Hybrid Images & Blending

Avinoam Nukrai

Introduction

This project aims to seamlessly blend two images using pyramid blending techniques, alongside generating their Hybrid image. It involves constructing Laplacian and Gaussian pyramids for the images and mask, manipulating these pyramids, and reconstructing the blended image. Key techniques utilized include Laplacian and Gaussian pyramids, image convolution, and padding.

Algorithms

In this section I will explain and detail the algorithm for each task, the Blending Images task and the Hybrid Image task, visualizations will be included also.

Blending Images Algorithm

Algorithm Description:

The algorithm is built from several steps whose general purpose is to produce a Laplacian pyramid for each image and then connect the pyramids so that we take the desired amount of pixels and frequencies from our input images, according to some mask, the steps are as following:

1. Decomposing 2 images into RGB color channels, **for each** color channel do the next steps.

2. Construct Laplacian pyramids L_1 and L_2 for the input images im1 and im2, respectively by:

2.1 Construct Gaussian pyramid Gn_1 and Gn_2 for im1 and im2, each level will be reduced:

2.1.1 First the image will be blur by gaussian filter

2.1.2 Second we will sample each 2nd pixel in the last level.

2.2 Construct L_1 and L_2: $L_1 = Gn_1 - \text{Expand}\{Gn_{1+1}\}$, $L_2 = Gn_2 - \text{Expand}\{Gn_{2+1}\}$.

2.2.1 Expand function works as follows:

2.2.1.1 Resize the level by factor of 2 and pad it with zeros in each 2nd pixel.

2.2.1.2 Blur the resized image for overlay the pixel values with zero pixels.

3. Construct a Gaussian pyramid G_m for the provided mask.

4. Construct the Laplacian pyramid L_{out}: $L_{out}[k] = G_m[k] \cdot L_1[k] + (1 - G_m[k]) \cdot L_2[k]$.

5. Reconstruct the resulting blended image from the Laplacian pyramid L_{out} and return it.

5.1 Expand as needed all the Laplacian images in the pyramid.

5.2 Multiply the expand lap images with the right index in the given coefficients array.

5.3 Sum all the results layers and return.

6. Chaining all the results of the 3 channels and normalizing pixel values with threshold of 1.

As can see, I used division into color channels, pyramids and their analysis as we learned in class.

Algorithm Implementation Details (by steps):

We starting the all process in the rgb_images_blending() function which gets the next parameters: 2 RGB images (im1, im2), binary mask (bm), maximum levels of the pyramids we'll build (max_levels), the filter size of images (filter_size_im), and the mask filter size (filter_size_mask).

1. Enter rgb_images_blending() function and decompose each image to 3 RGB color channels then for each channel of the two images we do pyramid blending with pyramid_blending() function, then we create the final_blended_rgb which is np.dstack of the 3 pyramid blends.

2. Enter to the pyramid_blending() function which gets the same parameters as the function in step 1 and construct Laplacian pyramids L_1 and L_2 for the input images im1 and im2, by:

2.1 For each image we enter to build_laplacian_pyramid() function which get as params - image, max_level + filter_size_im, and returns the Laplacian pyramid + filter_vec used for pyramid construction. build_laplacian_pyramid() wroks as follows:

2.1.1 Enter to build_gaussian_pyramid() which gets the same params and returns the gaussian_pyramid + gaussian_filter (used for pyramid construction), the build_gaussian_pyramid() works as follows:

2.1.1.1 First we create the gaussian filter with create_filter() which is trivial (doing enough times convolutions with the the (1,1) kernel.

2.1.1.2 Second, we iterate the on the max_levels param and calculates the gaussian image be reducing the two layers using the reduce() function, which works as follows:

2.1.1.2.1 Blur given image with given filter.

2.1.1.2.2 Sample each 2nd pixel and return.

2.1.1.2.3 Return reduced_image.

Then add the gaussian image to the array pyr.

