Image Processing - Exercise 3

Shachar Cohen | 206532418

Introduction

The goal of this exercise is to explore the usages of Gaussian and Laplacian pyramids in image processing, and more specifically in combining two different images into one image.

The Gaussian pyramid is a popular image processing technique used to create a multi-scale representation of an image. It is a type of image pyramid that is formed by creating multiple layers of the same image, each layer with a different level of detail. The Gaussian pyramid is named after the Gaussian function, which is used to generate the different layers of the pyramid: the first layer contains the original image, then the following layers are created by blurring the previous layer with a gaussian kernel and sampling every second pixel. The blurred image at deeper levels of the pyramid contains mainly lower frequencies from the original image, while the shallower levels contain more detail, i.e., higher frequencies.

A Laplacian pyramid is another popular multi-scale representation of an image, where each layer is created by the following formula. Denote $\mathrm{G_k}$ as k’th level of the gaussian pyramid of an image, and N as the depth of the gaussian pyramid. then the k’th layer of the Laplacian pyramid is calculated by:

$$L_k = \begin{cases} k < N: G_k - \text{enlarge}(G_{k+1}) \\ \quad k = N: G_k \end{cases}$$

Notice that when scaled to the same size, the Laplacian layers sum to the original image (telescopic sum):

$$\sum_{k=1}^{N} L_k = \left[\sum_{k=1}^{N-1} \text{scaled}(G_k) - \text{scaled}(G_{k+1})\right] + \text{scaled}(G_N) =$$

$$\text{scaled}(G_1) + \sum_{k=2}^{N-1} \text{scaled}(G_k) - \sum_{k=2}^{N-1} \text{scaled}(G_k) - \text{scaled}(G_N) + \text{scaled}(G_N) =$$

$$\text{scaled}(G_1) = \text{original image}$$

This property makes Laplacian pyramids especially useful in blending images together, as shown later in this report.

Algorithms

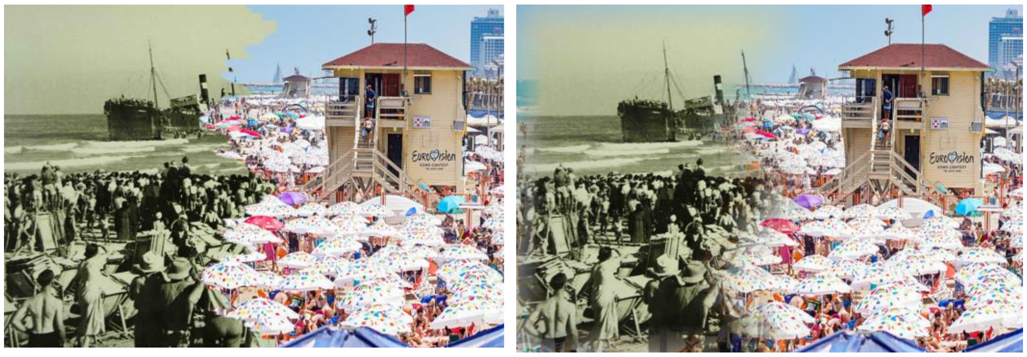
Algorithm 1: Blending
Input: Two images A and B and a mask M (i.e. an image with values 0 and 1) – all of the same size, blurring factor T (i.e., how many times to apply a blurring kernel over the mask). Apply a blurring gaussian kernel of size 5*5 over the mask T times. Create two Laplacian pyramids LA and LB for each of the images. Create a gaussian pyramid for the mask GM. Create the layers of the blurring pyramid: Layer k = Lak * GMk + (1-GMk) * Lbk Sum the layers and return the result.

Algorithm 2: Hybridization
Input: Two images A and B (with the same size) and some threshold t between zero and 1. Create two Laplacian pyramids La and Lb. Compute T = round(number of layer in La * t). Create the layers of the hybrid pyramid: Layer k = Lak if k <= T otherwise Lab. Return the sum of the layers.

I implemented both algorithms independently, using the pyramid data structure I created (elaborated in the Algorithms section). The main concept I used in both algorithms is that the Laplacian of an image sums up to the original image.

Additionally, since higher levels a Laplacian correspond to the difference between increasingly blurred representation of the original image, they represent the lower frequencies in the image.

