CMPEN 454 - Project 1

By: Rui Zheng, Jacob Minnich, Chenning Zhang, Nelson Huang



**a) Summarize in your own words what you think the project was about. What were the tasks you performed; what did you expect to achieve?**

The project seeks to test our understanding with various image processing techniques by allowing us to write a computer algorithm that uses those techniques. The final purpose is to generate a blending picture. Our project is divided into four sub tasks.

1. Blur the image
2. Downsample the image
3. Upsample the image
4. Blending with two image

One image processing technique we worked on was to generate a multi-scale pyramid representation of an image. The goal of this task was to apply an enhancement algorithm on our image by subtracting our original image with a blurred version of it. By applying this enhancement algorithm, we expect the resultant image to have a lower high-frequency spatial information. We start off with a 256x256 size image. First, we applied a Gaussian filter of [1 4 6 4 1] / 16 and it’s transpose to approximate a Gaussian filter with sigma = 1 on the image. Next, we find the differences of the filtered image with our original image to build Differences of Gaussian pyramid. After that, we downsample the blurred image, and repeat the process multiple times to a resolution of 32x32. Originally we tried to downsample to 4x4 but due to image properties, upsampling from a lower resolution create visible black edges that greatly affect upsampling quality, so instead we go back to 32x32 as it produces a more stable result. Lastly, we upsample from the 32x32 image and compensate it with the difference of Gaussian pyramids at each resolution level. The resultant image after we applied this enhancement algorithm looks just like our original image.

Another task we worked on was to blend portions of two images together. To do so, we first need two images with the same dimensions. A portion of the blending process requires us to create a Gaussian pyramid, which we had already done so when we wanted to generate a multi-scale pyramid representation of an image. We then created a binary mask and we combine the two images together by computing a value from the equation:

G = GA .\* (1-W) + GB .\* W

Where GA= first image with the DoG, GB = second image with the DoG, W = binary filter. Lastly, we subtract from it the blended DoG of the 2 images, which are created in the same formula at each level, and repeat the process until we reach the original resolution.

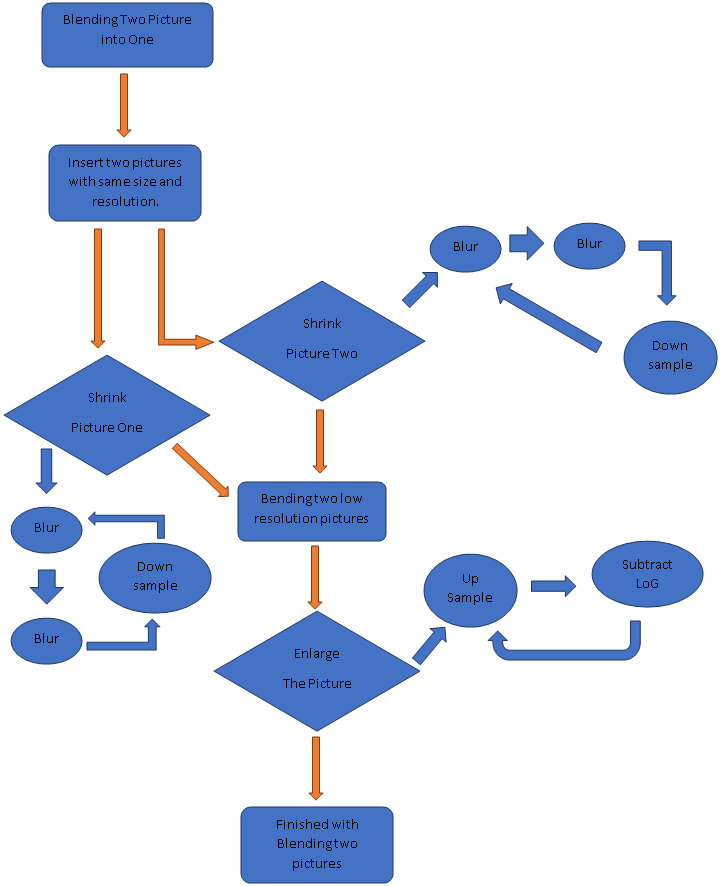
**b) Present an outline of the procedural approach along with a flowchart showing the flow of control and subroutine structure of your Matlab code. Explain any design decisions you had to make. For example, what data structure did you use to represent Gaussian and Laplacian pyramids? Did you make any modifications to the basic algorithms presented in the Crowley or the Burt and Adelson papers?**

// we need show some our code on each steps

I created a separate word file. I’ll submit it soon.

We only use matrix instances to represent Gaussian and Laplacian pyramids. We didn’t implement them in arrays of matrices because we chose not to go too far with our resolution, so we just used the code for our early experiment with Matlab. Algorithms are not modified but we didn try different padding option, though no visible change of image quality is observed.

