

MO446 – Introduction to Computer Vision

Project 1

Breno Leite
Guilherme Leite

29/08/2017

Input Images

Throughout this project some images are used as input to test the algorithms. Figure 1 was used as input for the pyramids exercises **2.1**, **2.2**, **2.3** and **3.1**, its dimensions are 400x300 and it is a colored image.



Figure 1: Input image for pyramids and Fourier transform exercises. (**p1-1-0**)

Important note: The borders seen in the figures are not part of the image, they are figurative information about the starting and ending points of the image. Moreover, all the image scales in this report were changed in order to make the text more readable.

Images in Figure 2 are used for the blending exercises (**2.4** and **3.2**), their dimensions is 540x392. Note that the images are slightly rotated, this will affect some results which comes from the image and not by any error on the process itself.

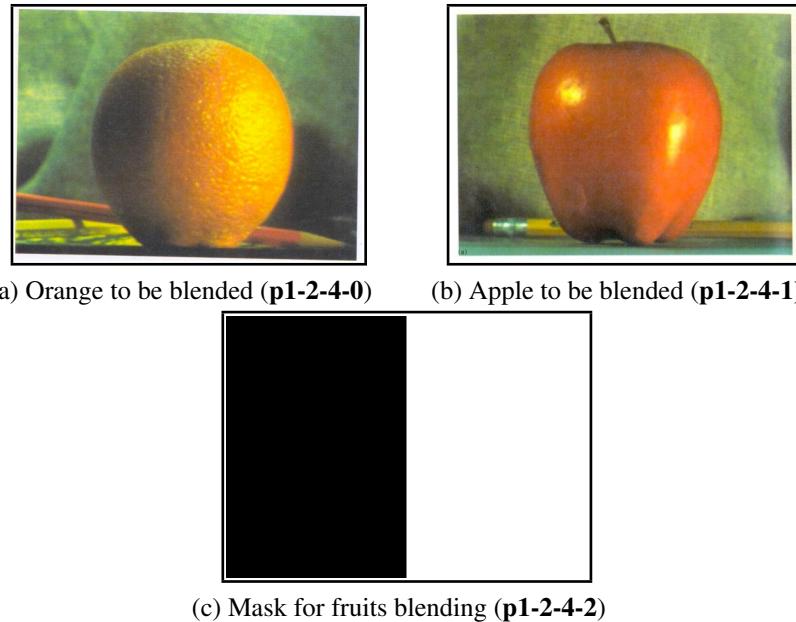


Figure 2: Initial images used on the blending experiments.

For more testing on the blending exercise **2.4** the images in Figure 3 were also used. We decided to use them as well because they seemed complexer than the first one.

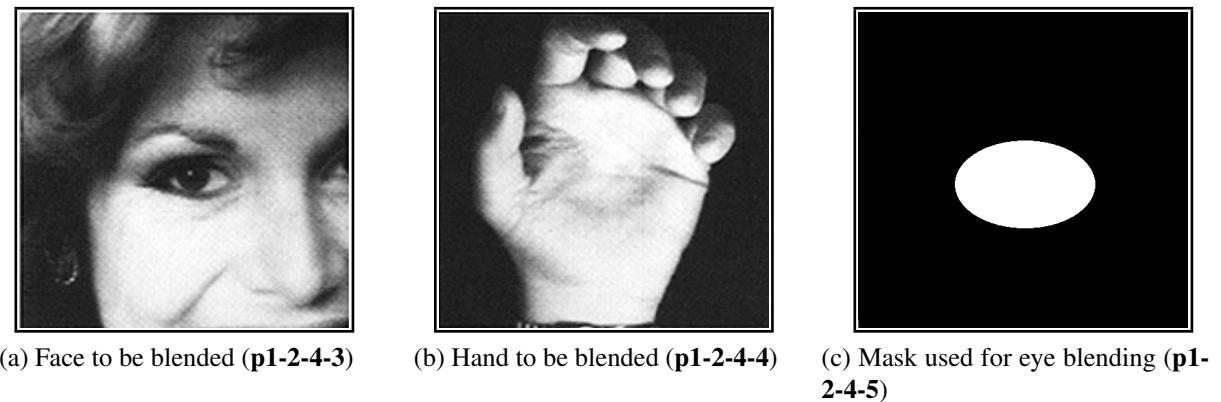


Figure 3: Complementary images used for further testing the blending operations.

Question 2 - Spatial blending

2.1) Convolution can be understood as the action of applying a kernel to an image, in this case every pixel in the image is affected by the weighted sum of the filter applied to the pixel neighborhood. This approach requires some special attention when dealing with the edges of the image, since the application of the kernel will overshoot the image borders. Our solution to this problem was to extend the original image filling its new borders with zeros.