In each of the algorithms I added a hyperparameter that affects the "smoothness" of the result. In the blending algorithm, blurring the mask makes the transition between the images smoother:

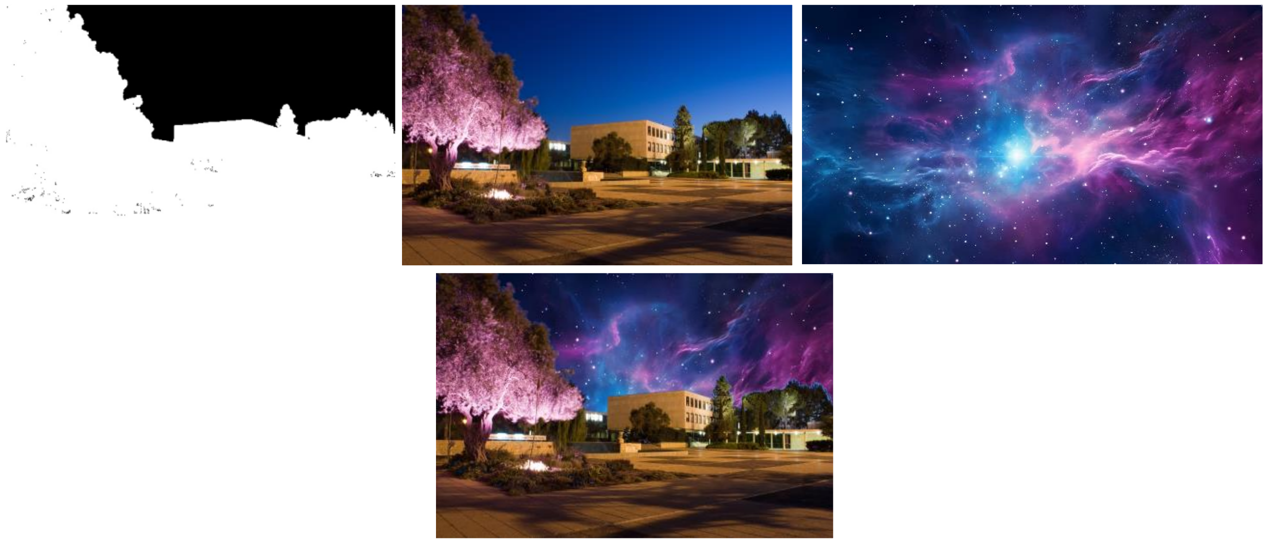

An Aliya ship arriving at Tel Aviv shore in 1939 compared to modern day Tel Aviv. The blurring effect is demonstrated on the right.

On the hybridization algorithm, the threshold hyper parameter affects "how much" of the first image we want to depict in respect to the other image. In practice this parameter hasn't affected much on the final result.

Challenges: one of the challenges I faced in the implementation was dealing with image sizes that are not powers of two. Therefore, before manipulating the images I first resized the second image to the first image shape, and then padded both images with zeros so their new heights and widths are powers of 2.

Results

Blending results:



Hybridization:

Failed results:
In the first image I tried to incorporate features of one person into the face of another person, which didn’t work well because the transition wasn’t smooth enough. For this reason, I added a blurring hyper parameter to the algorithm. In the second image you can see the blended image is distorted, since I didn’t convert the image matrices’ type to int16 (which means the Laplacian values are always positive).

Pyramids

To simplify the usage of pyramids in this exercise, I developed an abstract data structure named "Pyramid". This structure stores multiple layers of an image and provides the following functionality: getting a layer at a certain depth (in its original size or enlarged to a specific size), and summing the layers (after enlarging all the layers to the first layer’s size). Each class that inherits from Pyramid must implement the layers generation method. Then, I created four sub-classes of this abstract class: Gaussian, Laplacian, Blend and Hybrid.

The Gaussian and Laplacian pyramids: These data structures simply implement the algorithms stated in the introduction.

Algorithm 3: Pad to Powers of Two
Input: An image. Output: A padded image with dimensions that are powers of two. Get the width and height of the image. Set the new height and new weight to be 2 ** ceil(log2(x)) with x the original size respectively. Pad the original image with zeros symmetrically to fit the new size, and return the padded image.