2.1.1.3 Third, we return the layers array – pyramid and the gaussian filter.

2.2 Creates the Laplacian pyramid by using np.subtract() while iterating the all gaussian pyramid we have for the image and expending the i+1 layer by using the expand() function works as follows:

2.2.1 Pad the image with zeros with pad_image_with_zeros():

2.2.1.1 Create an array in 2 times size of image.

2.2.1.2 Insert image into the even pixels of array.

2.2.1.3 Return the pad_image.

2.2.2 Blur the pad Image with the blur_filter with get_blur_image().

2.2.3 Return the expand image.

2.3 Return the Laplacian pyramid and the gaussian filter we used.

3. Calculate G_m, the gaussian pyramid (2.1.1) of the mask, it will help us "take a lot of the pixels around the cat of the mask and little bit less from the unwanted pixels but in graduated manner" .

4. for k in range(len(laplacian_pyramid)) - $L_{out}[k] = G_m[k] \cdot L_1[k] + (1 - G_m[k]) \cdot L_2[k]$ when L_1 and L_2 are the laplacian pyramids of image1 and image2 we got from step 2.

5. we call laplacian_to_image() on L_{out} pyramid, will do that as follow:

5.1 Expand (2.2.1) as needed all the Laplacian images in the pyramid according to the dim of the given laplacian pyramid as an input, then add the expand version to array.

5.2 Multiply each expand lap with the right index in the given coefficients array.
5.3 Sum all the results layers and return it.

6. After we got the final blended image, we return to the father function rgb_images_blending() their we chaining all the results of the 3 channels, normalizing the pixels with threshold of 1, and return the final image (we can now plot it!).

Additional Details (Libraries, Hyper-Parameters/Thresholds & Challenges):

I utilized essential libraries including os, numpy, matplotlib, scipy, skimage, and imageio for IMPR tasks such as reading, analysis, and visualization. Hyperparameters like minimum image dimension (set to 16) were employed to ensure effective operations. Normalization of pyramid levels was achieved by multiplying with the dimension coefficient. Adjustable parameters like maximum pyramid levels and Gaussian filter size were set for result accuracy. A pixel threshold of 1 was applied in the final blending to ensure smoothness and minimize unwanted artifacts. MAX_GS = 255 was used for grayscale conversion. Overcoming the challenge of algorithm design, I outlined steps, conducted unit testing, and fine-tuned parameters for optimal outcomes.

Hybrid Image Algorithm

Algorithm Description:

The algorithm takes two images, X and Y, and computes their hybrid image, where viewing up close reveals X and from a distance reveals Y. It builds upon the previous algorithm's implementation, utilizing the same technology in a slightly different manner. The hybrid image is created by combining Laplacian pyramids of the input images.

1. Read the input images.
2. Build Laplacian pyramids for the two images.
3. Combine Laplacian pyramids to create the hybrid image, $L_{\text{out}} = L_{\text{ap1}}[:2] + L_{\text{ap2}}[2:]$.
4. Reconstruct the hybrid image from the combined pyramid.
5. Normalize the hybrid image.
6. Create the actual hybrid image and return it (ready to plot!).

Algorithm Implementation Details (by steps):

Step 1 is trivial. Step 2 described already in 2.1 of the last algorithm. In step 3 we takes the 2 first layers of the first image Laplacian pyramid and the other from the second images Laplacian pyramid. Then, we reconstruct in step 4 by the laplacian_to_image() function described already in step 5 of the last algorithm, then we normalizing the result image for smooth result and finally we create the actual image by multiply all pixels with MAX_GS = 255 and return it.

Additional Details (Libraries, Hyper-Parameters/Thresholds & Challenges):

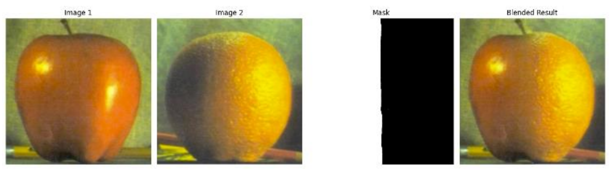
Utilized identical libraries, hyperparameters, and thresholds from the blending algorithm. Challenges included understanding the operation initially; initially attempted Gaussian blurring and pasting images with unsatisfactory results. After implementing the first algorithm, discovered that combining pyramids into the Laplacian likely yields more accurate results.