A minor downside of this approach is the lessen effect of the kernel around the edges, this solution was chosen for its simplicity to implement. Additionally, the convolution was tested with three different Gaussians smoothing kernels, with sizes 3x3, 7x7 and 15x15. In comparison to the embedded solution by OpenCV our convolution was noticeably slower as seen in Table 1. Our intuition about this is that the OpenCV is running C code, which is highly optimized.

		Time (seconds)		
Kernel Size		3x3	7x7	15x15
Convolution	Implemented	4.481	4.535	5.083
	OpenCV	0.001	0.004	0.010

Table 1: Comparison between our implementation and OpenCV convolution time.

As expected the kernel smooth the edges around the image and suffers with a darkening close to the borders as seeing in Figure 4a, with bigger kernel the image suffered more distortion to the point of loosing all of its fine details (Figure 4c). The loss of these details is accounted by the size of the neighborhood that affected each pixel, thus a larger kernel smooths more than a smaller one.



Figure 4: Convolution results applying different Gaussian kernels.

2.2) To store the Gaussian and Laplacian pyramids on this project we used a linked list, in which each node holds a level of the pyramid. Since the data structure is a list the access method is as in a array, e.g. `pyramid[i]` returns the image in the *i*th pyramid level, disregarding the necessity to implement an access function.

The strategy chosen to implement the interpolation was the bilinear interpolation. These decisions were taken regarding the time to implement and simplicity to understand. It is also worth noting that due to some confusion about the meaning of Up and Down, these functions were renamed as follow: Expand referring to the action of expanding the image width and height, and Contract analogously. Figure 5 shows the pyramid formed with the implemented function.

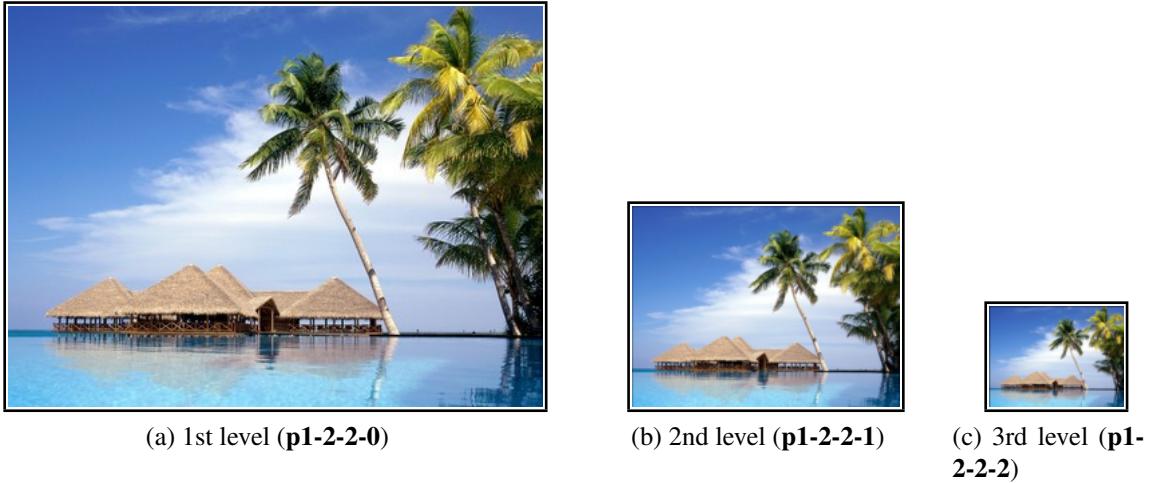


Figure 5: Images composing the levels on the Gaussian Pyramid formed from image **p1-1-0**

2.3) The Laplacian pyramid demanded more attention to its details, like the fact that the last level of the pyramid is a copy of the same level at the Gaussian pyramid. This particular level is used to perform the reconstruction of the original image, first the image is extended and interpolated to match the previous level size. And then, it is summed with the previous level of the pyramid. The repetition of these steps results on the original image that was compressed in the pyramid, Figure 6 shows the Laplacian pyramid.

