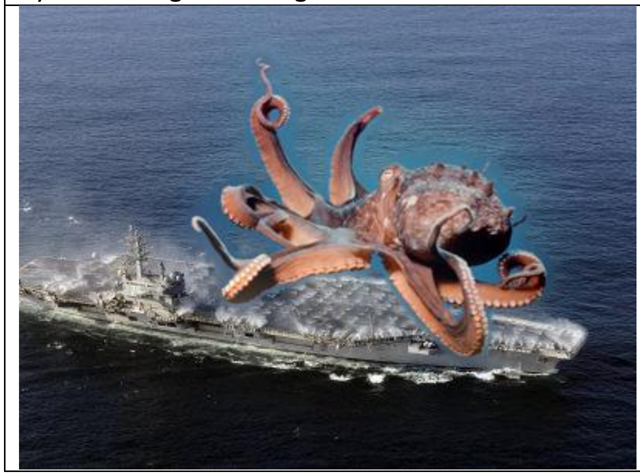
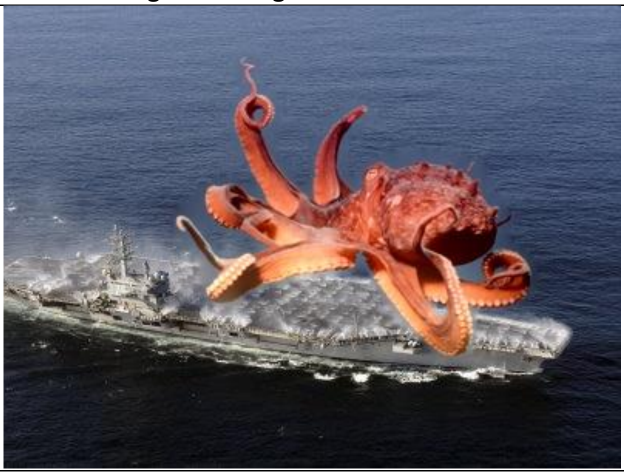
The following images depict the given source images, the target images and the results obtained in Task 1 (i.e. Pyramid blending of images) and Task 2(i.e. Poisson blending of images.)

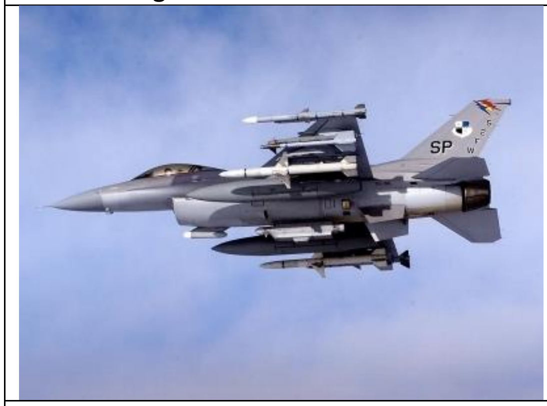
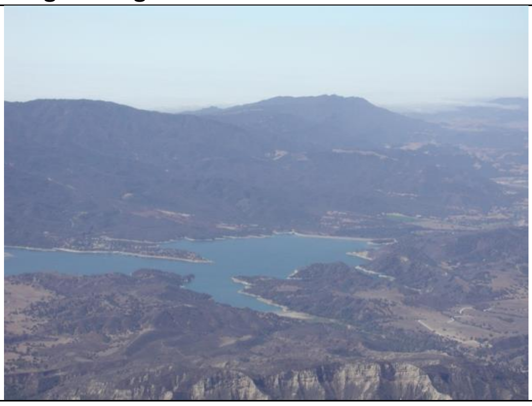
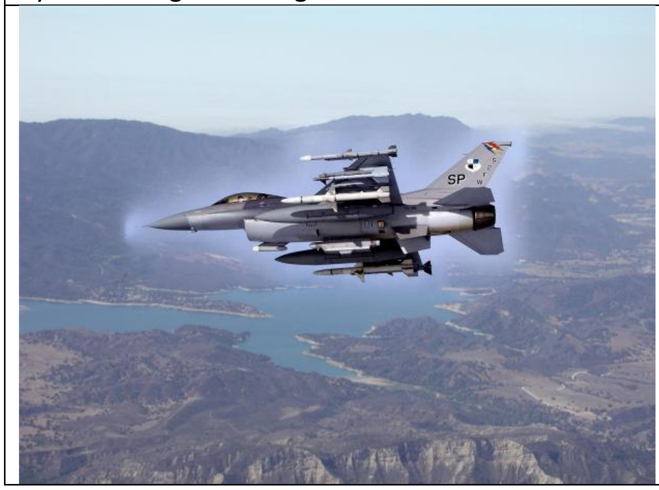
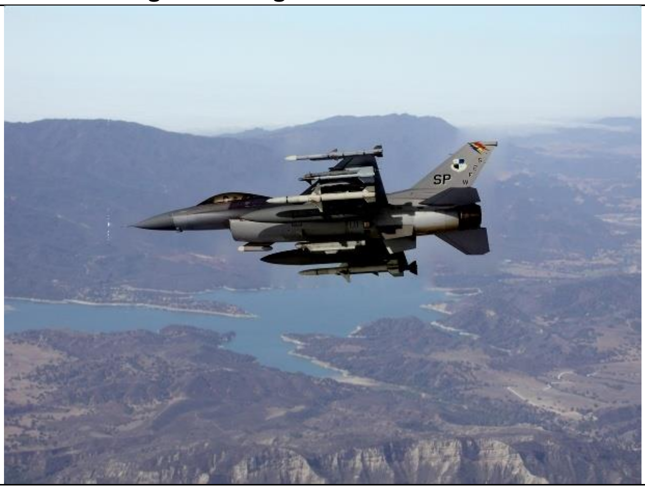
Source Image:	Target Image:
	