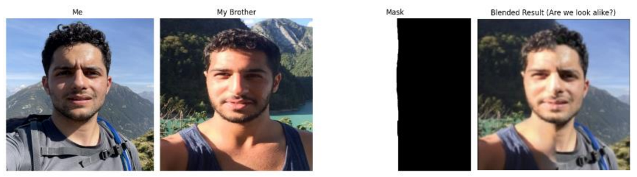
Algorithms Difference:

The difference between the two algorithms is in the complexity of their implementation. While the first algorithm uses a binary mask in order to get the perfect combination between the pixels in the blending seam, the second algorithm only combines between the changing frequencies of the images, that is, between the layers of the 2 Laplacian pyramids and this is very simple after the blending.

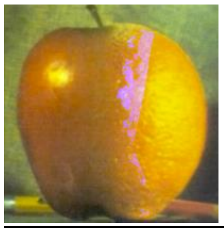
Results

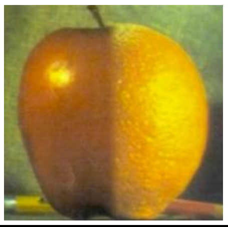
2 Blending examples:

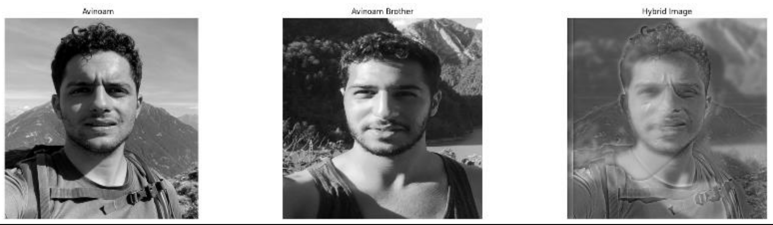




Hybrid Image 2 intermediate blending results:

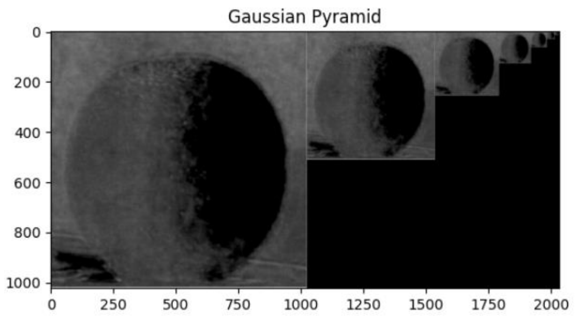


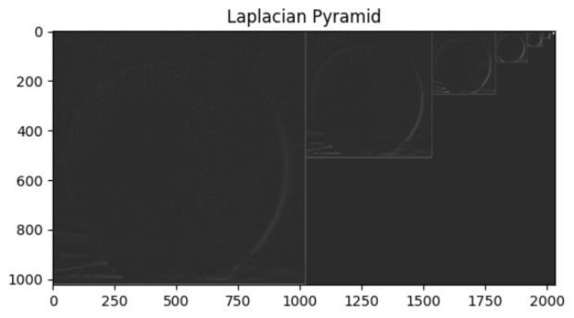




The damage of the first result is due to the fact that we increased the size of our Gaussian filter, as a result we blur the mask and the images very strongly which leads to the side effect of the strange color. The damage of the second result is due to the fact that we set the maximum size of the pyramids to be 4, therefore the combination of the images is not deep enough and the result is quite bad.

Pyramids:





Visualization of these pyramids aids in comprehending the algorithm's operation. In the Gaussian pyramid, as we ascend, images become smaller and more blurred. Subtracting successive levels yields image edges, akin to Laplacian pyramid levels. In the Laplacian pyramid, higher levels resemble the original image more closely. This understanding guides our algorithm: by blurring the mask, the two images blend seamlessly, with pixels gradually transitioning at the seam for a pleasing result.

Conclusion

In conclusion, in this project I learned how to really play with pictures. Using the cool tools we learned in class I implemented image blending as well as a combined "magic" image, for the first time I felt what it's like to play with images from the initial stage of their analysis to the stage where there is a literally really cool result.