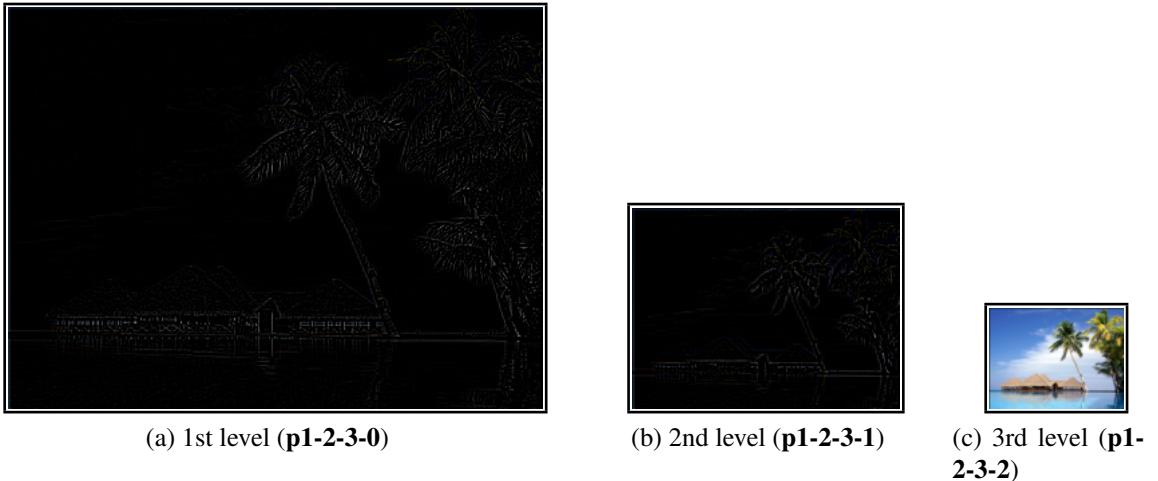
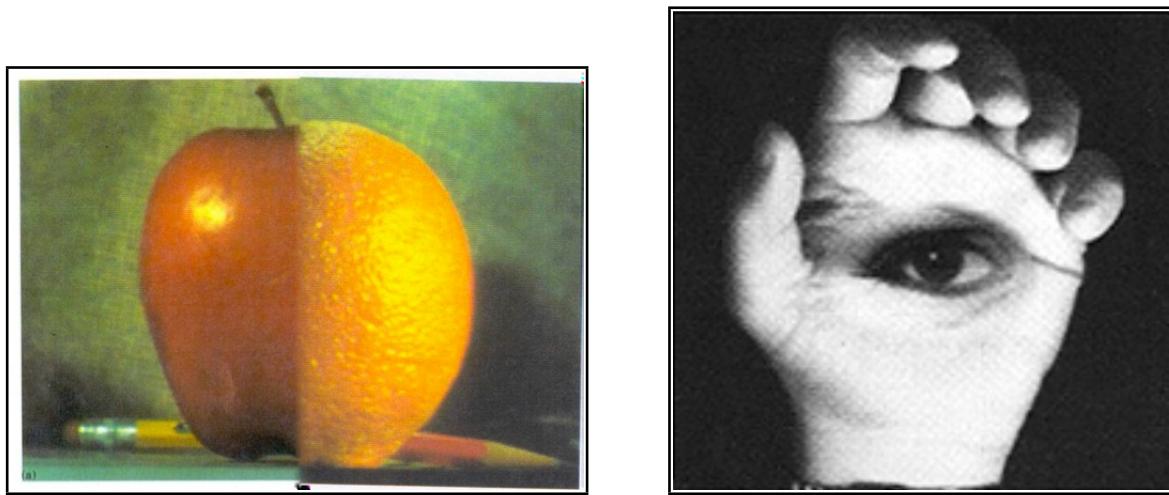


Figure 6: Images (a), (b) and (c) composing the levels on Laplacian Pyramid formed from image **p1-1-0**

2.4) The process of blending two images implemented here is referred as “Splining Regions of Arbitrary Shape” in Burt and Adelson’s paper, it consists of three Gaussian pyramids GA , GB and GM , and three Laplacian pyramids LA , LB and LS , in which LS is the resulting pyramid and the image at its base is the blended image.

The algorithm to blend the images uses both LA , LB and GM pyramids simultaneously, from the top of the pyramid it multiplies each level image with the equivalent level mask. Then, sum the resulting images with each other and expand the resulting one, repeat these steps and the resulting image at the base of LS will be the blended image. The core point of this process is the Gaussian filter applied to the mask that smooths the borders, another key point is the expand operation per-

formed in the pyramid. Two tests were performed reproducing the original results in the paper, the results are shown in Figure 7.



(a) Blending using images in Figure 2 (**p1-2-4-0**)

(b) Blending using images in Figure 3 (**p1-2-4-1**)

Figure 7: Images after the process of blending using the Burt and Adelson method.

As seen, the results are pretty similar with the reported into the paper. We can see how the process of smoothing performed by the pyramids gave a better looking to the edges of the combined images. In Question 3.2 we will see how important this process is to the blending operation.

Question 3 - Frequency Blending

In this question, we will present some experiments developed using *numpy* and *OpenCV* functions in order to transform the image from spatial domain to frequency domain. The process will be divided into two different experiments for better readability.