Flowchart:

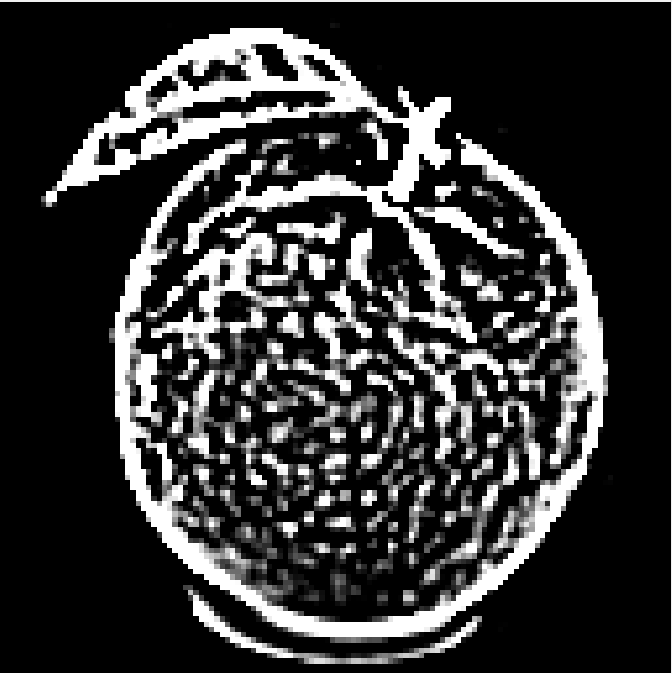


**c) Image coding milestone. What do you observe about the behavior of your code for pyramid decomposition and image regeneration? Does it seem to work the way you think it should? For a sample image, can you regenerate the original image after decomposing it into a low-res Gaussian and Laplacian pyramid? Show some intermediate results of the process, such as images from different levels of the Gaussian pyramid and the Laplacian pyramid.**

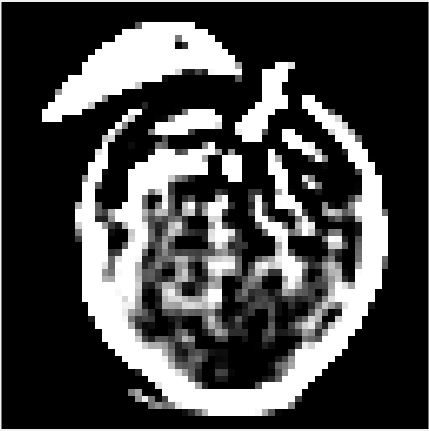
// Could someone attach each steps pictures, each blur and down /up sample.

DownSample - Reduce the size of an image by two in the row and column dimensions. We initial have an image of Orange with 256\*256.( Figure 1 ) To reduce to half of the resolution of it, we need to blur it 3 times to void high-frequency after downsample by Gaussian pyramid filter, afterward we took the left top pixel on every 2\*2 pixel and form a low resolution photo, and get a 128\*128 image of Orange. Same process on next downsample, downsample until 32\*32 is our lowest resolution.

UpSample - Increase the size of an image by two in the row and column dimensions. We upsize our lowest resolution picture ( 32\*32 ) to 64\*64, and extend one pixel to four pixel and apply DoG filter in it to generate double resolution picture. Same steps on next upsample with higher resolution DoG??LoG filter, and upsample until we get the same resolution as we original input.

128x128 image after downsampling, DoG 1, DoG 2

64x64 image after downsampling, DoGs



32x32 image after downsampling



Original Image vs. Image upsampled

**d) Experimental results. Show examples output by your blending application – the images chosen, the binary region mask, and the resulting blended image. Do some examples work better than others? How did you choose images to use for the blending application? How did you generate a binary mask image? Did you develop any rules of thumb by trial and error to tell you what to look for when choosing images that would be good to blend?**

To generate a blending image, we choose two pictures with the same size and resolution. The purpose of this is to reduce the errors during the downsample and upsample. When we apply the same process of Gaussian filter and subtract LoG on both images, the results still have same resolution and size. And easy for blending. We choose two pictures, apple and orange, both with 256\*256 resolution ( Figure 1, Figure 2 ). Afterward we use Gaussian filter to blur the image and downsample the image, the resolution of both image reduce to 32\*32. We use binary mask to choose which part of each pictures we needs to blending. We generate a binary mask which take left side of picture and right side of another, to achieve that we input ‘1’ on a full 0 filter, and input ‘1’ from the right side to the middle. We take left side of Orange and right side of Apple. Combine two images with their lowest resolution we have ( 32\*32 ). Now we got a blended picture with 32\*32, then we use LoG subtraction to upsample the image to 256\*256 resolution ( Figure 3 ).

In our case, some images do better work than others. If we choose to use two different resolution pictures on the project, for example, Orange with 256\*256 and Apple with 128\*128, the lowest resolution of two pictures will not be same, same result happen on upsample. The final result will show similar with the correct one, but due to the different resolution, Orange part will look clearer than Apple part. And also blending with black and white photo will much easier than color one. Because everytime using Gaussian filter or LoG, colored pictured needs to take triple times, one for red, one for green, and one for blue. However, black and white photo only need take ones.

The binary mask we use is half-half. We create a all-zero picture and input ‘1’ from right to left until the middle. We create a 32\*32 binary mask, same as lowest resolution images. And Generate different binary mask with the same form but different resolutions. In every level, the binary mask follows upsample and fit with image.



Figure(1) Figure(2)



Figure(3)

**e) Document what each team member did to contribute to the project. It is OK if you divide up the labor into different tasks. This is also where you let us know if any of your teammates were slacking off on this project.**

We divide this project into four sub tasks: Blur, Downsample, Upsample, and Blending. Each of us take one of them, Chenning took Blur, Nelson took Downsample, Ray took Upsample, and Jacob took Blending. After we finished each parts, we gathering all code into one file to test its working and debug.