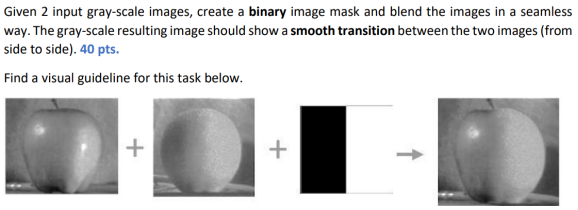
Image Blending using Laplacian Pyramids Report

Sara Carolina Gómez Delgado || 0226594
Luis Eduardo Robles Jiménez || 0224969

Part 1



General Process

1. Read images
2. Create a Gaussian pyramid for each image.
3. Create its Laplacian pyramid *(for each image).*
4. Join both halves *(orange + apple)* for each Laplacian pyramid layer
5. Reconstruct the image.

Each step explanation

Gaussian Pyramid: *(do this for both apple and orange images)*

1. In order to get the Gaussian pyramid for each image, we first create a copy of the original image so we don't modify it.
2. Create an empty array *(here we will store the Gaussian pyramid layers)*. Let's call it "*xGauss*"

3. Append the copy to this empty array (*xGauss*).
4. Downsample the copy `n` times *(where n = # of layers we want this Gaussian pyramid to have)* and append the result to *xGauss*.

Laplacian pyramid: *(do this for both apple and orange images)*

1. Store *xGauss[n]* in a new variable (we do *xGauss*[n] to get the smallest image in the Gaussian pyramid).
2. For each element in *xGauss*, we will find its corresponding Laplacian layer. To do this, we need the next formula:

   ```
   laplacian_layer[i] = xGauss[i-1] - current_level_xGauss
   ```

3. Then, we append this *laplacian_layer*

Join both halves for each Laplacian pyramid layer:

1. Create a new variable (here we will store each blended laplacian pyramid layer)
2. Go through the laplacian pyramid of the orange and apple images.
3. Join each apple Laplacian pyramid layer with its correspondient orange one.

Reconstruct the image

1. Go through the resultant pyramid from the last step.
2. Upscale each layer and add the result to the current resultant pyramid from de last step.
3. Show the last result.

Conclusion:
From the first part, we can conclude that we needed exactly **6** pyramid layers, otherwise, the smaller this number is, the rough the blend gets. On the other hand, the greater it is, the blurrier the image gets. This ocurrs because the Gaussian + Laplacian pyramid has the effect of blurry the image as it is downscaled (reducing pixels) and upscaling it (trying to retrieve the original image data).

Part 2

Part 2 is very similar to part 1, in theory. This time, you will create your own 2 input color images. Be creative and have fun with it, you could use your camera phone, but you need to consider an ideal angle and light settings (right?).

Next, create a **gray-scale** image mask (how will you do it?). Consider the characteristics of you input images.

Once everything is ready, blend the images in a seamless way. The **color resulting** image should show a **smooth transition** between the two images). 60 pts.

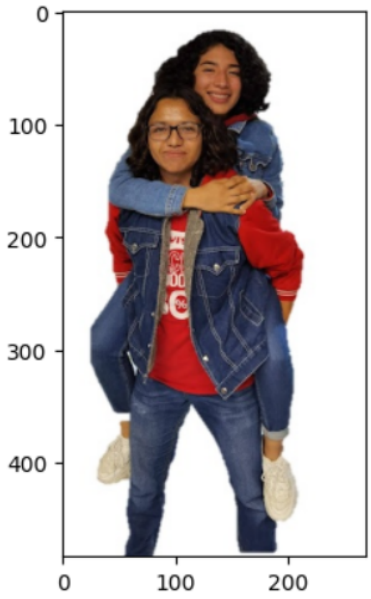
General Process

1. Read images.
2. Construct mask.
3. Create a Gaussian pyramid for each image and the mask.
4. Create their Laplacian pyramids.
5. Join both images for each Laplacian pyramid layer.
6. Reconstruct the image.

Each step explanation

Read images:

Have you ever wondered how good would a tattoo look on your skin? Let’s find out if this is the right match for Lalito’s face.



Construct the mask:

1. Having the right mask and images is a crucial step for this task. So, a lot of preprocessing was needed in order to have the tattoo ready for this person's face.

   Some of the operations performed over the image on the left were:

   a. Crop the original image to get rid of the upper part of the tattoo.

   b. Resize the image in order to correctly fit on the person's cheek.

   c. Create a copy of the face to give us an idea of what the tattoo will look like.

   d. Threshold the tattoo to start creating the mask.

   e. Add an alpha value with a vanishing effect to give extra detail to the tattoo (gotta show off the gray-scale mask).

   f. Rewrite the mask to avoid having values greater than 1.0.

   g. Use a kernel to smooth out the vanishing effect.

   h. Apply a slight blur to the whole mask in order to have better results.

   i. Add a padding to both the tattoo and the mask in order to match the face size.