On the first one (**3.1**), we will be modifying the phase and magnitude in order to see the impact on the image reconstructed on the Fourier transform. On the second experiment (**3.2**), we will be blending the two images used in Question **4.2** in order to compare frequency and spatial blending.

3.1) Exploring Fourier Space

To explore the Fourier space, we implemented two different functions. The first one is responsible to transform an image into vectors of magnitude and phase, while the other is responsible to reconstruct the image from the magnitude and phase vectors. The Figure 8 shows the magnitude and phase obtained transforming the Figure 1 to the frequency domain.

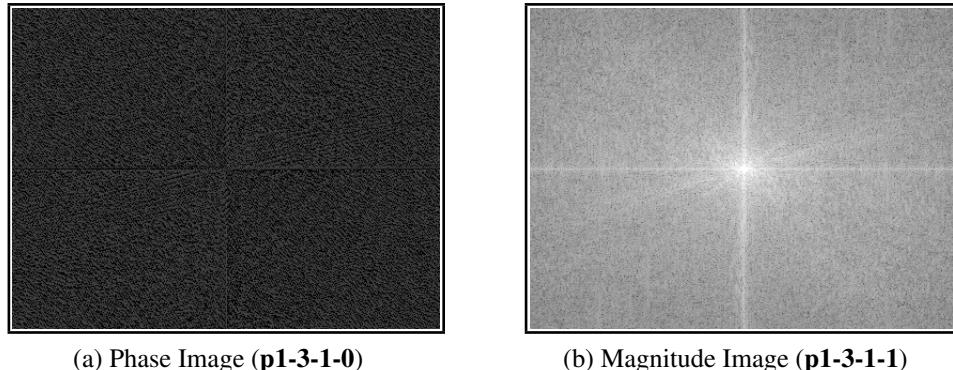


Figure 8: Images representing phase and magnitude on the Fourier transform using image **p0-1-0**.

In order to represent these images, the results from the Fourier transform were shifted to the center and also reduced by the functions, $20 * \log_e(\text{magnitude})$ and $40 * \log_e(\text{phase})$, in which *magnitude* and *phase* are vectors from the transformation. The reduction is necessary to keep the numbers in a range from 0 to 255. This normalization does not affect the transformation, the values are converted back before the reverse Fourier transform.

The Figure 8a represents the phase, it is quite hard to obtain any information from this representation. Moreover, the Figure 8b represents the magnitude of the image, brightness on the magnitude image represents high frequencies on the original image. In this way, the magnitude indicates that image **p1-1-0** has more high than low frequencies, the intuition comes from the high brightness on the magnitude image.

In the next experiments, we will be modifying the values of the phase and magnitude before the reverse Fourier transform, this process should reconstruct the original image when using the same phase and image. In order to modify them, we used a percentage of the pixels from the magnitude or phase images, the rest of the pixels were zeroed out. In the experiments this process uses just 1 pixel or a percentage of pixels of 25%, 50%, 75% and 100% to be blended.

This pixels used might be in two different ways, highest or lowest, e.g. 25% lowest pixels on

phase image would maintain the values and the rest would be zeroed out. As expected, all the images reconstruct using 100% of the pixels in magnitude and phase were perfectly reconstruct (Figure 9). The following experiments were made using the Figure 1 in grayscale, the grayscale was used for more simplicity and further on one experiment will show the same results in a colored image to prove algorithm capacities.



Figure 9: Reconstructed image from inverse Fourier transform using 100% of pixels in magnitude and phase. (**p1-3-1-6, p1-3-1-11, p1-3-1-16, and p1-3-1-21**)

In order to show the differences when modifying the phase or magnitude image, we will divide the experiments into two parts, also for better readability.

Phase - First we will be modifying the phase image and performing the inverse Fourier transform. In this experiment we were expecting changes on the position of the objects. The Figure 10 show the results when using the lowest values of the phase image.