Pyramid Image Blending:	Poisson Image Blending
	

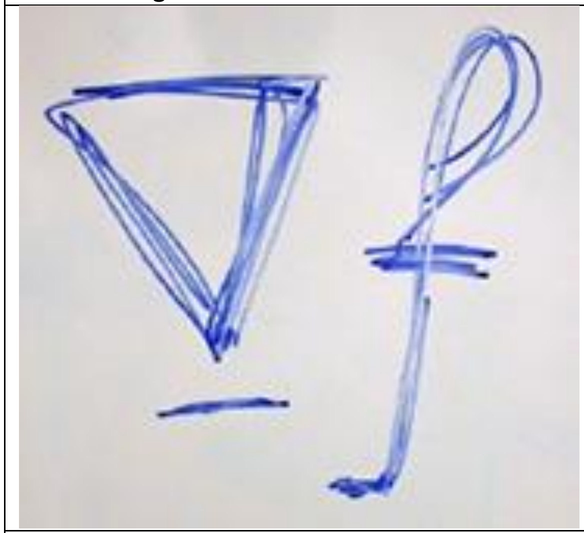
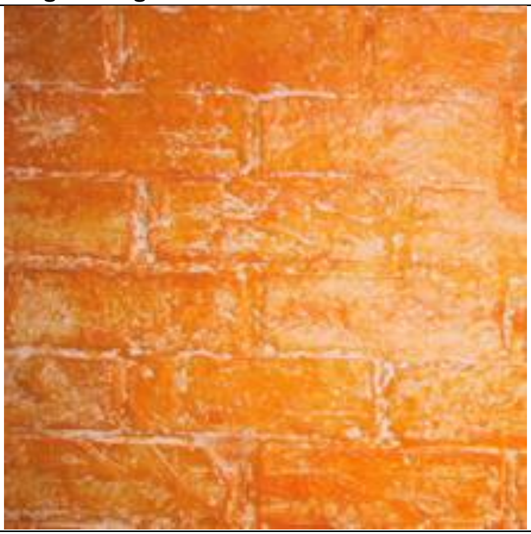
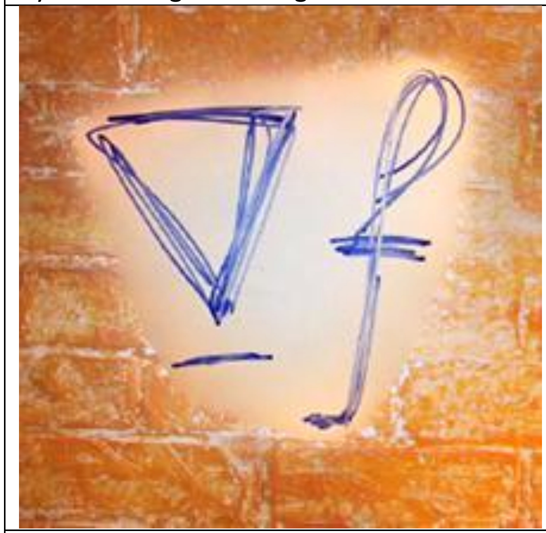
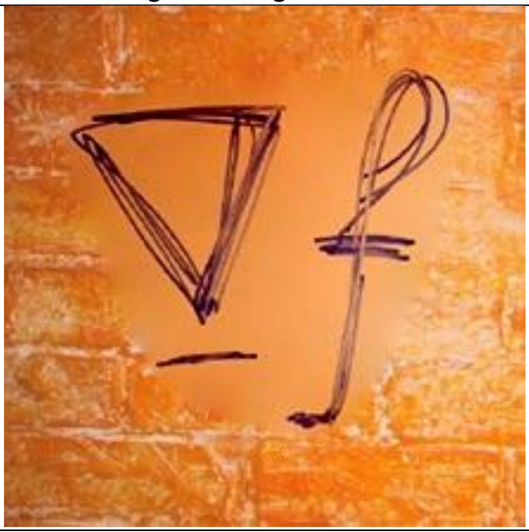
Source Image:	Target Image:
	