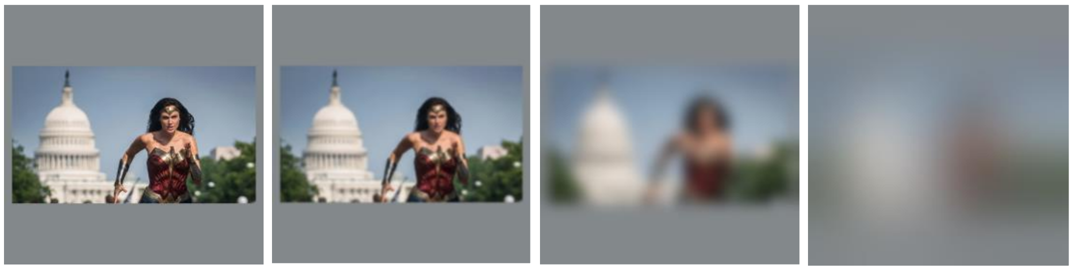
Algorithm 4: Shrink
Input: An image. Output: A down-sampled version of the image. Blur the image with a gaussian kernel of size 5*5 and sample every second pixel (i.e., remove odd rows and columns). The kernel: $\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$

Algorithm 5: Enlarge
Input: An image. Output: A up-sampled version of the image. Add a zero column and a zero row between every two columns and rows. Blur the image with a gaussian kernel of size 5*5 and multiply the image by 4 (since the average of the original image is 4 times the average of the enlarged image) .

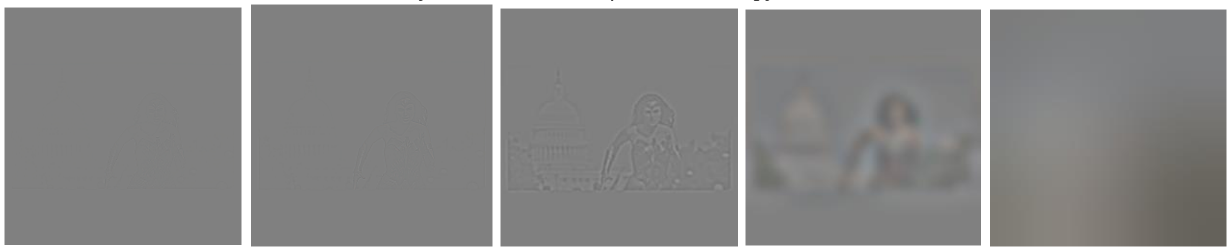
```
Algorithm 6: Gaussian Pyramid Layers Generation
Input: A padded image.
Set N the number of layers to be floor( log2(min(weight, height)) )
Initialize an empty array of layers.
Add a copy of the original image to the array.
For each index k from 1 to N-1:
        Shrink the last layer in the layers array and add the result to the array.
```

```
Algorithm 7: Laplacian Pyramid Layers Generation
Input: A padded image.
Set N the number of layers to be floor( log2(min(weight, height)) )
Get the Gaussian pyramid of the image. Denote Gk as the k'th level of the Gaussian pyramid.
Initialize an empty array of layers.
For each index k from 0 to N-2:
        Add Gk – enlarge(Gk+1) to the array.
Add $G_{N-1}$ to the array.
```

Though I implemented my own versions of enlarging and shrinking, I used OpenCV's versions of these algorithms (as they are slightly more efficient). For the rest of the algorithms, I used my own implementation. One of the key steps during processing is converting the image array type from uint8 to int16 to allows both negative and positive results.



Layers 0, 3, 6, and 9 of the Gaussian pyramid



Layers 0, 1, 3, 6, and 9 of the equivalent Laplacian pyramid (after dividing by 2 and adding 128)

In the Gaussian pyramid, it is evident that the image becomes progressively more blurred as we move to deeper levels. This happens because each level of the Gaussian pyramid applies additional blurring and downsampling, effectively smoothing out details and reducing the resolution.

In contrast, in the Laplacian pyramid, the focus is on capturing the edges and finer details at each level. As a result, the edges appear more pronounced but also slightly blurred, particularly in the higher levels of the pyramid. This is because the Laplacian pyramid represents the difference between consecutive Gaussian levels, highlighting the high-frequency components (edges and fine details).

Conclusion

The exercise demonstrated the practical utility of Gaussian and Laplacian pyramids for image manipulation. These techniques provide a robust framework for blending, hybridization, and other frequency-based image processing tasks. By developing and refining these algorithms, the exercise highlighted the importance of multi-scale representations and laid the foundation for advanced image processing techniques.