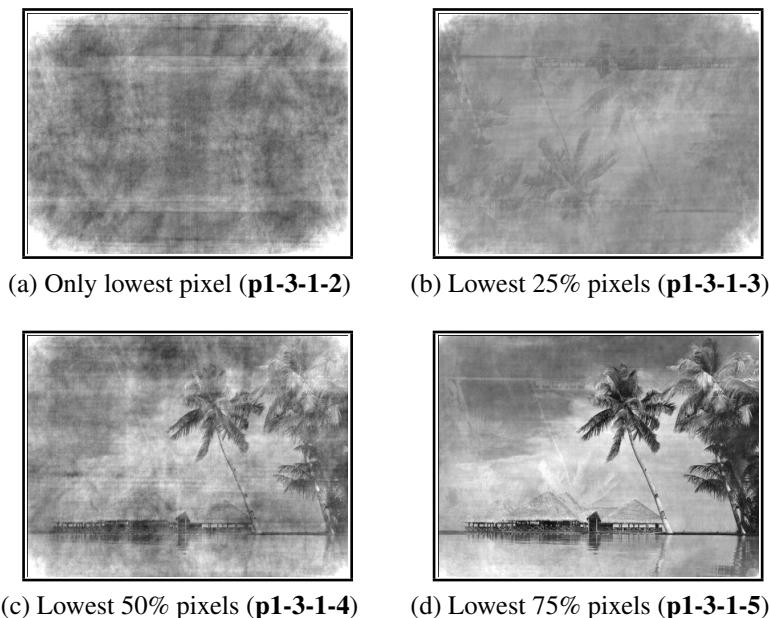


Figure 10: Results obtained by inverse Fourier transform using lowest pixels on phase image.

As we can see the image shows us information about orientations, in this way, selecting dif-

ferent values changes the orientation of the image. Another interesting fact are the upside down elements on the reconstructed images, our intuition is that it happens because of the range in which the transform is working (0 to 380 degrees). In other hand, the Figure 11 shows the results on the reverse transform using the highest pixels on phase image.

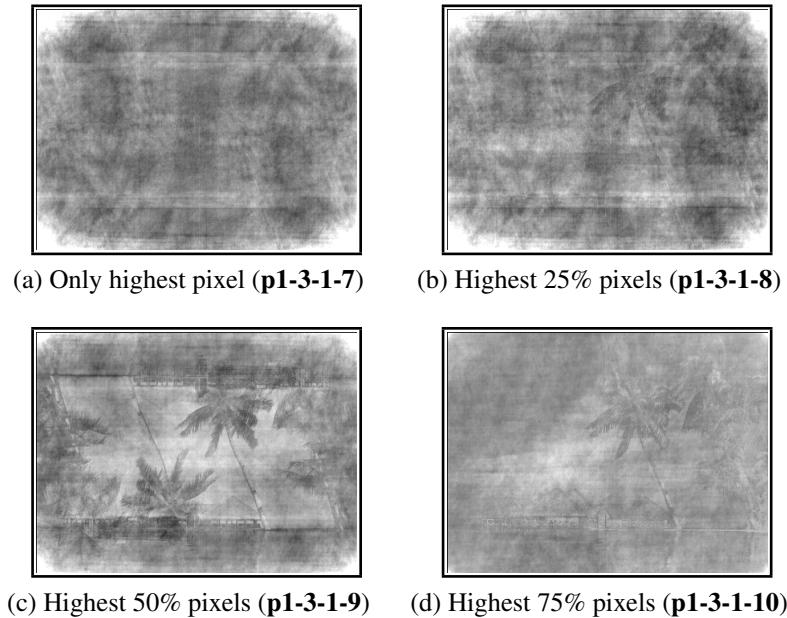


Figure 11: Results obtained by inverse Fourier transform using highest pixels on phase image.

An interesting feature is that the images contain complementary information, as example the Figure 11b has less information than Figure 10d about the original image, however their information is complementary. The same effect might be seen in the other images as well.

Magnitude - In this part we will be doing similar experiments, but this time, modifying the magnitude image. In addition, we expect to see different results on the intensity on the reconstructed images, which is described by the magnitude on the frequency domain. The Figure 12 shows the different images obtained on the inverse Fourier transform using the lowest pixels from magnitude image.

Extracting the lowest pixels from magnitude is similar to use only low frequencies of the image, because of this the first row of Figure 12 is almost all black pixels. Proving our intuition looking at the magnitude brightness, as the original image has more high frequencies using low pixels does not describe a good portion of the image.