Pyramid Image Blending:	Poisson Image Blending
	

Surce Image:	Target Image:
	
Pyramid Image Blending:	Poisson Image Blending
	

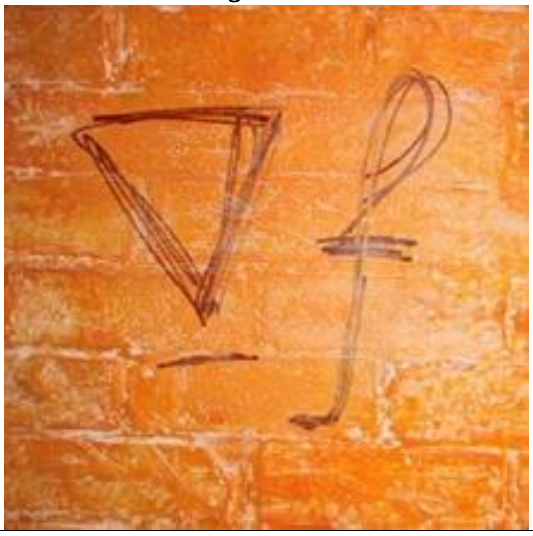
Source Image:	Target Image:
	
Pyramid Image Blending:	Poisson Image Blending
	

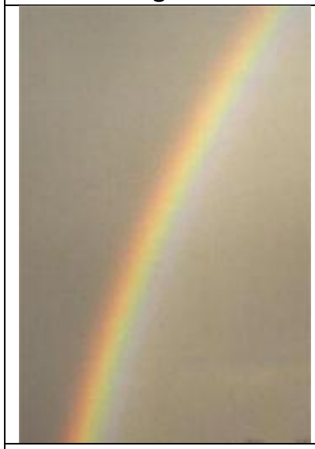
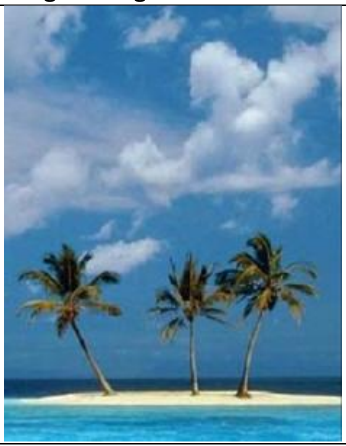
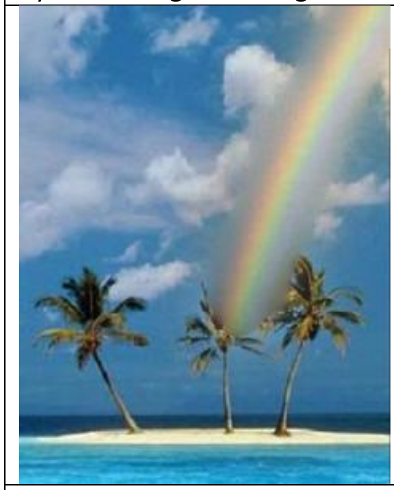
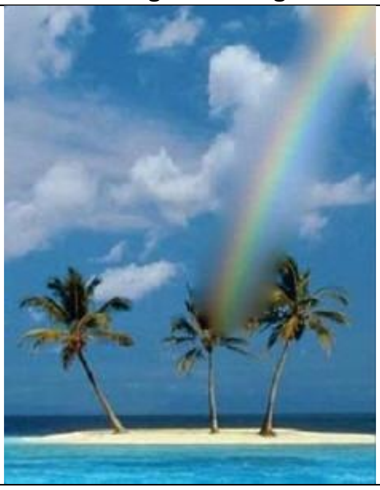
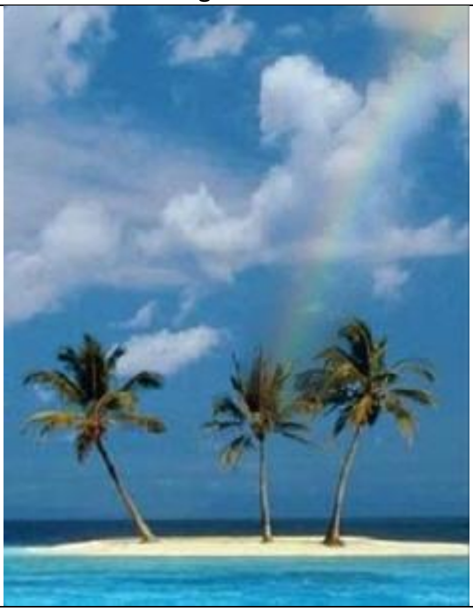
Source Image:	Target Image:
	