2. As it's possible to see, the whole process had to be figured out by trial and error. It required a lot of time, effort and several visits to previous homework and techniques.

Gaussian Pyramid:

- Once the images are ready, the blending part is pretty similar to the first part.
- `n` Gaussian pyramids are created per image in order to start the blending process.
- This is achieved by using the cv2's functions. `pyrDown` and `pyrUp` where the latter increases the image size and the other reduces it by powers of two.

Laplacian pyramid:

- This pyramid is created in order to keep the details of the images to achieve a successful blending.
- It comes from the subtraction of different layers of the Gaussian pyramids, and it's specifically done to keep track of important information that is lost.

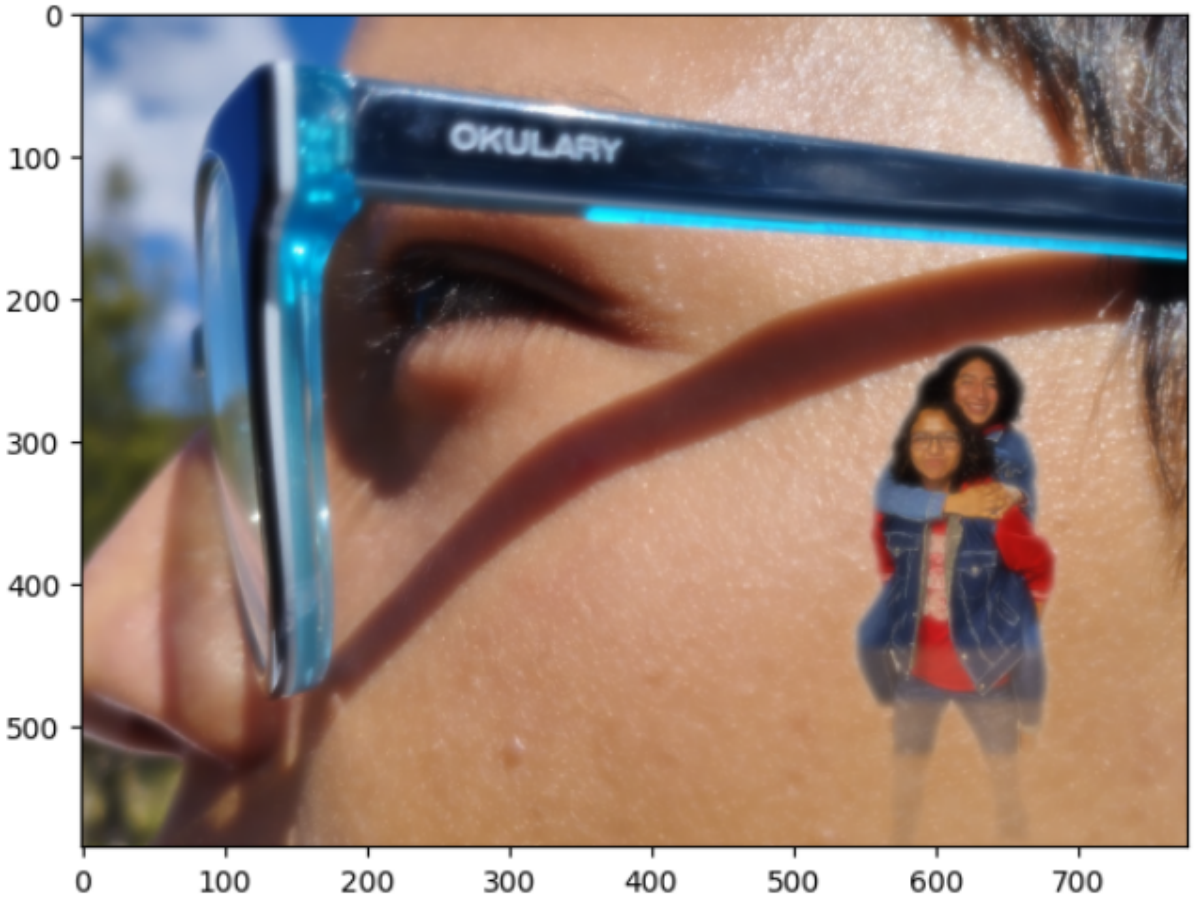
Join both both images using each Laplacian pyramid layer:

- This step is the one that differs the most from the first part, and it is because as the mask is gray-scale, the blending step has to be completed by applying a weighted sum.
- The formula for this step is applied to every channel of both images and using the values of the mask.

```
lF = np.add(np.multiply(lS, (1 - gM)), np.multiply(lT, gM))
```

Where lS and lT are the respective Laplacian layers.

Final result



💡 **Conclusion.**
It’s extremely impressive the quality of the result given by this approach and how dependant it is of the number of layers. Also, the more we learn about the topic, the more useful it gets and we’re sure there are a lot of applications for this technique. Now, even though the execution worked and it was well implemented, we wouldn’t recommend this tattot to that person, glad we were able to see it prior to doing it.