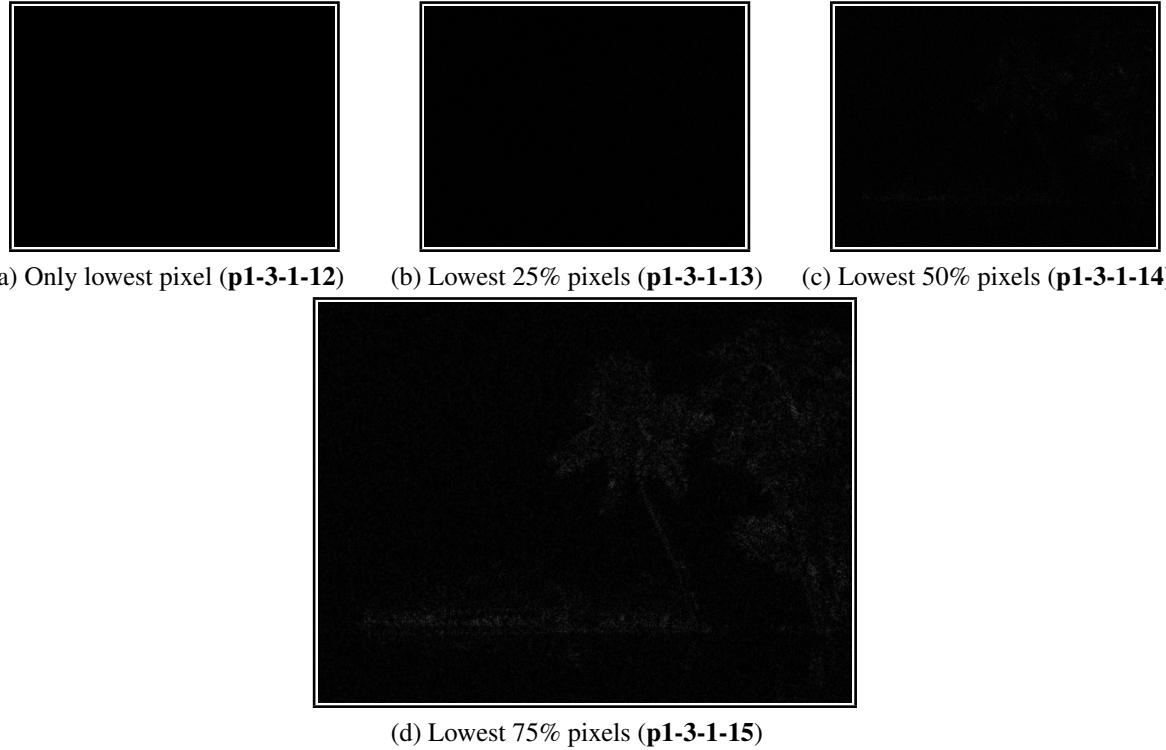


Figure 12: Results obtained by inverse Fourier transform using the lowest pixels on magnitude image.

However, we can see some of the contours being marked in Figure 12d. This effect produces a image similar to the convolved image on Figure 6a, which is a High-Pass-Filter. So, this process could be used as a high-pass filtering in the frequency domain. In contrast, The Figure 13 show the same process using highest pixels on the magnitude image.

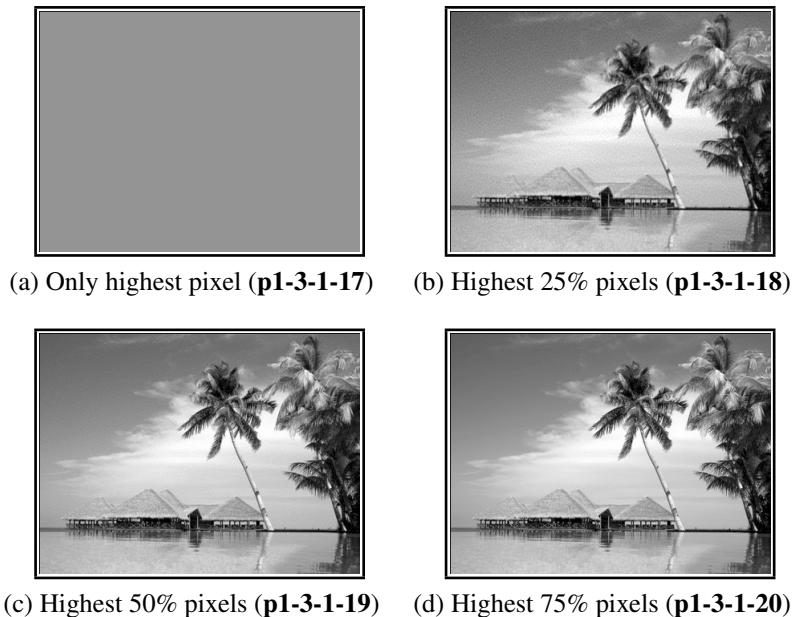


Figure 13: Results obtained by inverse Fourier transform using the highest pixels on magnitude image.

As we would expect, even with less pixels from magnitude (25%) the image was better repre-

sented, which indicates that a small part of the highest pixels on magnitude have most part of the information about the original image. We can note that the quality of the reconstructed image is proportional with the percentage used on the magnitude image. The first image (Figure 14a) does not have enough information to reconstruct the image, but the color already informs the predominant color intensity on original image (with just the highest pixel).

In order to show that our implementation could also deal with colored images, we used the same experiment as before with a colored image. The Figure 14 shows the result of using the highest values on the magnitude image on this colored image.