Pyramid Image Blending:	Poisson Image Blending
	

Source Image:	Target Image:
	
Pyramid Image Blending:	Poisson Image Blending
	

Source Image:	Target Image:
	
Pyramid Image Blending:	Poisson Image Blending:
	

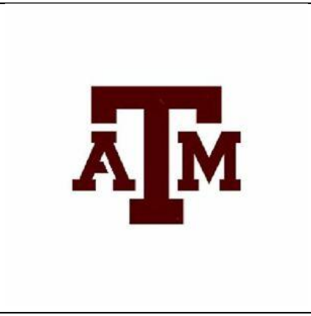
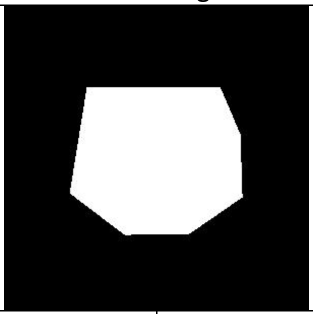
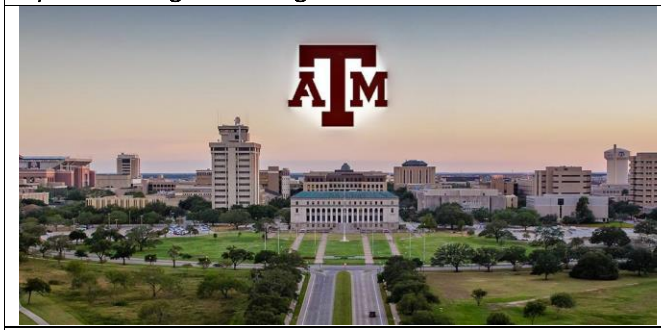
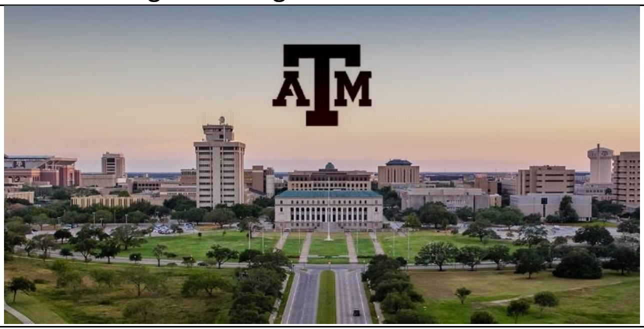
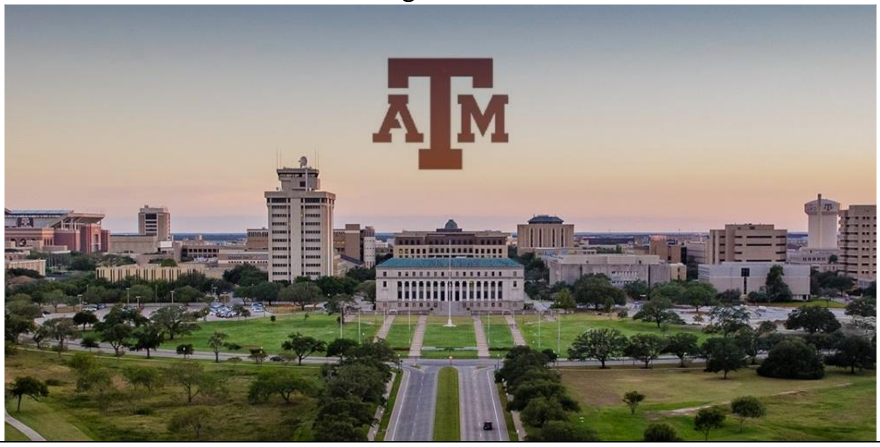
Mixing Gradients:



Source Image:	Target Image:
	
Pyramid Image Blending:	Poisson Image Blending
	
Mixing Gradients:	
	

External Example

The following example is taken externally. The mask image generated using getMask() is also shown:

Source Image:	Target Image:
	
Mask Image:	
	
Pyramid Image Blending:	Poisson Image Blending
	
Mixing Gradients:	
	

VIII. References

1. https://en.wikipedia.org/wiki/Discrete_Poisson_equation
2. Patrick Pérez, Michel Gangnet, and Andrew Blake. 2003. Poisson image editing. ACM Trans. Graph. 22, 3 (July 2003), 313–318. DOI:https://doi.org/10.1145/882262.882269

~ End of Report ~