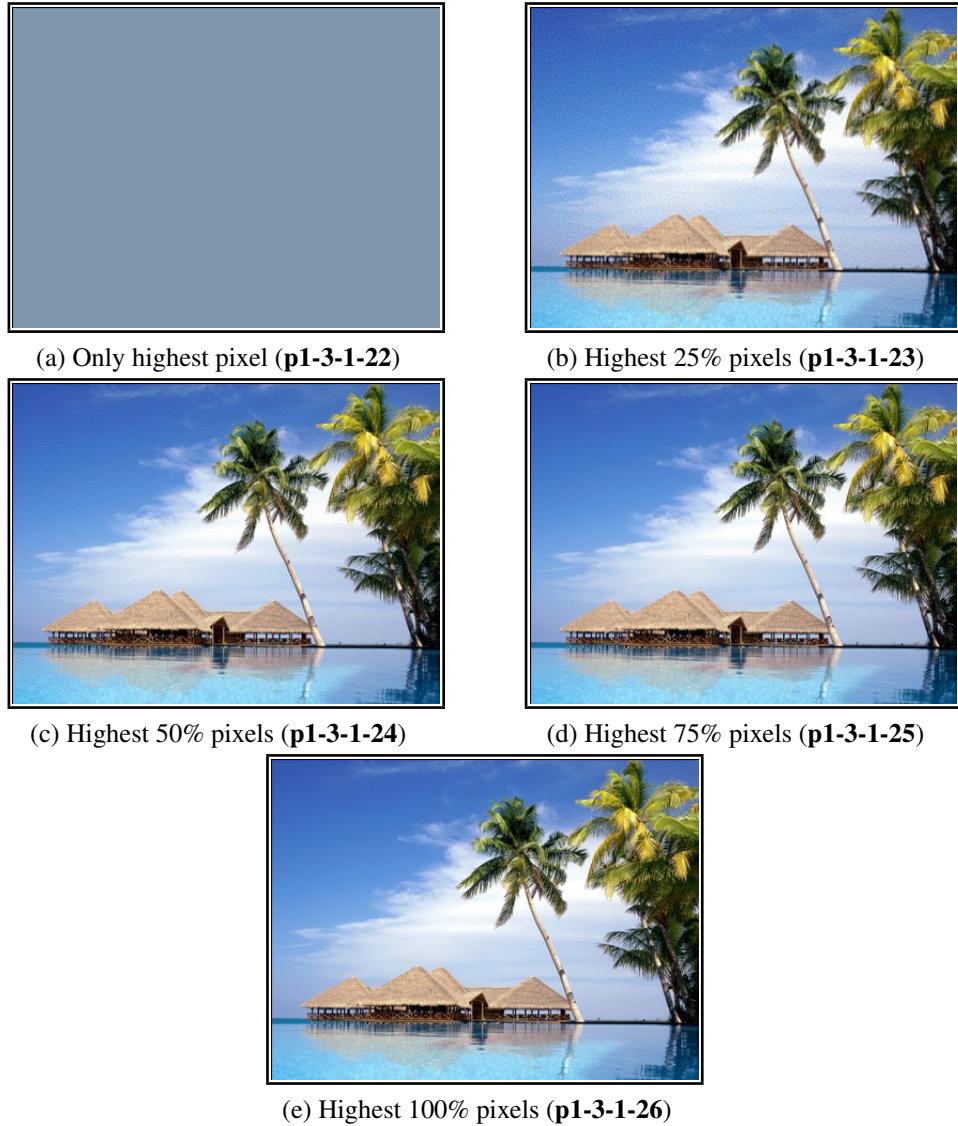


Figure 14: Results obtained by inverse Fourier transform using the highest pixels on magnitude image from the colored image.

In order to use a colored image, the Fourier transform was applied in each color channel. Note that the Figure 14e is the perfect reconstruction of the image **p1-1-0**, which indicates the process of Fourier transform was successful.

3.2) Blending

In order to obtain the blending on the frequency domain, we first transformed both images from spatial to frequency domain using the Fourier transform, which gives us the vectors magnitude and phase. Our first approach was to sum these vectors together, this approach obviously did not work, and the result can be seen in Figure 15.

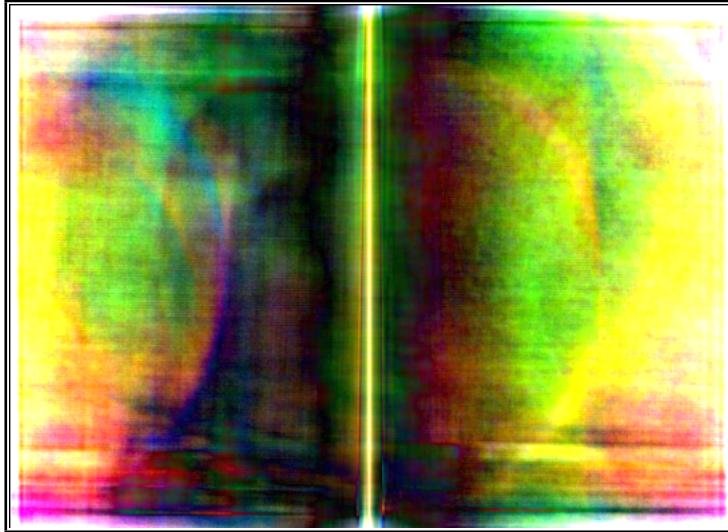


Figure 15: Blending summing phase and magnitude vectors.

After this approach, we have tried to implement the blending using different values for the phase and magnitude images, as example, we would use 50% of highest values on both images and then perform the sum. At this time, we were expecting that the sum of the zeros (generated by the 50% lowest values) could work with the summation. This approach did not work out, and the result is shown in Figure 16.

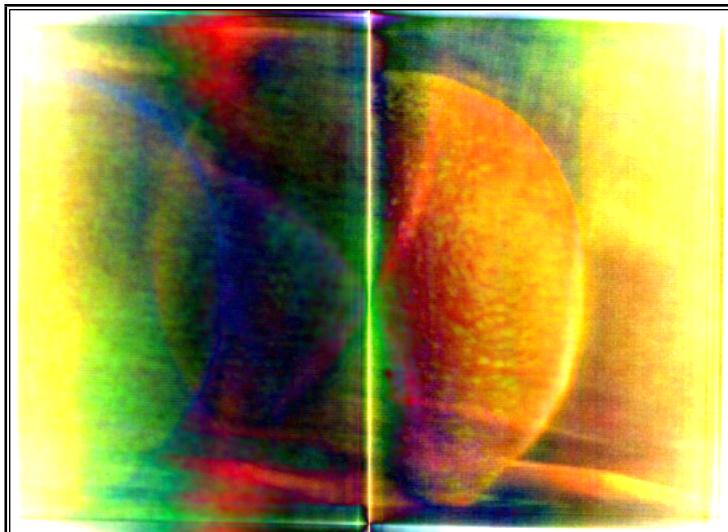


Figure 16: Blending summing phase and magnitude vectors using the 50% highest pixels from magnitude and phase images.

After these tries, we started thinking about our results. Our conclusion was that no merging

between the phase and magnitude would work as a blending. Moreover, we concluded that our approach was not working because of the way we were summing the vectors. We were dealing with complex numbers, and complex numbers do not add as normal numbers.

So in order to fix this issue, we first construct a *numpy* complex function $magnitude * np.exp(1j * phase)$. This way we could create one function to each image, and then sum them. Doing in this order *numpy* knows that it is dealing with complex numbers, and it performs the right operation to sum te complex numbers. Which was what we were looking for, the result of this process is shown in Figure 17.

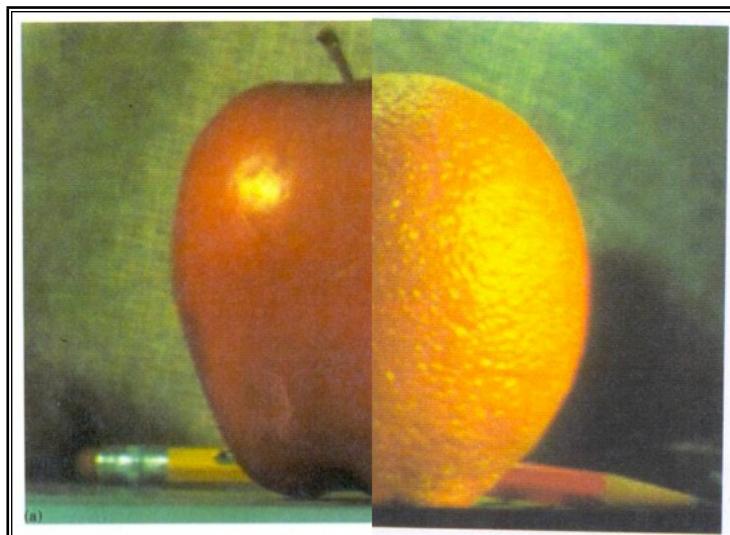


Figure 17: Blended image using *numpy* complex sum. **p1-3-2-0**

The generated image is a blend of two images, but when compared to the blending on spatial domain (Figure 7a) the results are not good. There is a high contrast between the images, we have tried some filtering approaches in order to solve it. However, none of the filtering ideas worked.

So, analyzing the result we figured that a pyramid could solve our problem (as it does on spatial domain). However, we had problems with the up-sampling and down-sampling. We could not change pixels arbitrary, mainly because one pixel on frequency domain represents more information on the original image, as seen in exercise **3.1 Exploring Fourier Space**. We did not have enough time to explore more this blending, but our intuition is that it might be possible to use pyramids with some “trick” applied to the frequency domain.